Google App Engine

Développer sur le Cloud en Java

LTE Consulting
www.lteconsulting.fr

Qu’est-ce que le Cloud ?

- Pas nouveau: plutôt un vieux rêve devenu réalité. Maîtriser un gigantesque ensemble d’ordinateurs.
- Gartner: “Le cloud computing est une façon de programmer par laquelle un nombre massivement scalable d’éléments informatiques sont fournis “tels un service” sur internet, et à de nombreux clients”. “Cloud computing is a style of computing where massively scalable IT-related capabilities are provided ‘as a service’ across the Internet to multiple external customers”.
- Massivement scalable,
- Vu comme un service.
- Paiement à la demande, pas d’engagement.
- Disponible.

Les offres du Cloud

Type de marché	Offres	Examples
Software as a service	Sites web RIA Office productivity Applications clientes se connectant au Cloud	Flikr Myspace Cisco WebEx office Gmail
App-components as a service	APIs d'intégration Services Web « assemblables »	Yahoo! Maps API Google Calendar API Amazon Flexible Payments Service, …
Software-platform as a service	Database & Data store Message Queue App servicer	Google App Engine MSQL Server Data service Engine Yard, Force.com
Virtual-infrastructure as a service	Serveurs virtuels Disques logiques Réseaux VLAN Systems Management	Akamai Amazon EC Joyent accelerator Nirvanix Storage Delivery Network
Physical infrastructure	Hébergement Collocation ISP Unmanaged hosting	OVH GoDaddy.com Rackspace Savvis

Citations

- The interesting thing about Cloud Computing is that we've redefined Cloud Computing to include everything that we already do. . . . I don't understand what we would do differently in the light of Cloud Computing other than change the wording of some of our ads. (Larry Ellison, Oracle CEO, 2008)
- A lot of people are jumping on the [cloud] bandwagon, but I have not heard two people say the same thing about it. There are multiple definitions out there of "the cloud." (Ansy Isherwoord, HP European Sales president)
- It's stupidity. It's worse than stupidity: it's a marketing hype campaign. Somebody is saying this is inevitable — and whenever you hear somebody saying that, it's very likely to be a set of businesses campaigning to make it true. (Richard Stallman)

Qu'apporte le Cloud ?

- L'illusion d'une quantité infinie de ressources disponibles à la demande
  - Pas besoin de plan de provisionnement à long-terme.
- L'élimination d'un engagement up-front
  - Les clients peuvent commencer petit et grossir au fur et à mesure besoin.
- La possibilité de payer les ressources selon le besoin, à court terme (CPU/heure, Go/jour).

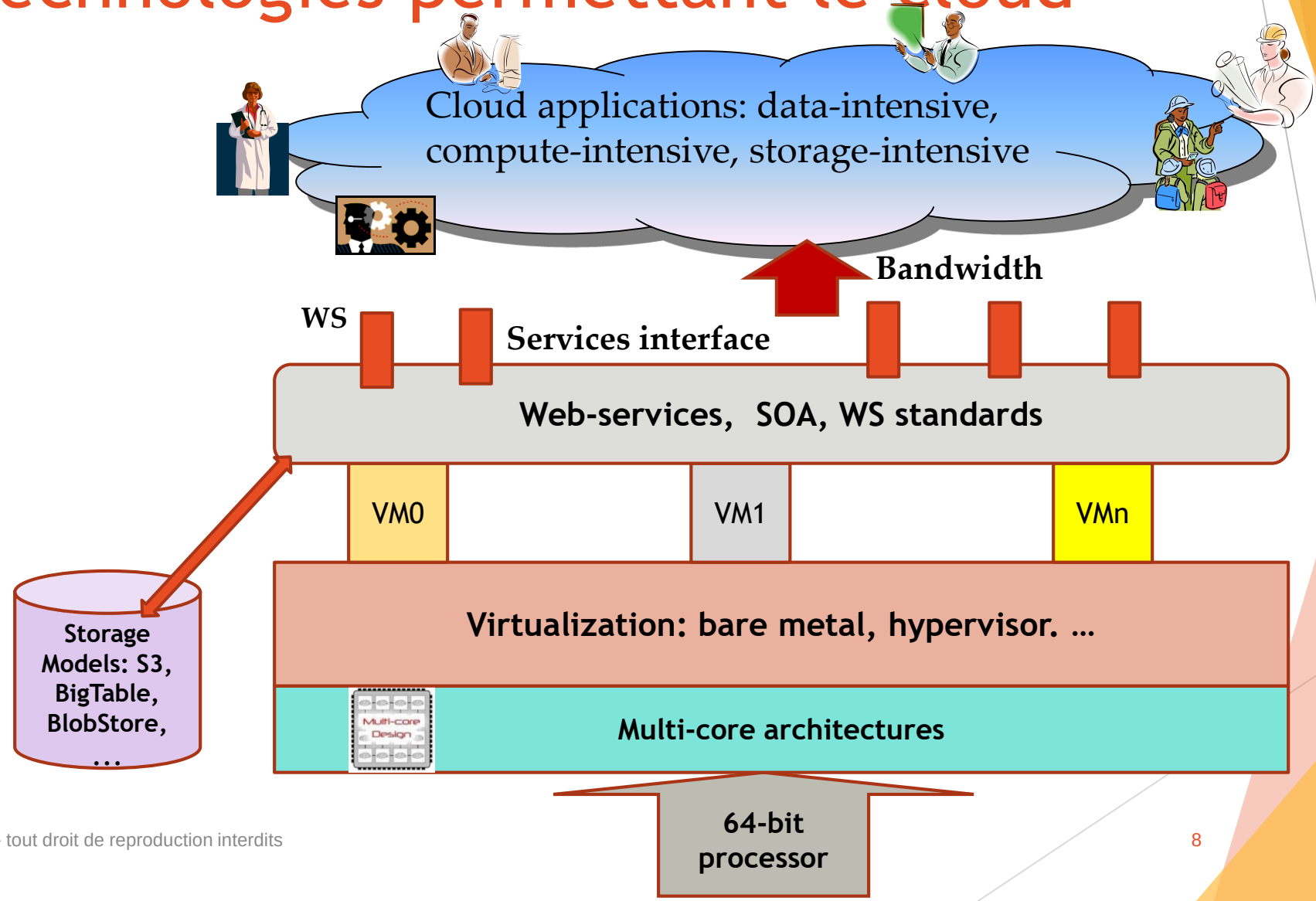
Et aux fournisseurs de Cloud ?

- Faire beaucoup d'argent: faible prix de location mais les fournisseurs ont des gros avantages avec les quantités qu'ils achètent.
- Exploiter les investissements existants. Par exemple, AWS a été initialement développé pour Amazon en interne. AppEngine tourne sur l'infrastructure de Google.
- Défendre une franchise: As conventional server and enterprise applications embrace Cloud Computing, vendors with an established franchise in those applications would be motivated to provide a cloud option of their own. For example, Microsoft Azure provides an immediate path for migrating existing customers of Microsoft enterprise applications to a cloud environment

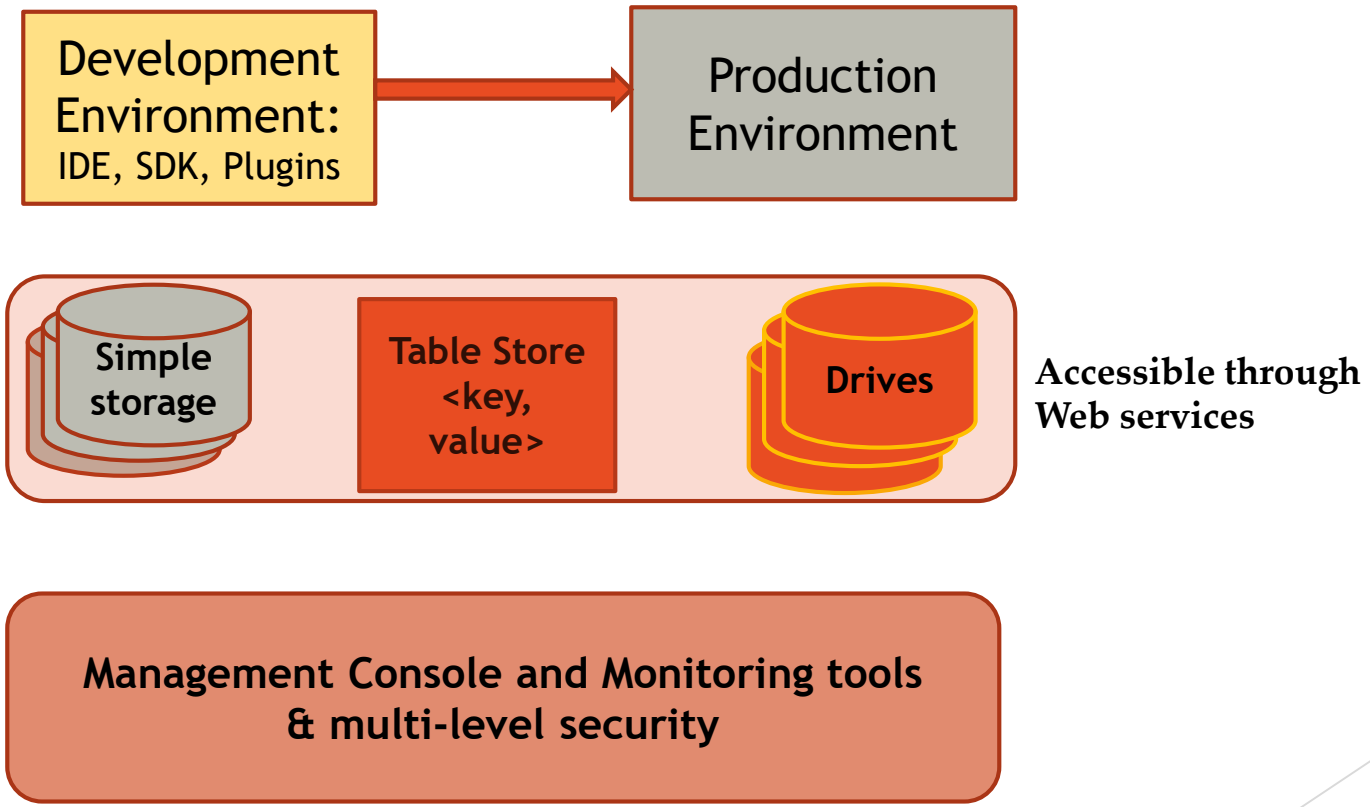
Les technologies permettant le Cloud

- Parallelisme croissant: 2x plus de coeurs par génération de processeurs, réseaux et stockage redondants
- Virtualisation: processing, storage, bandwidth
- Commodity components: serveurs X86, HDD, ethernet. Logiciels OpenSource=liberté d'adaptation et de personalisation. Data centers à bas prix divisent par 5 à 7 les coûts.
- Outsourcing croissant: "By 2012, 80 percent of Fortune 1000 companies will pay for some cloud computing service, and 30 percent of them will pay for cloud computing infrastructure." Gartner
- Allez voir le Open Compute Project.

Les technologies permettant le cloud



Architecture standard



Les modèles de déploiement

- Cloud privé (interne)
- Cloud communautaire: besoins spécifiques partagés
- Cloud public: location au public
- Cloud hybride: composition de plusieurs clouds, liés par une couche d'interopérabilité.

Cas d'utilisation du Cloud Computing

- A la demande: évite le provisionnement de ressources sur ou sous évalué. Passage du logiciel « fait main » à une fabrique logicielle.
- Calculs intensifs: possibilité d'indexer et d'analyser des énormes jeux de données (//).
  - Archivage et analyse de larges jeux de données à prix faible.
- Accessibilité: fait levier sur les prix de plus en plus bas des CPU et du stockage de données. Le défi est d'équilibrer l'adhérence à la plateforme par rapport aux innovations. Cloud serviced clients.

Compromis…

- Une application a besoin: d'un modèle de calcul, d'un modèle de stockage et d'un modèle de communication.
- Le multiplexage des ressources nécessaire à « l'illusion cloud » demande qu'elles soient virtualisées pour que soient cachées l'implémentation du multiplexage et du partage.
- Donc on a un compromis entre le niveau d'abstraction présenté au développeur et le niveau de gestion des ressources.
- => EC2 -> Azure -> **AppEngine**

Le contexte : énormes jeux de données

- Analyser des énormes jeux de données est devenu essentiel. (astronomie, biomédical, business intelligence, deep learning…)
- Nous sommes dans une économie de « connaissance »
  - Les données sont importantes pour une entreprise,
  - Découverte de connaissance,
  - Modèles de calcul complexes,
  - Intelligence, intuition sur les données.
- Grosse différence avec les données généralement utilisé en SGBD transactionnel : Write Once Read Many.

Opportunités pour le Cloud

- Application interactives mobiles. (disponibilité, larges jeux de données sous-jacent).
- IOT : volume des données générées.
- Traitements par lots parallélisés: analyse de TeraBytes de données. Par ex: The Washington Post utilisa 200 instances EC2 pour OCR de 17481 pages en 9 heures. MapReduce.
- Attention au coût de transfert des données vers et depuis le cloud !
- L'avènement de l'analyse: transition des calculs transactionnels aux calculs analytiques.
- Compute-intensive applications. Ex: MathLab

Google App Engine

Qu’est-ce que Google App Engine ?

- Service cloud de type Paas permettant d’executer des applications Web sur l’infrastructure de Google
- Environnement d’exécution pour applications web.
  - Prend en charge les requêtes HTTP(S). Fonctionne à merveille pour le Web et l’AJAX (et quelques autres services).
- Espace de stokage
- Google AppEngine fournit une alternative aux autres offres de déploiement cloud. Son intérêt reside dans l’automatisation de ses fonctions de load balancing et de scalabilité fournies d’office.
- Configuration très simple
  - Pas besoin d’ajustement de performance.
- Scalable
  - Nombre “inifini” d’applications, de requetes/seconde, de capacité de stockage
  - API très simple, presque stupide.
- Sécurisé

L’environnement de développement Google App Engine

- La plateforme supporte les applications écrites en Java, Python, Go et PHP *(plus Node.js et Docker sur les vm gérées)*.
- Un SDK est nécessaire pour développer une application App Engine.
- Le SDK fournit un environnement de déploiement local simulant l’exécution sur le cloud.
- Des outils de déploiement de type « one-click » sont fournis et permettent de gérer très facilement le déploiement de votre application.

Modèle de tarification

- Budget max quotidien
- Par resources consommées :
  - Bande passante : $0.12 / GB
  - Instance : ~ $0.16 / heure
  - Données : $0.24 / GB / mois (Datastore), $0.13 (Blobstore)
  - Channel : $0.01 / 100 cannaux
  - Email : $0.0001 / envoi
- Par nombre d'opérations :
  - Ecriture : $0.1 / 100 000 opérations
  - Lecture : $0.07 / 100 000 opérations
  - Get=1 lect., nouv Entity put=2 ecr. + 2 ecr/valeur indexée + 1 ecr/valeur index composite

Tarification

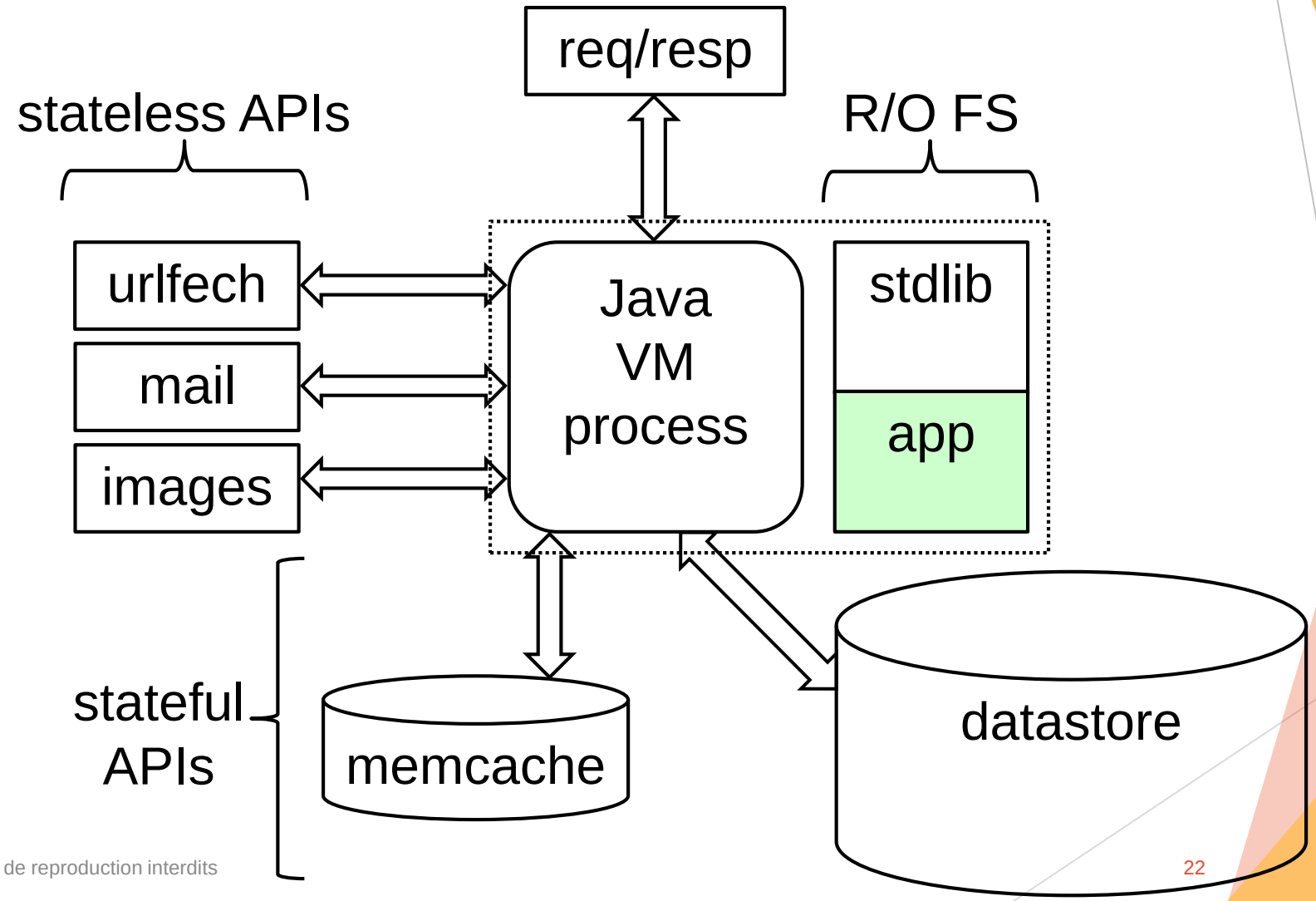
Les quotas

- Tout ce qu'une application fait est limité par les quotas
  - Nb de requêtes, bande passante utilisée, CPU, appels au Datastore, espace disque, emails envoyés, même les erreurs !!!
- Lorsqu'un quota est dépassé, l'opération en question est bloquée pour ~10 minutes
- Les quotas gratuits sont fait pour qu'une application bien écrite puisse survivre à des pics d'utilisation modérés.
- Les quotas permettent de servir un très très grand nombre de petites applications
- Les grosses applications ont besoin d'augmenter leurs quotas (manuellement)

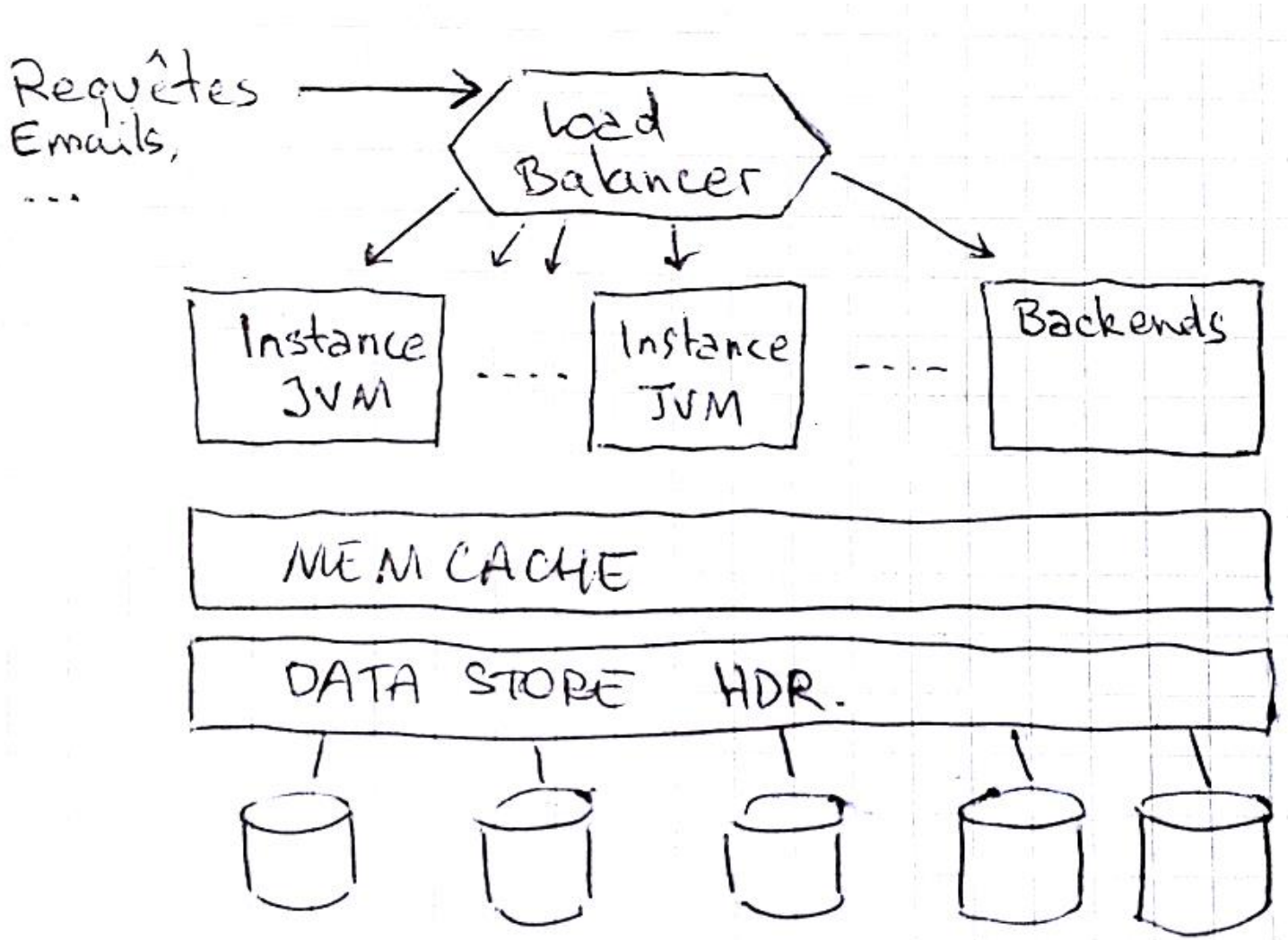
Quotas et Limites

- Exception : OverQuotaException
- Quotas quotidiens
- Blobstore : pas de limite
- Datastore : pas de limite en taille, 200 indexes max.
- Requêtes : entrée : -, sortie : 14GB
- Tâches : 20M d'appels
- URLFetch : 46M d'appels
- Channel : 91M appels, 32000 appels/minute
- Conversion : 100 conversions
- Mail : 1,7M d'envois, 4900 envois/seconde

Architecture



Architecture en exécution



Mise à l'échelle

- Applications légères : beaucoup sur un serveur.
- Applications lourdes : beaucoup de serveur pour 1 app.
- API stateless faciles à répliquer.
- Memcache est trivial à « sharder ».
- Datastore construite sur BigTable, conçue pour passer à l'échelle.
  - C'est une abstraction au dessus de BigTable
  - L'API est conçue (et contrainte) afin de passer à la mise à l'échelle:
    - Pas de jointures
    - Recommandations: dénormaliser et précalculer les jointures.

Mise à l'échelle selon les besoins de l'application

- Pas besoin de configurer les ressources nécessaires,
- Un CPU peut gérer beaucoup de requêtes par seconde,
- Les applications sont mappées sur des CPU:
  - Un pus par application, beaucoup d'applications par CPU
  - La création d'un nouveau process se fait en clonant un modèle générique de process puis en chargeant le code de l'application (en fait les clones sont pré-créés et attendent dans une queue)
  - Le process reste un peu pour gérer d'autres requêtes
  - Puis il est tué
- Les applications gourmandes sont associées à plusieurs CPU (automatiquement)

Sécurité

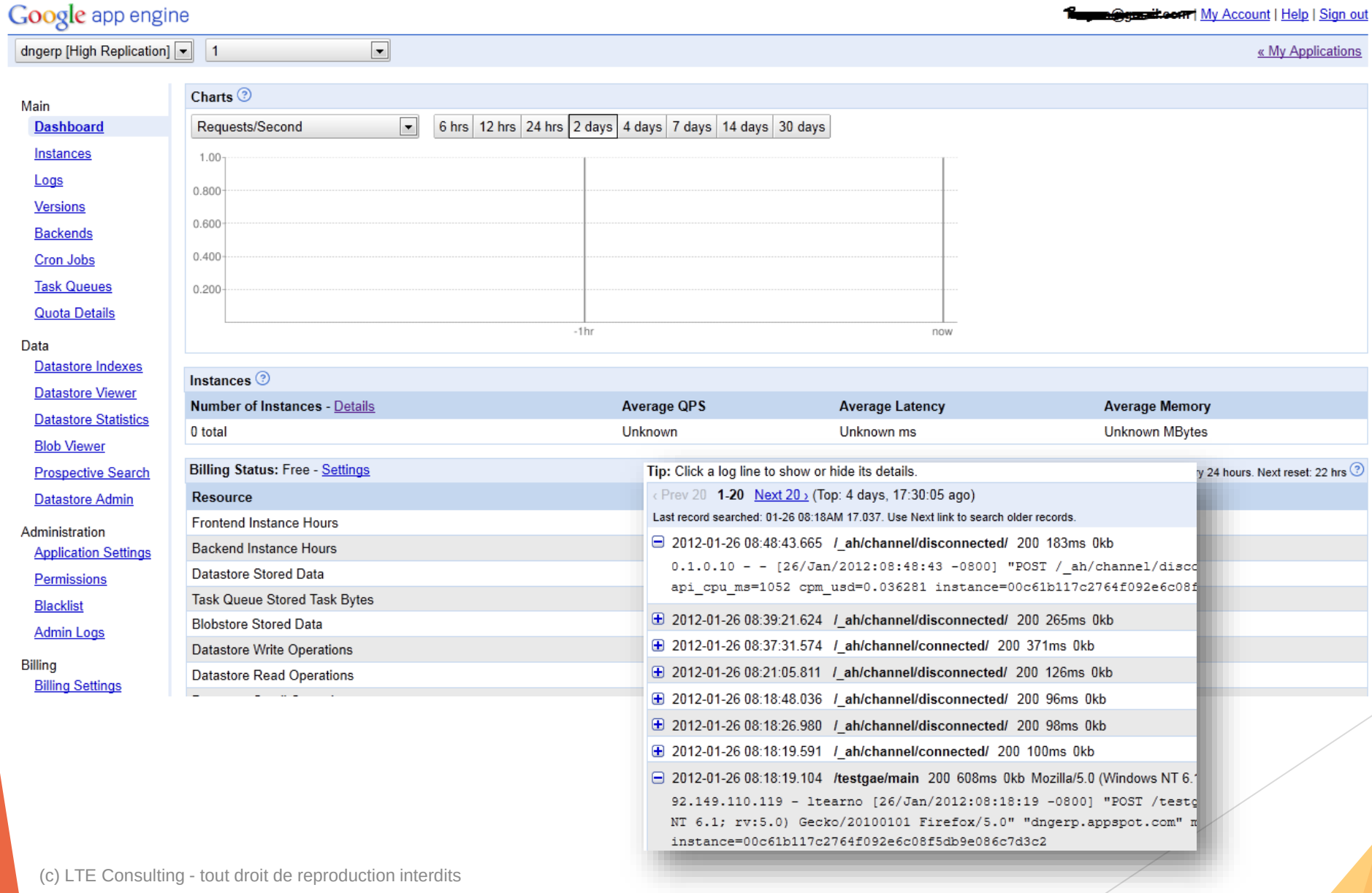
- Empêche les pirates de s'infiltrer, on profite de l'expertise Google
- Contraint les fonctionnalités « OS »:
  - Pas de process, thread, ni de liaison dynamique
  - Pas de sockets (URLFetch API)
  - Pas d'écriture sur disque (utiliser DataStore)
- Limitation de l'utilisation des ressources:
  - Max 1000 fichiers par application, fichiers < 1MB
  - Max 10 secondes par requête (la plupart se font en 300ms)
  - Requêtes et réponses < 1MB
  - Système de quotas.

Pourquoi pas LAMP ?

- LAMP est le standard
- Mais la gestion est lourde:
  - Configuration, tuning,
  - Backup, gestion de l'espace disque,
  - Pannes hardware, crashes système,
  - Mises à jour logicielles, patch de sécurité,
  - Rotation des logs, cron jobs, …
  - Reconception nécessaire quand la base de données a trop grossi.

Le modèle de développement

Console d'administration



Le contexte Java

- Environnement Servlet « sandboxé » (Java 7 JVM).
- SDK : appengine-api-*.jar
- URL : http://[*.]*your_app_id*.appspot.com
- Utilise Java Servlet API.
- Les requêtes sont routées à chaque fois
- Headers vérifiés puis transmis
- Les headers X-AppEngine-Country, Region, City et CityLatLong sont fournis
- Pour les admins connectés :
  - X-AppEngine-Estimated-CPM-US-Dollars : cout de 1000x la requête
  - X-AppEngine-Resource-Usage : resources utilisées

Le contexte Java

- Limite de 60 secondes par requête
  - DeadlineExceededException
- Interdits :
  - Système de fichiers, sockets, + autres…
- Avec limitation
  - Threads (utiliser la fabrique AppEngine, et ne pas dépasser la vie de la requête)
- Autorisés :
  - Réflexion, Class loader perso, java.util.logging
- Environnement :
  - com.google.appengine.runtime.environment, version, id, …
- Utilisation de java.util.logging.Logger pour logguer

Préparer son environnement

- Installer un JDK 7
- Si vous utilisez Maven, App Engine fournit un artifact plugin de build
- Si vous travaillez avec Eclipse, vous pouvez installer le plugin Google pour Eclipse
  - Il faudra alors aussi installer le SDK Google App Engine

Création d'une application

- https://appengine.google.com
- On crée un AppId (ex: lteconsulting)
- L'application sera disponibe à http://APP-ID.appspot.com
- Création du projet maven

- Le fichier appengine-web.xml
- Mode developpement
- Déploiement (un clic !)

- On peut utiliser l'archetype maven de google :
  - `mvn archetype:generate -Dappengine-version=1.9.32 -Dapplication-id=your-app-id -Dfilter=com.google.appengine.archetypes:`

Pom.xml

```
<dependency>
    <groupId>com.google.appengine</groupId>
    <artifactId>appengine-api-1.0-sdk</artifactId>
    <version>1.9.32</version>
</dependency>

<dependency>
  <groupId>com.googlecode.objectify</groupId>
  <artifactId>objectify</artifactId>
  <version>5.1.5</version>
</dependency>

<dependency>
    <groupId>com.google.appengine</groupId>
    <artifactId>appengine-testing</artifactId>
    <version>1.9.32</version>
    <scope>test</scope>
</dependency>

<plugin>
    <groupId>com.google.appengine</groupId>
    <artifactId>appengine-maven-plugin</artifactId>
    <version>1.9.32</version>
</plugin>
```

Configuration de l'application

- Web.xml : classique !
- Dans appengine-web.xml
  - ```
    <application>APP-ID</application>
    <version>1</version>
    <threadsafe>true</threadsafe> : pour traiter les requêtes en //
    <static-files>
        <include path="/**.png" expiration="4d 5h" />
    </static-files>
    <sessions-enabled>true</sessions-enabled> (Datastore + Memcache)
    <admin-console>
        <page name="Blog Comment Admin" url="/blog/admin/comments" />
        <page name="Create a Blog Post" url="/blog/admin/newentry" />
    </admin-console>
    ```
    ```

Autres fichiers de configuration (facultatifs)

- Datastore-indexes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<datastore-indexes
  autoGenerate="true">
    <datastore-index kind="Employee" ancestor="false">
        <property name="lastName" direction="asc" />
        <property name="hireDate" direction="desc" />
    </datastore-index>

    <datastore-index kind="Project" ancestor="false">
        <property name="dueDate" direction="asc" />
    </datastore-index>
</datastore-indexes>
```

- cron.xml :

```
<cronentries>
  <cron>
    <url>/recache</url>
    <description>Repopulate the cache every 2 minutes</description>
    <schedule>every 2 minutes</schedule>
  </cron>
</crontentries>
```

- Queue.xml : voir les TaskQueue
- Dos.xml : liste noire d'IPs

Cycle de développement

Construction du projet :

```
mvn install
```

Serveur de développement :

```
mvn appengine:devserver
```

Serveur de développement (rafraichissement automatique après un ‘mvn install’ :

```
mvn appengine:devserver_start
mvn appengine:devserver_stop
```

Publier :

```
mvn appengine:update
```

Users API

Users API

- GAE authentifie trois types d'utilisateurs:
  - Utilisateurs Google
  - Utilisateurs Google de votre domaine Google App
  - OpenID
- Détection de l'utilisateur loggué. Redirection à une page de login/logoff.
- On peut récupérer l'adresse email de l'utilisateur.
- On peut également détecter si l'utilisateur est administrateur de l'application.

Users API Exemple

- Exemple :

```
UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
If( user == null )
String urlToLogin = userService.createLoginURL( currentUrl );
```

- Avec GWT et l'API Servlet standard :

```
Principal user = getThreadLocalRequest().getUserPrincipal();
```

Single Sign On - OpenID

- Il est possible de faire du SSO avec App Engine

OAuth

- Permet à l'utilisateur d'autoriser l'appli à accéder à une autre appli web de sa part, de façon sécurisée.
- Exemple:
  - https://mail.google.com/mail/feed/atom/ : emails du compte GMail

Le Data Store

DataStore

- Base de données sans schéma supportant les transactions et un moteur de requêtes.
- Deux modèles de cohérence.

Datastore (HRD)

- Le Datastore (HRD) est une base de données répliquée, hautement disponible et distribuée.
- Essaie de proposer un compromis entre la scalabilité des bases NoSQL avec la facilité d'utilisation des systèmes RDBMS traditionnels.
- Trois points clé :
  - Largement utilisé à Google en interne,
  - Utilise l'algorithme Paxos pour la réplication entre data centers,
  - Cohérence à terme. 3 niveaux de cohérence à la lecture : current, snapshot and inconsistent.
- Transactions supportées, mais avec quelques contraintes

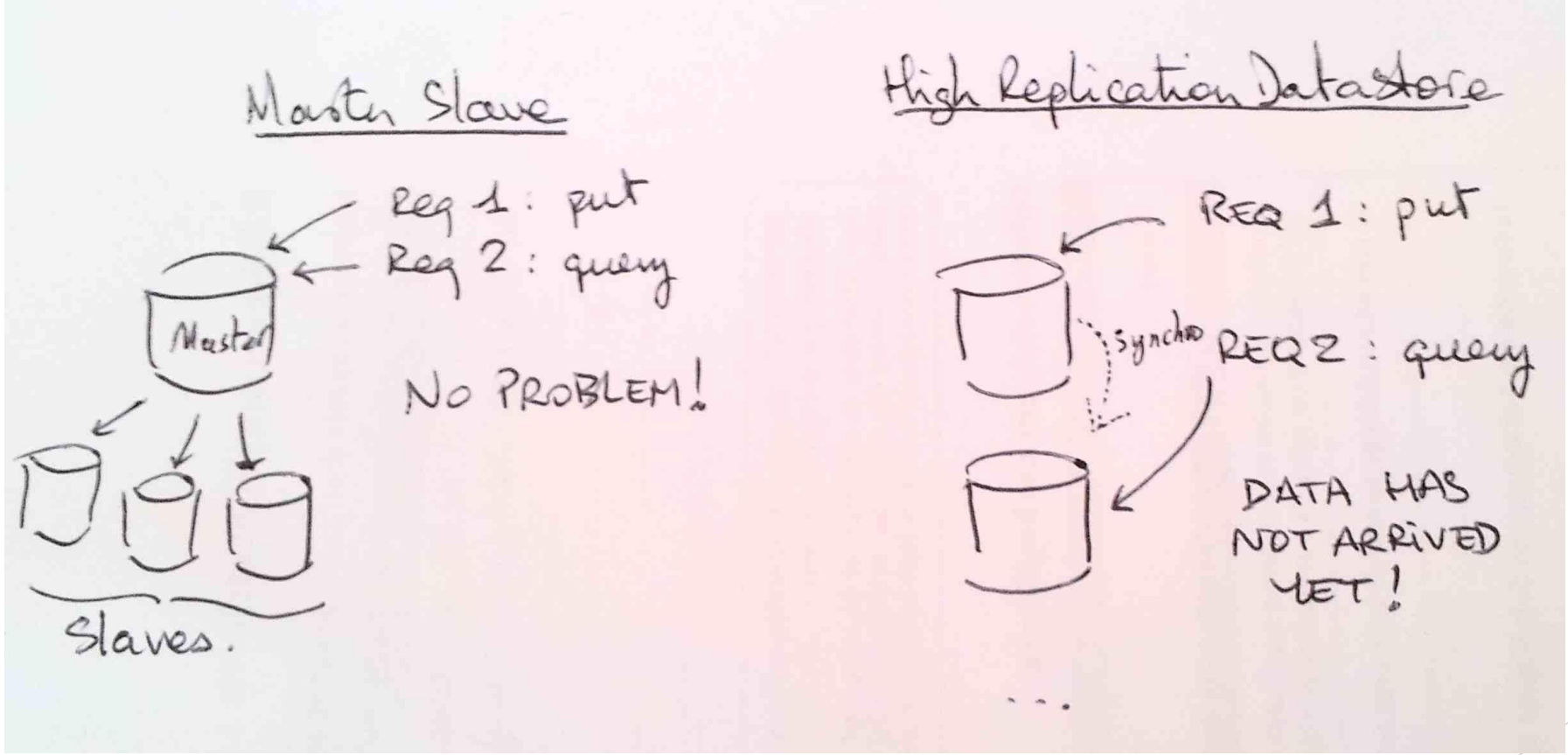
Datastore (Master/Slave)

- Déprécié !
- Un maitre, plusieurs esclaves.
- Les écritures sont propagées de façon asynchrone
- Cohérence forte
- Périodes en lecture seule planifiées occasionnellement
- Possible arrêt de service (restent rares)

Comparaison HDR / Master/Slave

	High Replication	Master/Slave
Performances		
Latence put/delete	1/2x – 1x	1x
Latence get	1x	1x
Latence requêtes	1x	1x
Cohérence		
Put/get/delete	forte	Forte
Requêtes ancêtre	forte	Forte
La plupart des requêtes	À la fin	Forte
Périodes lecture seule	NON	OUI
Temps d'arrêt non-planifiés	Extrêmement rare (2011=0%), pas de perte de données	Rare. Possibilité de perdre des écriture autour des périodes d'arrêt.

Problèmes avec la cohérence à terme (eventual consistency)



Solution : les requêtes ancêtre

Constraintes subies pour le HDR

- Le théorème CAP a prouvé une relation forte entre :
  - Cohérence
  - Disponibilité
  - Tolérance de panne (= réseau ou machine)
- Mais permet l'implémentation de la **cohérence à terme**.

APIs

- Bas niveau : DataStore API
  - Get, put, delete,
  - Préparation et exécution des requêtes,
  - Curseurs : pour servir une requête en plusieurs fois,
  - Paramétrage.
  - Package com.google.appengine.api.datastore
- Sous-ensemble de JDO et JPA (pas de jointures, …)
  - Supporté mais pas en adéquation avec le modèle initial
  - Fourni par l'implémentation open source DataNucleus
- Encore mieux : Objectify (Open Source), TWiG ou Slim3

Base de données hiérarchique

- Les Entities ont un Kind, une Key et des Properties
  - Entity => Enregistrement
  - Key => clé étrangère structurée
  - Kind => Table
  - Property => Colonne
- Dynamiquement typé: les types des Properties sont stockés par Entity.
- Key a soit un id, soit un nom et permet de trouver le chemin vers l'Entity.
- Entity groups sont l'unité transactionnelle (la racine et tous ses descendants)

Entities

- Possède une ou plusieurs Properties (nommées)
- Une clé.
  - Simple : Kind + EntityID
  - Evolué : Chemin depuis l'Entity racine, plus un ID ou un Nom (donné par l'appli).
  - Ne peut pas changer.
  - La relation parent-enfant définit les Entity Group.
- Sans schéma :
  - Les Entities d'une même Kind peuvent avoir de Properties différentes
  - Les Properties sont typées au niveau de chaque Entity.

Properties

- Types standards
  - Booléen, Int, Float (64 bits IEEE 754), String (<500 car.), Date, …
- Autres types standards
  - ShortBlob, Blob, adresse email, coordonnées géographiques
- Types particuliers
  - User, BlobKey, Link, IMHandle, PostalAddress, PhoneNumber, Rating, Text (<1Mo, non-indexé)

ReferenceKey property

- Référence à une autre Entity du modèle
- Peut représenter une relation PLUSIEURS-A-UN
- Utiliser la class Key

ListProperty

- Qu'est-ce ?
  - Une propriété qui a plusieurs valeurs
  - Une liste ordonnée de valeurs
  - Avec une requête à prédicat =
    - N'importe quelle valeur va matcher
    - Tri inutile sans index composite
- Pourquoi ?
  - Stocke l'information de façon dense
  - Comparable à une relation UN A PLUSIEURS
  - Très bon pour les requêtes 'set-membership', permet les requêtes merge-join
  - Evite un surplus de stockage (une seule entrée en index par valeur, pas de clé pour les Entity dans une relation UN A PLUSIEURS, pas d'entrée dans l'index PAR KIND)

ListProperty

- Attention quand même !
  - Utilise plus de CPU pour (dé)sérialiser les Entity
  - On modifie toute la liste pour un changement
  - La liste est sérialisée en bloc, >100 entrées donne une lecture lente
  - Fonctionne avec les tris sur une seule ListProperty, explosion d'index dans le cas contraire.
- Performances
  - Les mises à jour d'index sont effectuées en //. Donc mise à jour de 100 Entities avec 1000 éléments par liste simultanément
  - Limité à 5000 propriétés indexées par Entity

ListProperty

Favoris
couleur : StringlistPpty
username : StringPpty

```
SELECT * FROM Favoris
  WHERE username = "Sami"

SELECT * FROM Favoris
  WHERE couleur = "Jaune"
```

CouleurFavorite
couleur : StringPpty
username : StringPpty

```
SELECT * FROM CouleurFavorite
  WHERE username = "Sami"

SELECT * FROM CouleurFavorite
  WHERE couleur = "Jaune"
```

Illustration de l'API

- Récupérer une Entity
- Lecture asynchrone
  - DatastoreServiceFactory.getAsyncDatastoreService()
- Ne récupérer que les clés
  - Query.setKeysOnly()
- Requêtes GQL
- Callbacks

API

- Création d'une Entity :
  - ```
    DatastoreService datastore =
    DatastoreServiceFactory.getDatastoreService();

    Entity employee = new Entity("Employee");
    employee.setProperty("firstName", "Antonio");
    employee.setProperty("lastName", "Salieri");
    Date hireDate = new Date();
    employee.setProperty("hireDate", hireDate);
    employee.setProperty("attendedHrTraining", true);

    datastore.put(employee);
    ```
- Création d'un groupe d'Entity
  - ```
    Entity employee = new Entity("Employee");
    datastore.put(employee);
    Entity address = new Entity("Address", employee.getKey());
    ```
- Détruire une Entity
  - ```
    // Key employeeKey = ...;
    datastore.delete(employeeKey);
    ```
    ```

API

- Création des clés
  - ```
    Key k = KeyFactory.createKey(Employee.class.getSimpleName(),
    "Alfred.Smith@example.com");
    Key k = new KeyFactory.Builder(Employee.class.getSimpleName(),
    52234)
        .addChild(ExpenseReport.class.getSimpleName(),
    "A23Z79").getKey();
    ```
- Sérialisation des clés
  - ```
    String employeeKeyStr = KeyFactory.keyToString(employeeKey);
    // ...

    Key employeeKey = KeyFactory.stringToKey(employeeKeyStr);
    Entity employee = datastore.get(employeeKey);
    ```
- Opérations simultanées
  - .put, .delete et .get acceptent des Iterable pour effectuer les opérations en //

JDO / JPA

- Avec certaines limitations (on s'en doute)

Objectify

- Fait pour AppEngine
- Très simple
- Mime l'API Python

Objectify : entités

```
@Entity
public class ChatRoom
{
    @Id
    public Long id;

    @Index
    public String name;

    public ChatRoom()
    {
    }

    public ChatRoom( String name )
    {
        this.name = name;
    }
}
```

Objectify : déclaration des classes

```
ObjectifyService.register( UserRegistration.class );
ObjectifyService.register( ChatRoom.class );
…
```

Objectify : enregistrement d'une entité

```
ChatRoom monEntite = new ChatRoom(…);

ObjectifyService.ofy().save().entity( monEntite ).now();
```

Objectify : charger des entités

```
List<ChatRoom> chatRooms = ObjectifyService.ofy()
      .load()
      .type( ChatRoom.class )
      .order( "name" )
      .list();
```

Cloud SQL

Cloud SQL

- Dû à de nombreuses demandes
- Instance MySQL accessible par ligne de commande et API
- Max 10GB. Réplication synchrone.
- Pas de fonctions utilisateurs (ni trigger)
- Possibilité d'importer/exporter les données
  - Granularité par base de données seulement

Cloud SQL

- S’enregistrer et créer une instance (pour AppEngine)
- Créer sa BDD avec la ligne de commande
- Enregistrer le driver JDBC
  - ```
    import com.google.appengine.api.rdbms.AppEngineDriver;
    DriverManager.registerDriver(new AppEngineDriver());
    Connection c =
    DriverManager.getConnection("jdbc:google:rdbms://instance_name
    /guestbook");
    PreparedStatement ps = c.prepareStatement( « INSERT/UPDATE … »
    );
    ps.setString( 1, param );
    ps.executeUpdate();
    ResultSet rs = c.createStatement().executeQuery("SELECT
    guestName, content, entryID FROM entries");
    ```
    ```

Mem Cache

Le cache de données rapide et distribué

MemCache

- Une table de hash clé-valeur distribuée, en mémoire.
- Peux perdre des données à tout moment.
- On utilise soit l'API de Google, soit JCache API
  - Get, put,…
  - Get et put asynchrones (Future<?>)
  - Increment/decrement atomiques
- On peut spécifier les durées de stockage.
- Il faut prévoir que les données peuvent disparaître n'importe quand (même avant expiration!)

MemCache - API

```
String key = ..
byte[] value;

// Using the synchronous cache
MemcacheService syncCache = MemcacheServiceFactory.getMemcacheService();
value = (byte[]) syncCache.get(key); // read from cache
if (value == null) {
  // get value from other source (…)

  // populate cache
  syncCache.put(key, value);

}

// Using the asynchronous cache
AsyncMemcacheService asyncCache = MemcacheServiceFactory.getAsyncMemcacheService();
Future<Object> futureValue = asyncCache.get(key); // read from cache
// ... do other work in parallel to cache retrieval
value = (byte[]) futureValue.get();
if (value == null) {
  // get value from other source
  // ........

  // asynchronously populate the cache
  // Returns a Future<Void> which can be used to block until completion
  asyncCache.put(key, value);
}
```

Blobstore API

Blobstore API

- Stocke des données binaires, fichiers, etc…
- Maximum 2GB par object dans le BlobStore
- Ajout d'objet par HTML FORM ou avec l'API FileAPI
- Sert les Blobs au client en plaçant un header particulier dans la réponse, GAE remplace par le Blob.
- Les Blob sont read-only.
- L'appli peut lire les Blob par chunk de 32Mb
- Peut être utilisé avec ImageService

Blobstore API
Création d'un Blob par HTTP

- Création par HTTP Post:

```
<body>
    <form action="<%= blobstoreService.createUploadUrl("/upload") %>"
method="post" enctype="multipart/form-data">
        <input type="file" name="myFile">
        <input type="submit" value="Submit">
    </form>
</body>
```

- Dans la servlet:

```
Map<String, BlobKey> blobs = blobstoreService.getUploadedBlobs(req);
// blobs est indexée par les noms des fichiers uploadés
BlobKey blobKey = blobs.get("myFile");

// puis stockage de la clé BlobKey dans une Entity du Datastore
// …
```

Blobstore API
Création avec l'API File

```
String content = … // contenu du Blob
BlobKey blobKey = null; // clé du Blob créé
try
{
    FileService fileService = FileServiceFactory.getFileService();
    AppEngineFile file = fileService.createNewBlobFile( "text/plain" );

    // On locke le fichier, car on va le finaliser
    FileWriteChannel writeChannel = fileService.openWriteChannel( file, true );

    PrintWriter out = new PrintWriter( Channels.newWriter( writeChannel, "UTF8" ) );
    out.print( content );
    out.close();

    writeChannel.closeFinally();

    blobKey = fileService.getBlobKey( file );
}
catch( Exception e ) { e.printStackTrace(); }

if( blobKey == null )
    return;

Entity someEntity = …; // L'Entity dans laquelle nous allons stocker la clé du Blob

someEntity.setProperty( "myBlobKey", blobKey );
dataStoreService.putEntity( someEntity );
```

Blobstore API
Servir un Blob

- On peut envoyer au client la clé du Blob :
  - blobKey.getKeyString()
- Ensuite dans la servlet appropriée :

```
BlobKey blobKey = new BlobKey( req.getParameter( "key" ) );
bs.serve( blobKey, resp );
```

Image API

Image API

- Manipulation basiques d’images.

Images API

- Opérations basiques :
  - Taille, rotation, flip, crop, lucky
- Formats pris en charge :
  - En entrée : jpeg, png, webp, gif, bmp, tiff, ico
  - En sortie : jpeg, webp, png
- Utilisable à partir du Blobstore
- getServingUrl() pour servir des images transformées

URL Fetch

API URLFetch

- Accès aux ressource Web (REST apis, RSS, etc…)
  - Seulement HTTP et HTTPS (pas d'auth. de l'hôte avec certificats auto-signés)
  - Deadline de 10 secondes
  - API synchrone et asynchrone
- Deux API : l'API bas niveau de Google…
- … et prise en charge de java.net.URLConnection
- On utilise Google Secure Data Connector pour joindre son réseau privé.
- HTTP Response URLFetchService.fetch(« http://… »);

File API

File API

- Utilisable avec Blobstore et Google Cloud Storage

```
FileService fileService = FileServiceFactory.getFileService();
GSFileOptionsBuilder optionsBuilder = new GSFileOptionsBuilder()
      .setBucket("mybucket").setKey("myfile").setMimeType("text/html").setAcl("public_read")
      .addUserMetadata("myfield1", "my field value");
AppEngineFile writableFile = fileService.createNewGSFile(optionsBuilder.build());

// Open a channel to write to it, withour locking
FileWriteChannel writeChannel = fileService.openWriteChannel(writableFile, false);

// Different standard Java ways of writing to the channel are possible. Here we use a PrintWriter:
PrintWriter out = new PrintWriter(Channels.newWriter(writeChannel, "UTF8"));
out.println("The woods are lovely dark and deep.");
out.println("But I have promises to keep.");

// Close without finalizing and save the file path for writing later
out.close();


// Later on, we want to finish writing
String path = writableFile.getFullPath();
// Write more to the file in a separate request:
writableFile = new AppEngineFile(path);
// This time lock because we intend to finalize
writeChannel = fileService.openWriteChannel(writableFile, true);
// This time we write to the channel using standard Java
writeChannel.write(ByteBuffer.wrap("And miles to go before I sleep.".getBytes()));
// Now finalize
writeChannel.closeFinally();
```

File API

```
// …
// At this point the file is visible in App Engine as: "/gs/mybucket/myfile"
// and to anybody on the Internet through Google Storage as:
// (http://commondatastorage.googleapis.com/mybucket/myfile)
```

- Pour lire un fichier Cloud Storage

```
// So reading it through Files API:
String filename = "/gs/mybucket/myfile";
AppEngineFile readableFile = new AppEngineFile(filename);
FileReadChannel readChannel = fileService.openReadChannel(readableFile, false);
// Again, different standard Java ways of reading from the channel.
BufferedReader reader = new BufferedReader(Channels.newReader(readChannel,
"UTF8"));
String line = reader.readLine();
// line = "The woods are lovely dark and deep."
readChannel.close();
```

Conversion API

- Conversion facile entre formats de documents.
- Permet même l'OCR !!! (image vers pdf par exemple)

```
// Create a conversion request from HTML to PNG.
Asset asset = new Asset( mimeType, contentAsBytes, "Name" );
Document document = new Document( asset );
Conversion conversion = new Conversion( document, "application/pdf" );

ConversionService service = ConversionServiceFactory.getConversionService();
ConversionResult result = service.convert( conversion );

if( result.success() )
{
    // Note: in most cases, we will return data all in one asset,
    // except that we return multiple assets for multi-page images.
    for( Asset resultAsset : result.getOutputDoc().getAssets() )
    {
        resp.addHeader( "Content-type", "application/pdf" );
        resp.getOutputStream().write( resultAsset.getData() );
    }
}
```

Mail API

Mail API
Envoi d'email

- Avec l'API JavaMail :

```
Properties props = new Properties();
Session session = Session.getDefaultInstance(props, null);
String msgBody = "...";

// dans un bloc try/catch (AddressException, MessagingException, …)
// l'expéditeur doit être un admin de l'application ou l'utilisateur connecté
Message msg = new MimeMessage(session);
msg.setFrom(new InternetAddress("admin@example.com", "Example.com Admin"));
msg.addRecipient(Message.RecipientType.TO,
                 new InternetAddress("user@example.com", "Mr. User"));
msg.setSubject("Your Example.com account has been activated");
msg.setText(msgBody);
Transport.send(msg);
```

Mail API
Réception d'emails

- Votre application reçoit les emails à l'adresse :
  - anystring@appid.appspotmail.com
- Les emails sont reçus sous forme de requête HTTP à l'adresse /_ah/mail/address (handler /_ah/mail/* par ex.)
- Désactivé par défaut. Appengine-web.xml :
  - <inbound-services><service>mail</service></inbound-services>
- Console locale : http://localhost:8888/_ah/admin/

```
Properties props = new Properties();

Session session = Session.getDefaultInstance(props, null);

// dans un bloc try/catch (MessagingException)
MimeMessage message = new MimeMessage( session,
            req.getInputStream() );

String from = message.getFrom()[0].toString();
```

Channel API

ChannelAPI

- Envoi de données sans polling vers le client (eg Push, Comet, …)
- Requiert JavaScript sur le client (bien sûr)
- Construit sur le système Google Talk
- Exemple:

```
String channelId = …; // cette chaine identifie de manière unique
la connection au client

ChannelService svc = ChannelServiceFactory.getChannelService();
// Le token doit être passé au client.
String token = svc.createChannel( channelId );

// Pour envoyer un message au client :
String message = …; // la chaine String qu'on veut envoyer
svc.sendMessage( new ChannelMessage( channelId, message ) );
```

Channel API

- Pour être notifié des connections et déconnections :
  - Dans appengine-web.xml :
  - `<inbound-services><service>channel_presence</service></inbound-services>`
- L'application reçoit des POST aux urls :
  - /_ah/channel/connected
  - /_ah/channel/disconnected
  - ChannelService.parsePresence(request) pour obtenir le clientId
- Un seul client par clientId
- Un seul channel par page

Channel API
Côté client

- En javascript :

```
<script type="text/javascript" src="/_ah/channel/jsapi"></script>

<script>
    channel = new goog.appengine.Channel( token );
    socket = channel.open();
    socket.onopen = onOpened;
    socket.onmessage = onMessage;
    socket.onerror = onError;
    socket.onclose = onClose;

    onMessage = function( message ) {
        alert( “on a reçu : ” + message.data );
    };
</script>
```

Data Store
Utilisation avancée

Datastore Callbacks

- Permet d'exécuter du code à différents points du processus de persistence
- Before/After get/put
- Ajouter <SDK_ROOT>/lib/impl/appengine-api.jar comme processeur d'annotations
- Exemple :

```
class PrePutCallbacks {
    static Logger logger = Logger.getLogger("callbacks");

    @PrePut(kinds = {"Customer", "Order"})
    void log(PutContext context) {
        logger.fine("Putting " + context.getCurrentElement().getKey());
    }

    @PrePut // Applies to all kinds
    void updateTimestamp(PutContext context) {
        context.getCurrentElement().setProperty("last_updated", new Date());
    }
}
```

Transactions

- Modèle de concurrence optimiste
  - Exécution en // mais une seule réussira
- Se font avec la granularité d'un Entity Group
- Supporte les tx Cross-EntityGroup avec le commit à deux phases (moins performant)
  - Maximum de 5 Entities pour ces transactions
  - Les requêtes non-transactionnelles pourront voir toutes, une partie ou aucune des modifications d'une transaction validée (committed).
  - Plus lent car algo adapté (two phase commit => plus lent)
- => impact sur le design du modèle de données

Transactions - Modèle

- Toutes les écritures sont transactionnelles
- Journaux estampillés
- Historique des version des Entity
- Modèle de concurrence optimiste

Lire l'horodatage du dernier commit	Key: A — Entity: X (3:00) — Journal: ... — **Committed: 3:00**
Ecrire le journal	Key: A — Entity: X (3:00) — **Journal: 3:45 A←Y** — Committed: 3:00
Appliquer le journal	Key: A — **Entity: Y (3:45) X (3:00)** — Journal: 3:45 A←Y — Committed: 3:00
MAJ horodatage commit	Key: A — Entity: Y (3:45) X (3:00) — Journal: 3:45 A←Y — **Committed: ~~3:00~~ 3:45**

Transactions - Exemple

```
def pay(parent_key, child_key, amount):
  parent, child = db.get(parent_key, child_key)
  parent.cash -= amount
  child.cash += amount
  db.put(parent, child)

db.run_in_transaction(
  pay, parent.key(), child.key(), 10)
```

- Lire l'Entity Group de **`parent`**
- Garder en mémoire le timestamp, 3:00pm
- Lire **`parent`** et **`child`** entities à l'état 3:00pm
- Sérialiser **`parent`** et **`child`**
- Ecrire dans le journal dans la ligne de l'EntityGroup de **`parent`**
- Appliquer les changements journalisés à **`parent`** et **`child`**
- Ecrire le nouveau timestamp à l'Entity Group
- Mise à jour des indexes pour **`parent.cash`** et **`child.cash`**

Indexes

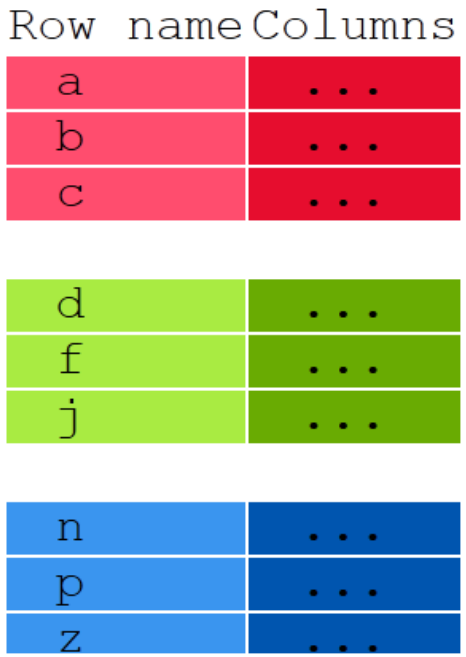
- Les Properties sont automatiquement indéxées par kind+valeur
  - Un index pour chaque combinaison Kind + Property name
  - A chaque mise-à-jour d'une Entity, tous les index sont mis-à-jour
  - Certains types ne sont pas indexés (Blob et Text)
- Contiennent les résultats d'une requête dans l'ordre désiré.
- Supportent les requêtes simples (and et =) et la recherche dichotomique rapide

Indexes

- Création d'indexes composites pour servir les requêtes plus complexes
  - Les indexes sont proposés automatiquement par le SDK
  - Supportent les inégalités et le tri des résultats
  - Attention à l'explosion des indexes sur les champs de type List !!!
- Les lignes sont triées selon :
  - Ancêtres, filtres d'égalité, filtres d'inégalité, ordre du résultat (tri)
- On peut spécifier à GAE de ne pas indéxer certaines propriétés :
  - Entity.setUnindexedProperties(…);

Indexes en détail

- Tout repose sur BigTable
  - Un tableau trié, réparti
  - Read, write, delete, single-row transaction, scan prefix ou range



Indexes en détail

- L'Entities Table.
  - Table primaire, stocke toutes les Entities de toutes les applis
  - Le nom de ligne est la clé de l'Entity
  - Une seule colonne : l'Entity sérialisée avec protobuf

```
/Grandparent:Alice
/Grandparent:Alice/Parent:Sam
/Grandparent:Ethel
/Grandparent:Ethel/Parent:Jane
/Grandparent:Ethel/Parent:Jane/Child:Timmy
/Grandparent:Ethel/Parent:Jane/Child:William
/Grandparent:Frank
```

Indexes en détail

- Les indexes mappent des valeurs vers les Entities
- Pas de filtrage ni de tri en mémoire
- Query planner : convertit les requêtes en scan dense d'index
  - Trouver l'index
  - Choisir prefix ou range
  - Scanner

Indexes en détail

- L'index Kind
  - Sert les requêtes de toutes les Entities d'une certaine Kind
  - SELECT * FROM GrandParent
  - Scanner avec le préfixe GrandParent :

Kind	Key
Child	Jimmy
Child	Timmy
Grandparent	Ethel
Grandparent	Fred
Parent	Jane
Parent	John
Parent	Todd

Index en détails

- Index pour une seule propriété (Single property index)
  - Sert les requêtes portant sur une seule Property
  - Un descendant, un ascendant

Kind	Name	Value
Parent	address	1 Palm Dr.
Parent	name	Alice
Parent	name	Bob
Parent	name	Brad
Parent	name	Chelsea
Parent	name	Jane
Parent	name	John
Parent	title	Ninja Pirate

Index en détails

- WHERE name = ‘John’
- Scanner avec préfixe Parent name John

Kind	Name	Value
Parent	address	1 Palm Dr.
Parent	name	Alice
Parent	name	Bob
Parent	name	Brad
Parent	name	Chelsea
Parent	name	Jane
Parent	name	John
Parent	title	Ninja Pirate

Index en détails

- ORDER BY name DESC
- Scanner (DESC) avec le préfixe Parent name

Kind	Name	Value DESC
Parent	address	1 Palm Dr.
Parent	name	John
Parent	name	Jane
Parent	name	Chelsea
Parent	name	Brad
Parent	name	Bob
Parent	name	Alice
Parent	title	Ninja Pirate

Index en détails

- WHERE name>='b' AND name<'c' ORDER BY name
- Scanner l'intervalle [Parent name b, Parent name c)

Kind	Name	Value
Parent	address	1 Palm Dr.
Parent	name	Alice
Parent	name	Bob
Parent	name	Brad
Parent	name	Chelsea
Parent	name	Jane
Parent	name	John
Parent	title	Ninja Pirate

Index en détails

- Index composés
  - Servent les requêtes sur plusieurs Properties ou Ancestor
  - WHERE firstname='Ryan' AND lastname='Barrett'
  - WHERE firstname>='b' AND firstname<'c' AND lastname='Smith'
  - WHERE ANCESTOR IS :ethel ORDER BY firstname
- Index(Parent, [ANCESTOR], lastname, firstname)

Kind	lastname	firstname
Parent	Anderson	Jane
Parent	Barrett	Ryan
Parent	Smith	Bob
Parent	Thomas	Alice

Index en détails

- Planificateur de requêtes :
  - Ajouter Kind au préfixe,
  - Ajouter l'Ancestor le cas échéant,
  - Ajouter les filtres d'égalité, le cas échéant,
  - Convertir en Range scan si il y a des filtres d'inégalité,
  - Ajouter l'ordre de tri, le cas échéant.

Kind	lastname	firstname
Parent	Anderson	Jane
Parent	Barrett	Ryan
Parent	Smith	Bob
Parent	Thomas	Alice

Index en détails

- Autre exemple :
  - WHERE ANCESTOR IS :Ethel ORDER BY firstname
  - Scanne avec préfixe Parent /GrandParent:Ethel

Kind	Ancestor	firstname
Parent	/Grandparent:David	Jane
Parent	/Grandparent:Ethel	Bob
Parent	/Grandparent:Ethel	Chelsea
Parent	/Grandparent:Ethel	Ryan
Parent	/Grandparent:Fred	Alice

Index en détails

- Merge-join
  - Servent de multiples = sans index composé
  - Peut inclure des Ancestor
  - Ex: WHERE firstname='John' AND lastname='Smith'
  - Scanne les index Single Property une fois pour chaque valeur filtrée
  - Fait l'intersection des résultats

- Scan with prefix **`Parent firstname John`**:

Kind	Name	Value
Parent	firstname	John
Parent	firstname	John
Parent	firstname	Ryan

- ...and with prefix **`Parent lastname Smith`**:

Parent	lastname	Barrett
Parent	lastname	Smith
Parent	lastname	Smith

- Intersect results incrementally, using key:

Kind	Name	Value	Key
Parent	firstname	John	X
Parent	firstname	John	Y
Parent	lastname	Smith	Y
Parent	lastname	Smith	Z

Index - Explosion

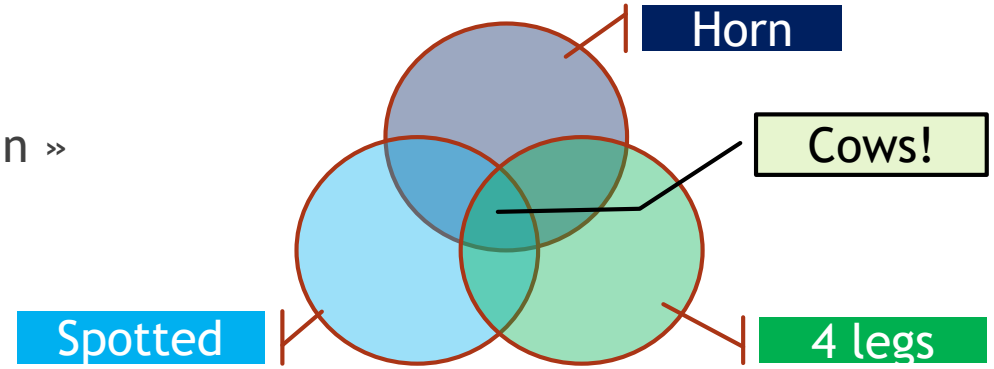
- Dans certains cas, les index peuvent exploser
  - Chaque propriété de chaque entité est ajoutée dans au moins 1 index (principal) + 1 fois par index qui référence cette propriété
  - Les propriétés à valeur multiples stockent chaque valeur distinctement
- Exemple :
  - Entity myModel = new Entity("Model");
    myModel.setProperty("x", Arrays.asList("one", "two"));
    myModel.setProperty("y", Arrays.asList("three", "four"));
    ds.put(myModel)
  - Cette entity provoque 12 écritures dans les index

Requêtes

- N’autorise que les requêtes efficaces (directement sur les indexes) :
  - Pas de jointure, pas d’agrégation,
  - Pas de tri en mémoire,
  - Pas de scans de table.
- Pas d’opérateur d’inégalité sur plus d’une Property,
- Pas de filtrage basé sur une requête précédente,
- Requêtes seulement sur les Properties indexées,
- Les propriétés dans les filtres d'inégalité doivent être triées avant que les autres ordres de tri ne soient appliqués

Requêtes

- Permet les requêtes « merge-join »



- Résultat contient soit toutes les Properties, soit seulement les Keys.
- Requêtes Ancêtres : les seules à garantir la cohérence forte.

Requêtes complexes

- Dev mode génère automatiquement des suggestions d'indexes.
- revenu>100000 and age<30
  - Non autorisé. Inégalité sur deux champs interdite.
- Sexe=f and age>18 and age<25
  - Autorisé, deux inégalités sur le même champ.
  - A besoin d'un index manuel (tri par sexe, puis par age)
- * sort by age desc
  - Nécessite aussi un index manuel (tri inversé)

Requêtes – pourquoi ces contraintes ?

- Pour **passer à l'échelle**
- Les performances sont **fonction de la taille des résultats et non de la taille de la base de données.**
  - Très important : Une requête dont le résultat contient 100 Entities se comportera de la même façon indépendamment si la base contient 100 Entities ou 100 millions.
  - Cela fait tout-à-fait sens pour des applications comme Gmail ou Google!
- Pas d'inégalité sur deux Properties ?
  - Le mécanisme des requêtes repose sur le fait que leurs résultats sont adjacents dans les indexes (éviter les scans de table). Un index ne peut pas représenter des filtres d'inégalité sur plusieurs Properties et maintenir cette propriété).

Requêtes - API

- Effectuer une requête
  - ```
    DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();

    // The Query interface assembles a query
    Query q = new Query("Person")
    q.addFilter("lastName", Query.FilterOperator.EQUAL, lastNameParam);
    q.addFilter("height", Query.FilterOperator.LESS_THAN, maxHeightParam);

    // PreparedQuery contains the methods for fetching query results
    // from the datastore
    PreparedQuery pq = datastore.prepare(q);

    for (Entity result : pq.asIterable()) {
      String firstName = (String) result.getProperty("firstName");
      String lastName = (String) result.getProperty("lastName");
      Long height = (Long) result.getProperty("height");
      System.out.println(lastName + " " + firstName + ", " + height.toString() + "
    inches tall");
    }
    ```
    ```

Requêtes - API

- Requête ancêtre
  - ```
    DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
    Entity person = new Entity("Person", "tom");

    Entity weddingPhoto = new Entity("Photo", person.getKey());
    weddingPhoto.setProperty("imageUrl",
        "http://domain.com/some/path/to/wedding_photo.jpg");

    Entity babyPhoto = new Entity("Photo", person.getKey());
    babyPhoto.setProperty("imageUrl",
        "http://domain.com/some/path/to/baby_photo.jpg");

    Entity dancePhoto = new Entity("Photo", person.getKey());
    dancePhoto.setProperty("imageUrl",
        "http://domain.com/some/path/to/dance_photo.jpg");

    Entity campingPhoto = new Entity("Photo");
    dancePhoto.setProperty("imageUrl",
        "http://domain.com/some/path/to/camping_photo.jpg");

    datastore.put(Arrays.asList(person, weddingPhoto, babyPhoto, dancePhoto,
        campingPhoto));

    Query userPhotosQuery = new Query("Photo");
    userPhotosQuery.setAncestor(person.getKey());

    // This returns weddingPhoto, babyPhoto and dancePhoto, but
    // not campingPhoto because tom is not an ancestor.
    List<Entity> results = datastore.prepare(userPhotosQuery).asList(
        FetchOptions.Builder.withDefaults());
    ```
    ```

Ecritures

- Limitées à 5 par seconde par Entity Group => utiliser le sharding si nécessaire
- Ecriture en deux phases:
  - Commit : écriture dans le journal
  - Apply : deux opérations en parallèle. L'Entity est écrite puis, les indexes sont mis-à-jour.
- Visibilité des écritures
  - Get, write et ancestor queries voient toujours le résultat à jour
  - Requêtes sur plusieurs Entity Group peuvent retourner des résultats partiels
  - Les requêtes avec prédicat seront à jour seulement après la phase « Apply ».

Curseurs

- Sérialisation de la position de la ligne courante dans le résultat pour continuer plus tard.
- Limitations:
  - Non supporté sur des requêtes avec IN ou !=
  - Tri sur propriété avec plusieurs value : l'Entity peut apparaître plusieurs fois dans les résultats.

- Exemple :

```
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
Query q = new Query("Person");
PreparedQuery pq = datastore.prepare(q);
int pageSize = 15;

resp.setContentType("text/html");
resp.getWriter().println("<ul>");

FetchOptions fetchOptions = FetchOptions.Builder.withLimit(pageSize);
String startCursor = req.getParameter("cursor");

// If this servlet is passed a cursor parameter, let's use it
if (startCursor != null) {
    fetchOptions.startCursor(Cursor.fromWebSafeString(startCursor));
}

QueryResultList<Entity> results = pq.asQueryResultList(fetchOptions);
for (Entity entity : results) {
    resp.getWriter().println("<li>" + entity.getProperty("name") + "</li>");
}
resp.getWriter().println("</ul>");

String cursor = results.getCursor().toWebSafeString();

// Assuming this servlet lives at '/people'
resp.getWriter().println(
            "<a href='/people?cursor=" + cursor + "'>Next page</a>");
```

Modéliser ses données

- Collections:
  - Les collections ordonnées obligent GAE à créer un index représentant la position dans la collection. Si beaucoup de changement de position, cela peut être très inefficace. Mieux vaut stocker dans une autre structure la position et trier dans le client.
- Composition / Agrégation :
  - Si la relation fait qu'une Entity possédée est détruite quand le possesseur est détruit, c'est de la composition : embarquer les données dans l'Entity (@Embedded ou @Persistent(serialized=true) en JDO).
  - Sinon, utiliser ReferenceEntity

Modéliser ses données

- Dénormaliser ou pas ?
  - En général: OUI.
  - Si beaucoup de mises à jour : NON.
  - Il faut comprendre le cycle de vie d'un objet.
- Relation UN VERS UN:
  - Utiliser ReferenceEntity

Relation UN A PLUSIEURS ReferenceEntity

- Stocker la Key du ONE dans les Entities du MANY avec une Property de type ReferenceEntity
- Ex:
  - Propietaire : nom : String
  - Animal : nom : String, proprietaire : ReferenceProperty
  - Pour obtenir les animaux d'un propriétaire :
    - Pet.all().filter(« proprietaire = » + proprietaire).fetch(1000)
    - Ou proprietaire.animaux_set.fetch(1000);
- * voir ReferenceProperty à propos de la collection virtuelle

Relation UN A PLUSIEURS
Utilisant la relation de parenté

- Quand on crée l'Entity du côté PLUSIEURS, on lui affecte l'Entity UN comme parent.
- Attention ! cela crée un EntityGroup (5 writes/second max)
- Animal = new Entity(« Animal », proprietaire);
- Comment choisir ? Penser aux transactions nécessaires à l'application. Penser également qu'une entité ne peut avoir qu'un parent, et qu'il ne peut pas changer.

Relation PLUSIEURS A PLUSIEURS Utilisant une table de jointure

- Même logique qu'en RDBMS
- Une table de jointure contenant une ReferenceProperty pour chaque côté de la relation
- Désavantage : beaucoup de lecture depuis DataStore, coûteux.

Relation PLUSIEURS A PLUSIEURS
Avec un liste de clés

- Un côté de la relation stocke une liste de clés des Entities de l'autre côté.
  - OK quand la cardinalité d'un côté est limité (sinon explosion d'index).
  - Bon pour mettre à jour la liste des références en un appel.
- Exemple:
  - Animaux possédés par un propriétaire : db.get(bob.pets);
  - Propriétaires d'un animal : Owner.all().filter('pets=' + felix).fetch(100);
- Limitations:
  - Charger explicitement les Entities du côté de la liste (on n'a que les clés)
  - Mettre la liste du côté ou la cardinalité est faible.
- Technique hybride:
  - http://code.google.com/events/io/2009/sessions/BuildingScalableComplexApps.html

Exemple: recherche full text

- Pas disponible d'office,
- Créer une table KeywordIndex
  - L'Entity KeywordIndex est une paire [mot-clé, entity key]
  - Normaliser les mot-clé (pas d'accents, etc)
- Quand on cherche un préfixe:
  - ofy().query(Customer.class).filter("keyword >=", prefix).filter("keyword <=", prefix + "\ufffd").list();

Les queues

TaskQueues API

- Fonctionne en dehors des requêtes utilisateur
- Organise le travail en petites tâches parallèles
- Deadline de 10 minutes au lieu de 30 secondes pour les requêtes
- Queues nommables et paramétrables (#/secondes, …)
- Nommer pour prévenir la duplication
- Les Tâches doivent être idempotentes (pourquoi ?)
- Exemple d'utilisation :
  - Envoi d'emails, Fan out, fan in, Migration du modèle de données

PushQueues et PullQueues

- PushQueue:
  - Consommées par votre application
  - Le dispatching est géré par AppEngine
  - Limite de 10 minutes pour exécuter la tâche (DeadlineExceededException)
- PullQueue:
  - Pour être consommé par l'extérieur (Task REST API) ou sur un Backend AppEngine
  - Consomme les tâches quand vous êtes prêt
  - Le consommateur de tâche doit passer à l'échelle !

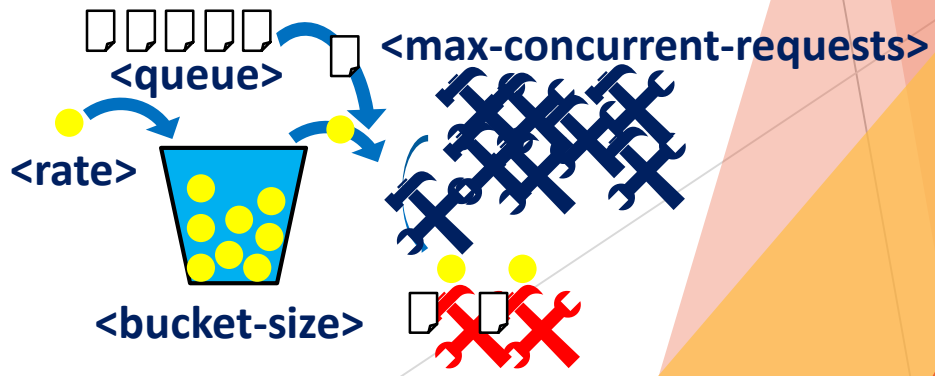
Configurer une Queue

Dans WEB-INF/queue.xml :

```
<queue-entries>
  <total-storage-limit>50M</total-storage-limit>
  <queue>
       <name>maFileDeTaches</name>
       <rate>1/s</rate>
       <bucket-size>20</bucket-size>
       <max-concurrent-requests>10</max-concurrent-requests>

       <retry-parameters>
            <task-retry-limit>7</task-retry-limit>
            <task-age-limit>2</task-age-limit>
            <min-backoff-seconds>10</min-backoff-seconds>
            <max-backoff-seconds>200</max-backoff-limit>
            <max-doublings>2</max-doublings>
       </retry-parameters>

       <mode>pull</mode>
       <acl>
            <user-email>bar@foo.com</user-email>
       </acl>
  </queue>
</queue-entries>
```



Mettre une tâche en file d'attente

- Simple :

```
import com.google.appengine.api.taskqueue.Queue;
import com.google.appengine.api.taskqueue.QueueFactory;
import static com.google.appengine.api.taskqueue.TaskOptions.Builder.*;
Queue queue = QueueFactory.getDefaultQueue();
queue.add( TaskOptions.Builder.withUrl("/worker").param("key", key));
```

  - Pour une pull queue : .withMethod(TaskOptions.Method.PULL)
  - Pour une tâche nommée : withTaskName(« name »)
- **Dans une transaction, max 5 tasks. Nommage obligé.**
- **Pour effacer une tâche :**
  - Admin console
  - purge() function. Attention à ne pas créer de tâche à ce moment

Push Queues : URLs

- Les Push Queues référencent leur implémentation par une URL. (par défault : /_ah/queue/queue_name)
- Pour sécuriser les urls, dans web.xml :
  - ```
    <security-constraint>
            <web-resource-collection>
                    <url-pattern>/tasks/*</url-pattern>
            </web-resource-collection>
            <auth-constraint>
                    <role-name>admin<role-name>
            </auth-constraint>
    </security-constraint>
    ```
- Pour faire exécuter la tâche par un Backend :
  - ```
    Queue.add(
      withUrl( url )
      .header( «Host», BackendServiceFactory.getBackendService().getBackendAddress(
    «backend1», 1 ) );
    ```

Pull Queues

- On ajoute une tache à la queue
- On réclame un bail sur *n* tâches
  - List<TaskHandle> tasks = queue.leaseTasks(nb, duration);
- On éxécute la tâche
- On l'efface
  - queue.deleteTask(« taskName »);
- Pour traiter les tâches en-dehore d'AppEngine :
  - API RESTful
  - https://www.googleapis.com/taskqueue/v1beta1/projects/*{PROJECT_NAME}*/taskqueues/*{TASKQUEUE_NAME}*

Les Backends

Backends

- Instances avec :
  - Pas de limite de temps, de mémoire ou de CPU
  - Etat persistant entre requêtes
- Paramétrage avec backends.xml
- Démarrées automatiquement par AppEngine (ou pas pour les Resident Backends)
- Load-balancing possible entre plusieurs instances
- Déclenchées par : continuellement, TaskQueues, Cron jobs, …
- Facturées par temps d'utilisation et non de CPU
  - Ex: $0.64/h pour 4.8Ghz avec 1024MB

Backends

- Peut être résident ou dynamique, privé ou public
- Ne scale pas automatiquement
- Au démarrage l'url /_ah/start est appelée
- Arrêt d'un Backend détectable par :

```
LifecycleManager.getInstance().setShutdownHook(new ShutdownHook() {
  public void shutdown() {
    LifecycleManager.getInstance().interruptAllRequests();
  }
});
```

```
while (haveMoreWork() && !LifecycleManager.getInstance().isShuttingDown())
  doSomeWork(); // et peut être sauvegarde de l'état…
```

- Adressable par http://[instance].nom.appId.appspot.com
- Obtenir le nom et l'instance du backend :
  - String backendApi.getCurrentBackend()
  - int backendApi.getCurrentInstance()

Map Reduce

MapReduce

- Le Map du MapReduce
- Lance une tâche sur toutes les entités d'une Kind donné dans le DataStore
- Configurer la MapReduceServlet:

```
<servlet>
  <servlet-name>mapreduce</servlet-name>
  <servlet-class>com.google.appengine.tools.mapreduce.MapReduceServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>mapreduce</servlet-name>
  <url-pattern>/mapreduce/*</url-pattern>
</servlet-mapping>
```

Configurer un Mapper

```
<configurations>
  <configuration name="Demo Mapper">
    <property>
      <name>mapreduce.map.class</name>
      <value>com.google.appengine.demos.mapreduce.TestMapper</value>
    </property>
    <property>
      <name>mapreduce.inputformat.class</name>
      <value>com.google.appengine.tools.mapreduce.DatastoreInputFormat</value>
    </property>
    <property>
      <name human="Entity Kind to Map Over">mapreduce.mapper.inputformat.datastoreinputformat.entitykind</name>
      <value template="optional">PBFVotes</value>
    </property>
  </configuration>
</configurations>
```

La classe Mapper

```
public class TestMapper extends AppEngineMapper<Key, Entity, NullWritable, NullWritable> {
  private static final Logger log = Logger.getLogger(TestMapper.class.getName());

  public TestMapper() {
  }

  @Override
  public void taskSetup(Context context) {
    log.warning("Doing per-task setup");
  }

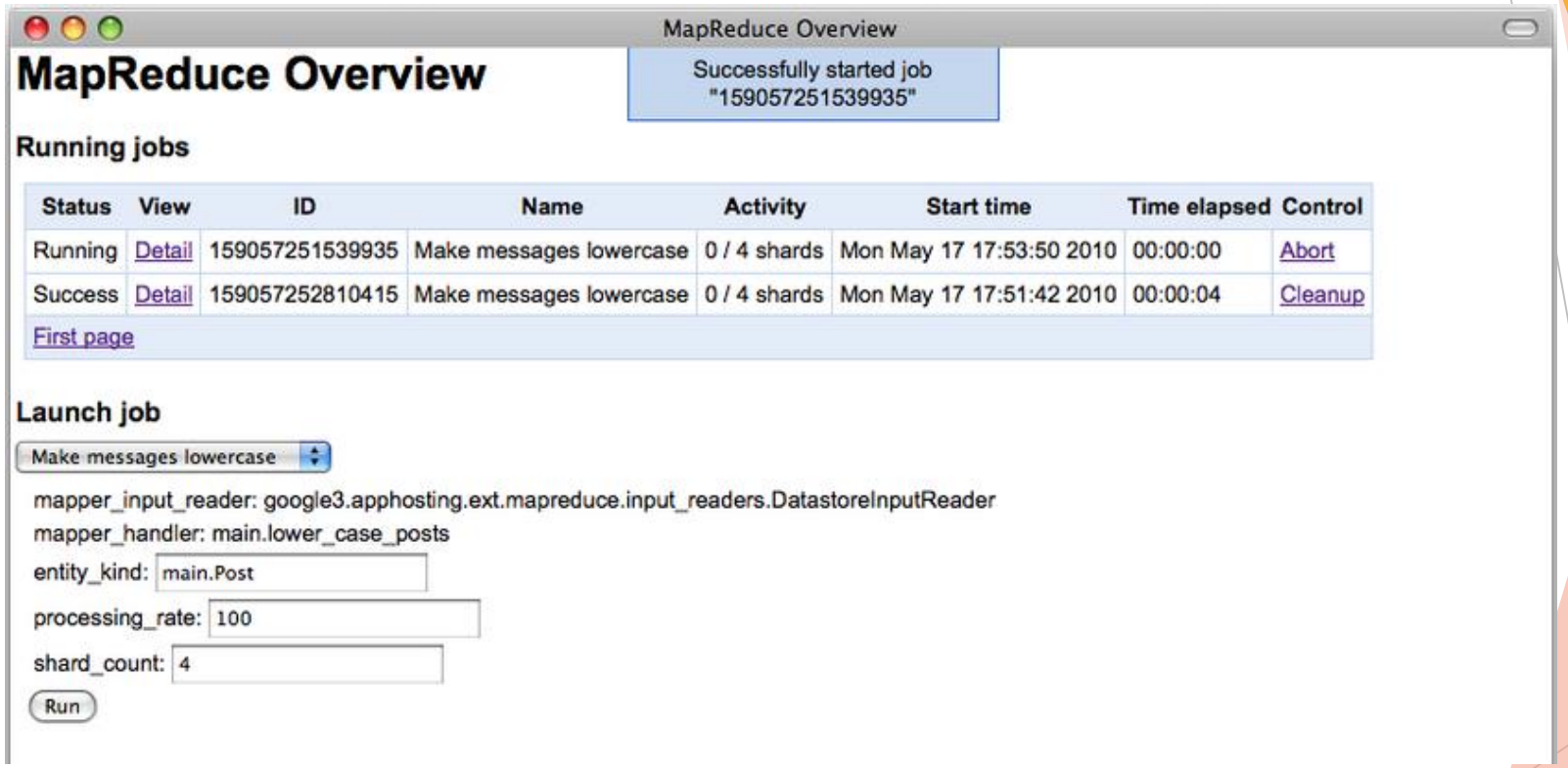
  @Override
  public void taskCleanup(Context context) {
    log.warning("Doing per-task cleanup");
  }

  @Override
  public void setup(Context context) {
    log.warning("Doing per-worker setup");
  }

  @Override
  public void cleanup(Context context) {
    log.warning("Doing per-worker cleanup");
  }

  // This is a silly mapper that's intended to show some of the capabilities of the API.
  @Override
  public void map(Key key, Entity value, Context context) {
    log.warning("Mapping key: " + key);
    if (value.hasProperty("skub")) {
      // Counts the number of jibbit and non-jibbit skub.
      // These counts are aggregated and can be seen on the status page.
      if (value.getProperty("skub").equals("Pro")) {
        context.getCounter("Skub", "Pro").increment(1);
      } else {
        context.getCounter("Skub", "Anti").increment(1);
      }
    }
  }
}
```

Lancement du MapReduce



Autres services

Search API

Push-To-Deploy

Experimental

Sockets API

Modules API

Cloud Endpoint

Multi tiers

Remote API

Traffic Splitting

Scheduled Tasks API

Autres services AppEngine

- XMPP
- App Identity
- Capabilities
- Remote API
- BulkLoading

Bibliothèques tierces

- Fantasm
- Objectify
- GeoModel, geodatastore
- …

Techniques avancées

Techniques avancées…

- … Ou comment appréhender DataStore quand on ne connaît (que) SQL !
  - Pratiques à adopter
  - Compteur distribué
  - Fan-in
  - Fan-out
  - GROUP BY, SUM, … ?

Pratiques à adopter

- MemCache, MemCache, et MemCache (10x + rapide)
- Interroger les Entity par Key, et non par requête
  - Encore mieux : plusieurs en parallèle
- Partitionner les données. Ne pas oublier que les Entities sont désérialisées en entier :
  - Ex: un fichier, son contenu et ses méta-données
- Pagination : Utilisez les Cursor
- Limiter la taille de Entity Groups
  - Ex: groupes correspondant aux données d'un seul utilisateur

Un compteur, c'est si simple !

- Pourquoi ne pas stocker le compteur dans une Entity ?
  - Souvenez-vous : 5 écritures max / seconde. Un compteur sous-entend beaucoup de « lire-incrémenter-écrire »
  - Compteur de connections : 5 nouveaux utilisateurs max / seconde ?
  - Contention !!! Failures !!!
- Solution possible : on distribue horizontalement le comptage
  - Requêtes // sur Entity Group distincts.
  - Donc on divise un compteur en N compteurs
  - Incrémentation : on en choisit un au hasard,
  - Interrogation : on les prends tous et on fait la somme.
- Pourquoi ça marche ?
  - AppEngine est bon pour la lecture en // sur des Group différents
  - Commutativité et associativité de l'addition

Un compteur, Implémentation

- Incremente()
  - ```
    long shardNum = generator.nextInt( numShards );
    Key shardKey = KeyFactory.createKey( «ShardCount», Long.toString(shardNum));
    Transaction tx = ds.beginTransaction();
    Entity thing;
    long value;
    try {
      thing = ds.get(tx, key);
      value = (Long) thing.getProperty( «count» ) + increment;
    } catch (EntityNotFoundException e) {
      thing = new Entity(key);
      value = 1;
    }
    thing.setUnindexedProperty(prop, value);
    ds.put(tx, thing);
    tx.commit();
    return value;
    ```
- getCount()
  - ```
    long sum = 0;
    Query query = new Query(«ShardCount» );
    for (Entity shard : ds.prepare(query).asIterable())
      sum += (Long) shard.getProperty(CounterShard.COUNT);
    return sum;
    ```
- Améliorations ??? Une idée ?

MemCache
Un autre compteur

- Si on n'a pas besoin d'un compteur précis
  - On garde le compte en MemCache
  - On incrémente par memcache.incr(key);
  - Un cron job récupère périodiquement la valeur de la cellule MemCache
  - Et incrémente une Entity Datastore.
- Autre utilisations de MemCache :
  - Stocker les pages fréquemment consultées.
  - Données temporaires avec écritures fréquentes.
  - Attention : l'application doit fonctionner si les données disparaissent.

Conclusion

- Platform-as-a-Service
  - Apprendre la plateforme,
  - Les contraintes sont souvent des opportunités
  - Travailler avec, ne pas se battre contre
- Développement très actifs
  - Nouveaux services régulièrement (NextGen queries, MapReduce, BigQuery, Cloud SQL, …)
- Une bonne façon pour apprendre à construire des systèmes qui passent la mise à l'échelle.

Références

- Ryan Barrett - Under the covers of the Google App Engine Datastore - http://sites.google.com/site/io/under-the-covers-of-the-google-app-engine-datastore
- Alfred R. Fuller - Next Gen queries