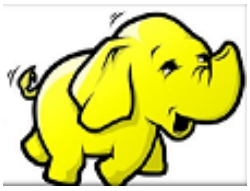
Architecture Hadoop



Nicolas Travers - CNAM

Historique

- Besoins de Google :
  - Stocker et traiter des Peta octets de données
  - Sur des milliers de nœuds
  - Méthode tolérante aux défaillances et simple à programme
- 2003 : Développement chez Google de
  - Google FS : Système de fichiers distribué et Tolérant aux pannes
  - MapReduce : Paradigme de programmation
- 2006 : Développement solution libre par fondation Apache
  - Hadoop FS : Système de fichiers distribué proche de Google FS
  - Hadoop = Hadoop FS + MapReduce

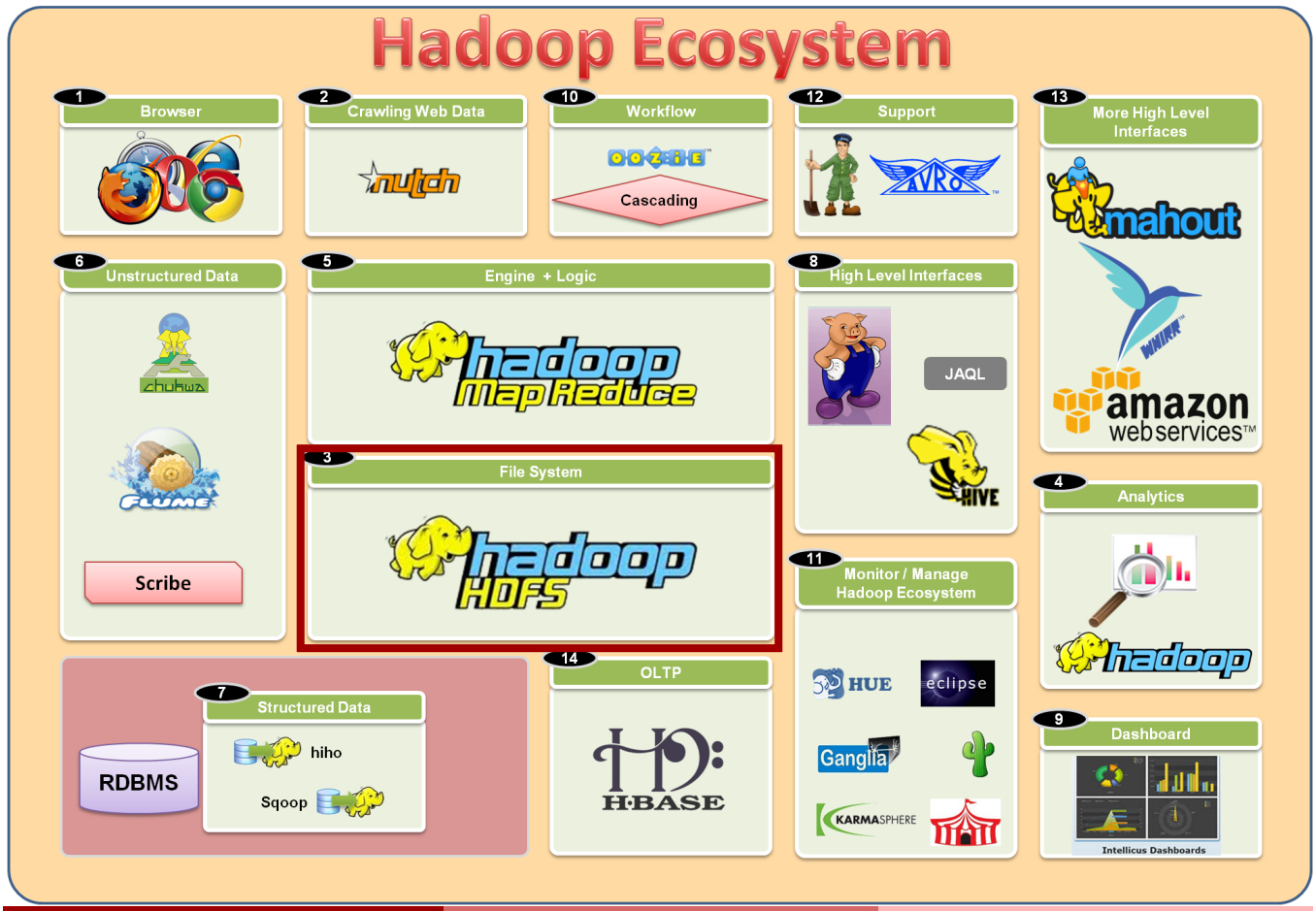
RDBMS vs Hadoop

RDBMS

- Giga octets de données
- Read and Write many times
- Schéma de données statique
- Haute intégrité des données
- Ne passe pas à l'échelle

Hadoop

- Peta octets de données
- Write once, Read many times
- Schéma de données dynamique
- Intégrité faible des données
- Passe à l'échelle



Hadoop File System

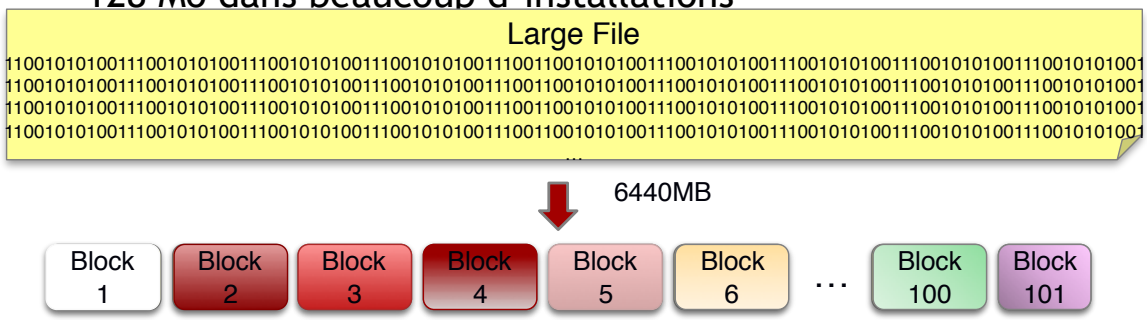
- Objectifs :
  - Passage à l'échelle (gestion plusieurs milliers de nœuds)
    - Exemple : Yahoo! Utilise Hadoop sur 4000 nœuds pour stocker des petabytes de données
  - Tolérance aux pannes (hardware et software)
    - Utilisation matériel conventionnel
    - Mot d'ordre : la panne est la règle et non le cas rare
  - Gestion de fichiers de grande taille
  - Accès données : Write once/Read Many times

Hadoop File System

- HDFS n'est pas fait pour les utilisations ci-dessous :
  - Accès aux données avec faible latence
    - HDFS optimisé pour traiter un grand volume de données
  - Grand nombre de petits fichiers
    - Meta-données des fichiers chargés en mémoire centrale (jusqu'à plusieurs dizaine de millions de fichiers par nœud)
  - Ecritures arbitraires, écrivains multiples
    - HDFS conçu pour écrivains uniques, et écriture en fin de fichiers

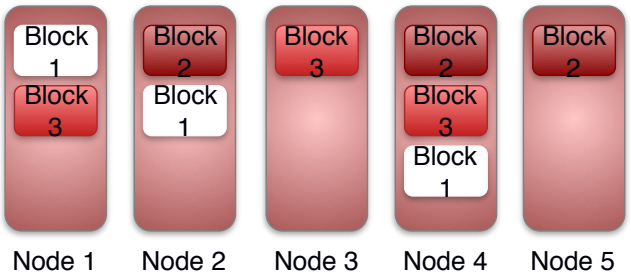
HDFS : Découpage des fichiers

- Utilisation concept blocs de données
  - Comme pour tout système de fichiers
- Fichiers découpés en blocs
  - Permet stockage distribué de gros fichiers
  - Chaque bloc stocké dans un fichier sur système de fichiers local
- Taille bloc : 64 Mo (par défaut)
  - 128 Mo dans beaucoup d'installations



HDFS : Placement des blocs

- Tolérance aux pannes et haute disponibilité
  - ➡ Placement noeuds différents
  - ➡ Réplication de chaque bloc sur 3 noeuds différents
    - 1er copie : sur noeud créant fichier
    - 2e copie : sur noeud aléatoire sur rack différent
    - 3e copie : sur noeud différent, même rack que 1er copie



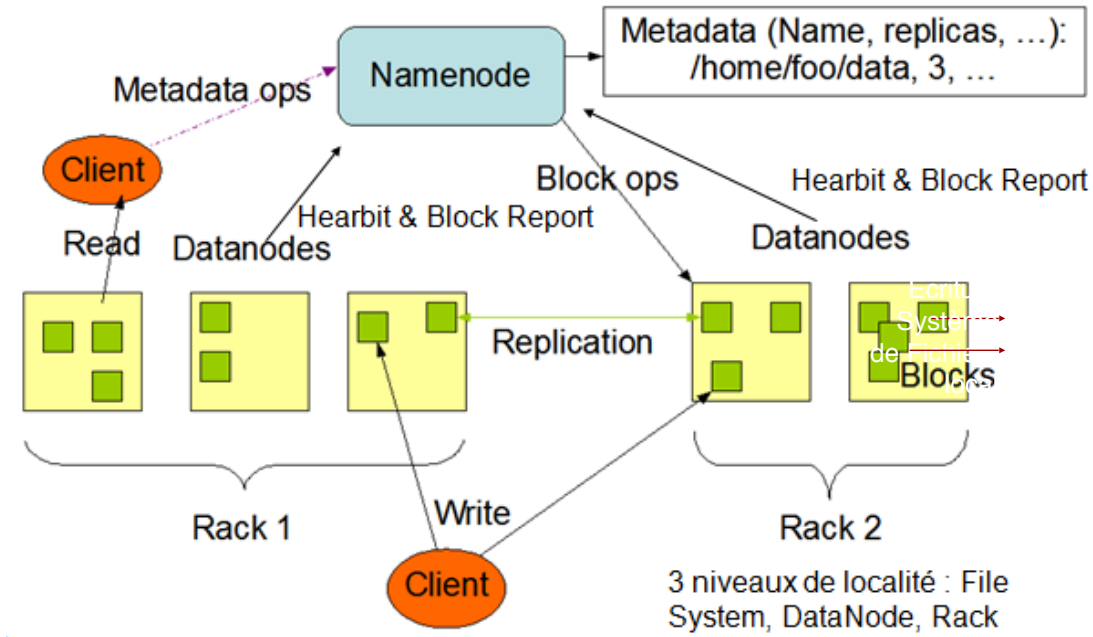
HDFS : Placement des blocs

- Tolérance aux pannes et haute disponibilité
  - ➡ Placement noeuds différents
  - ➡ Réplication de chaque bloc sur 3 noeuds différents
    - 1er copie : sur noeud créant fichier
    - 2e copie : sur noeud aléatoire sur rack différent
    - 3e copie : sur noeud différent, même rack que 2e copie
- Compromis entre fiabilité et bande passante
  - Réplication sur nœuds différents
    - meilleure bande passante mais aucune tolérance aux pannes
  - Réplication sur datacenter/rack différents
    - bonne tolérance aux pannes mais faible bande passante

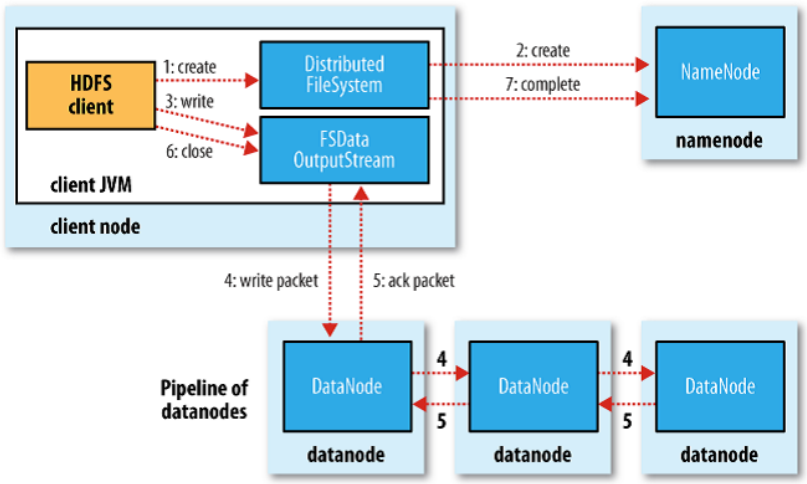
HDFS : Types de noeuds

- 2 catégories de noeuds dans HDFS
  - Namenode (maître) / Backup Node
  - Datanodes (esclaves)
- **Namenode** : Gère système de fichiers dont il a la charge
  - 1 seul Namenode par cluster
  - Maintient arborescence et meta-données sur Système de Fichiers
    - 2 type de fichiers : Namespace et Logs
  - Stocke liste des datanodes sous son contrôle
- **Backup Node** : Sauvegarde de l'état du Namenode
- **Datanodes** : Stockage des fichiers et exécution des tâches
  - Une instance sur chaque noeud du cluster (1 à 4000 noeuds)
  - Traitement des requêtes des clients
  - Informent périodiquement le *namenode*
    - blocs stockés
    - avancement des tâches

HDFS : Architecture

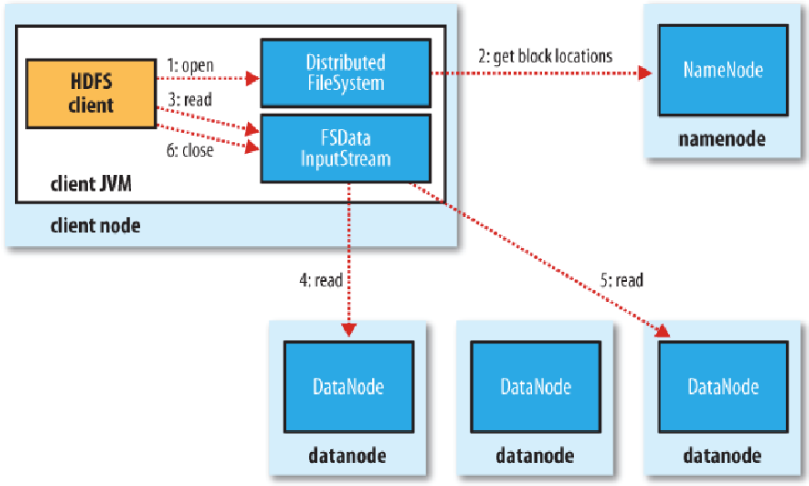


HDFS : Ecriture de données



- En cas de non confirmation (5 – Ackowledge) d'un ou plusieurs Datanodes
  - Aucun problème tant que répliqué au moins sur un Datanode
  - Namenode lance de façon asynchrone processus réplication

HDFS : Lecture de données



- Lectures réalisées directement sur Datanodes
  - guidé par le namenode qui indique le meilleur datanode à contacter

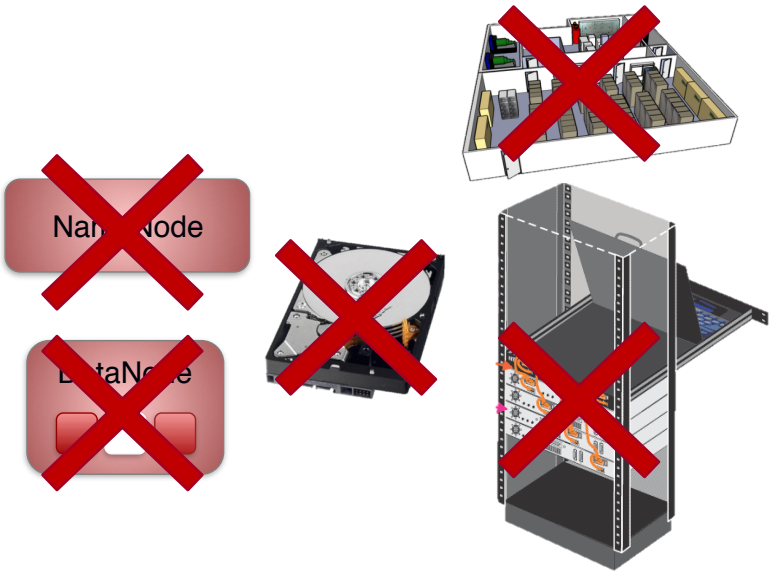
➡Permet scalabilité en répartissant les requêtes entre les Datanodes du cluster

HDFS : Gestion des droits

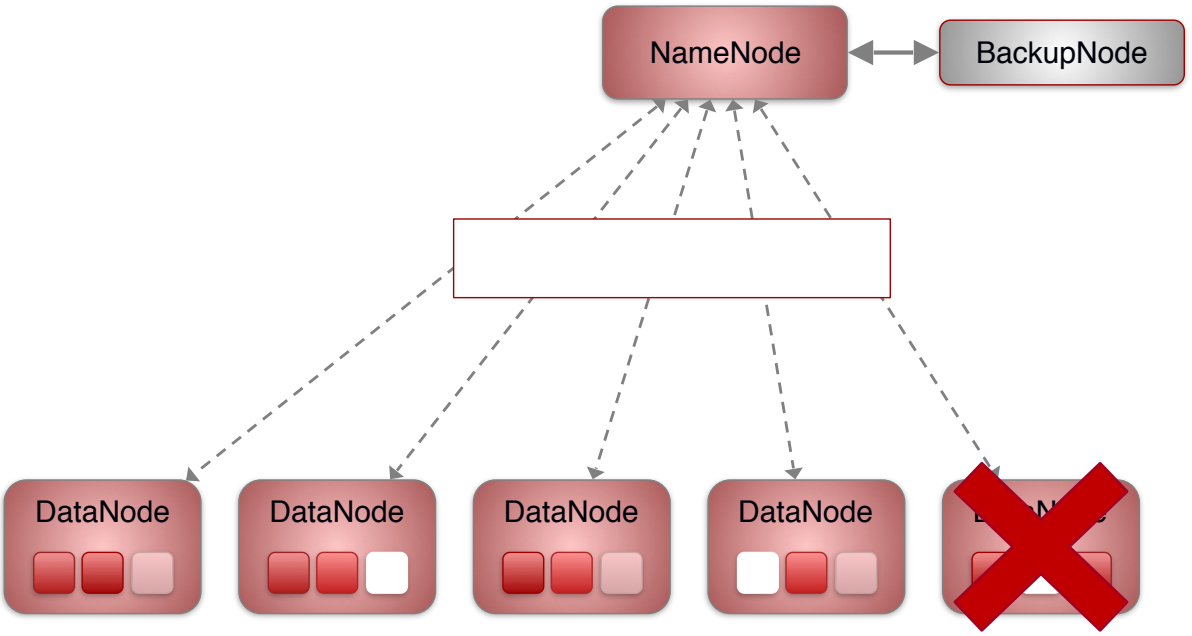
- Permissions proches de POSIX
  - Lecture (r), écriture (w) et exécution (x)
  - Permission exécution ignorée pour fichiers
    - Exécution tâches et non fichiers sur HDFS
  - Permission exécution utilisée sur repertoire pour accéder à son contenu
    - Répertoires non stockés sur datanodes (uniquement namenode avec meta-données)
- Propriétaire fichier
  - compte utilisé par utilisateur
  - défini droit accès aux fichiers
  - Namenode : “superuser”

HDFS : Tolérance aux pannes

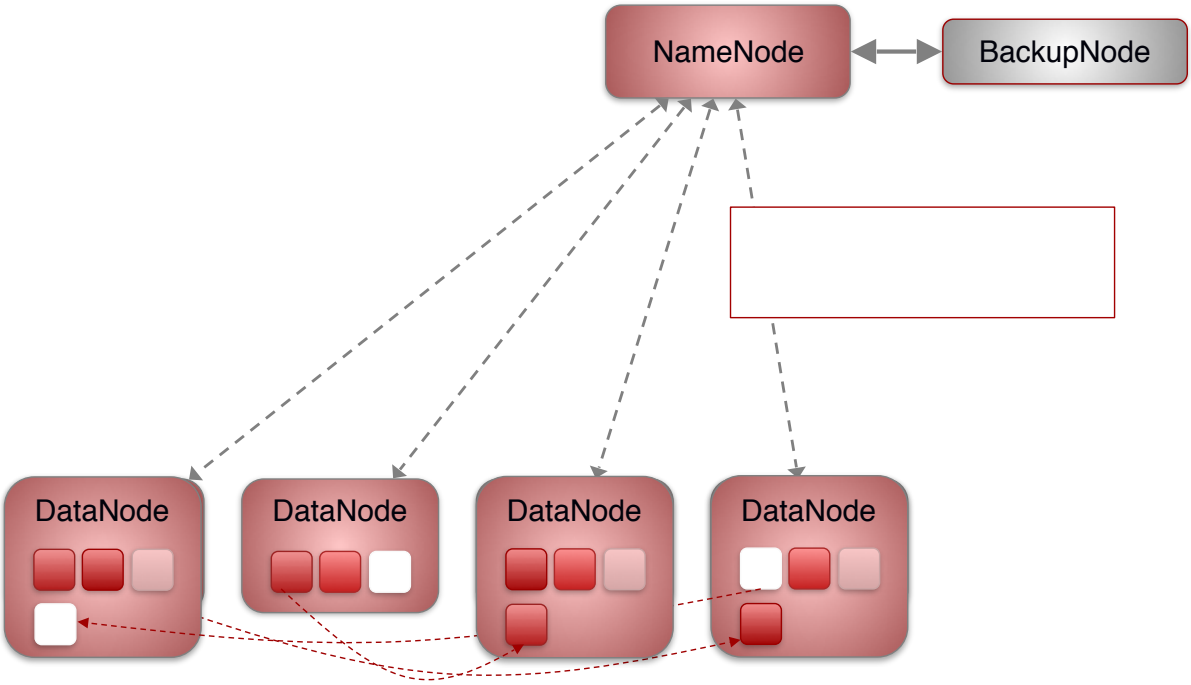
- HDFS conçu avec le postulat que pannes sont la norme
- Types de pannes
  - Disque
  - Switch/Rack
  - DataNode
  - NameNode
  - Datacenter



HDFS : Défaillance de Datanode



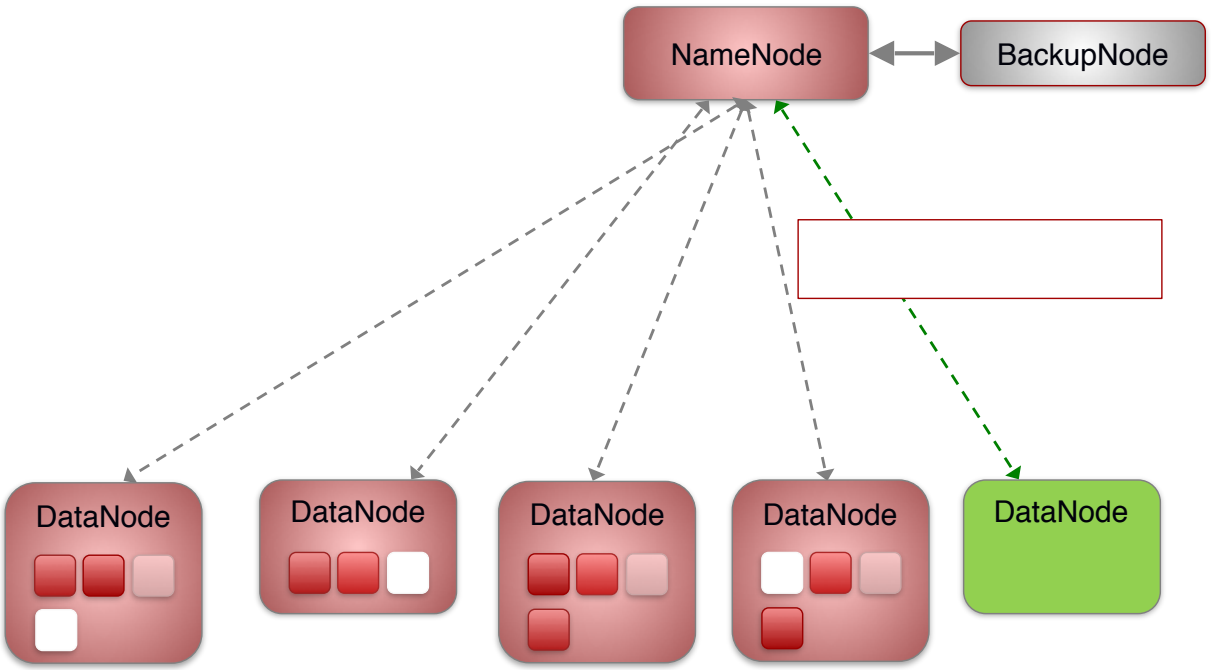
HDFS : Défaillance de Datanode



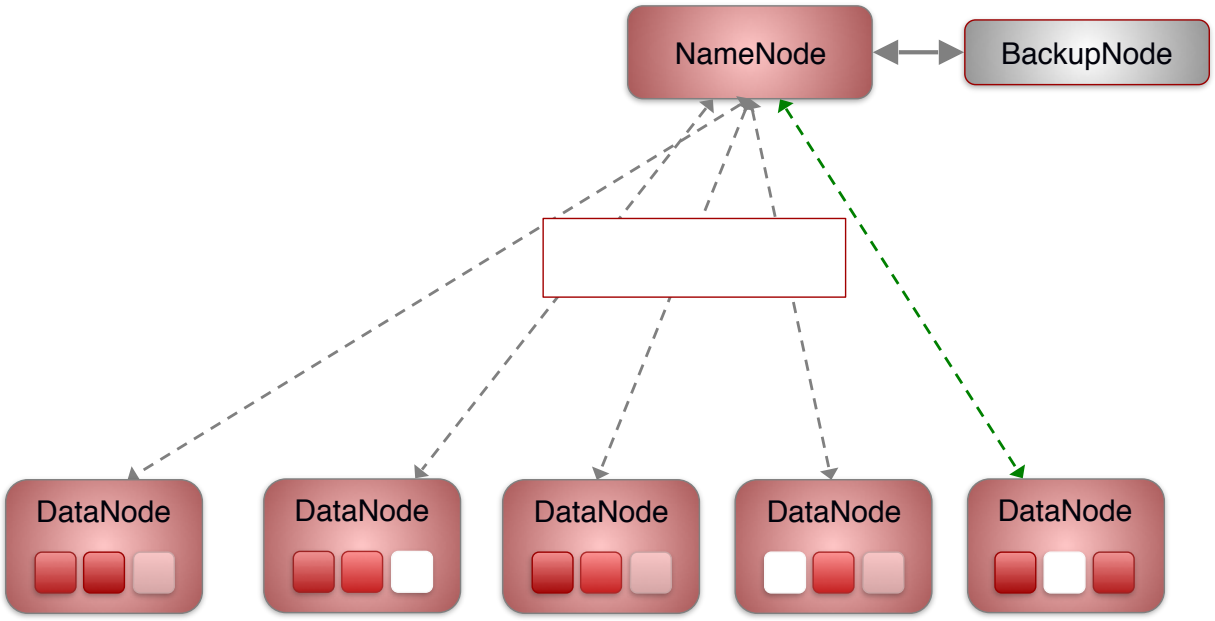
HDFS : Panne Namenode (2/2)

- Namenode est un point de défaillance
  - Seul à lancer les tâches
  - Seul à posséder une image du Système de fichiers
- 30 mins pour démarrage d'un namenode
  - Dialogue avec datanodes pour établir le namespace
  - Problème pour disponibilité
- Mise en place 2e Namenode (dernière version HDFS) mais besoin :
  - Partager image du Système de fichiers
  - Envoyer rapports aux 2 Namenodes par Datanodes
    - infos stockées en mémoire et non sur disque namenodes
  - Clients doivent avoir mecanisme pour envoyer tâche aux 2 Namenodes (de façon transparente)

HDFS : Scaling horizontale



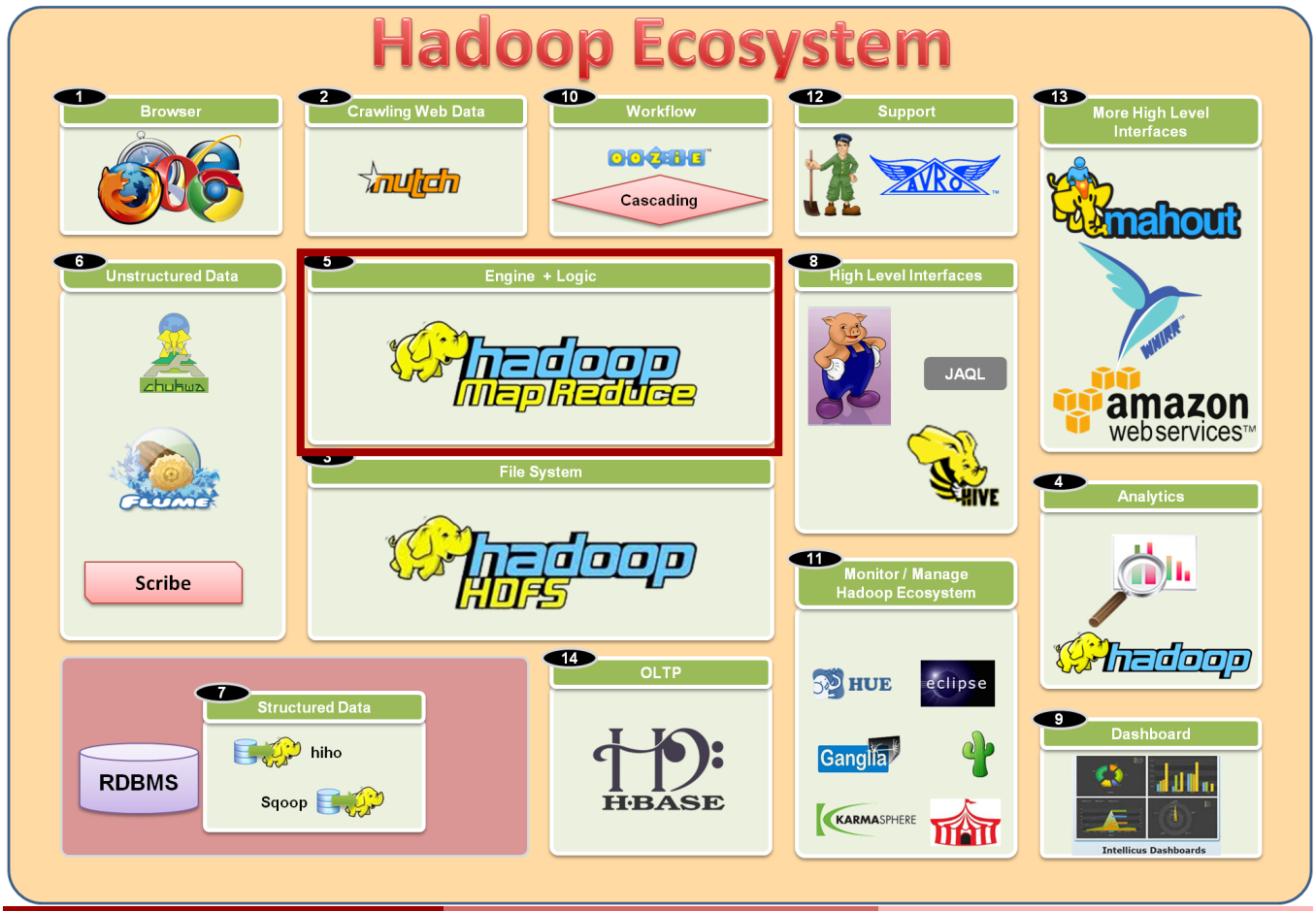
HDFS : Scaling horizontale



HDFS : Import de données

- Divers outils déjà existant
  - Apache Flume (*http://incubator.apache.org/flume/*)
    - Import fichiers de log serveur pour analyse
  - Apache Sqoop (*http://sqoop.apche.org*)
    - Import données issues de base de données
  - distcp
    - Transfert de données entre 2 clusters HDFS (même version)

```
Hadoop distcp [-update] [-overwrite] hdfs://
  namenode1/dir1 hdfs://namenode2/user/jdo
```

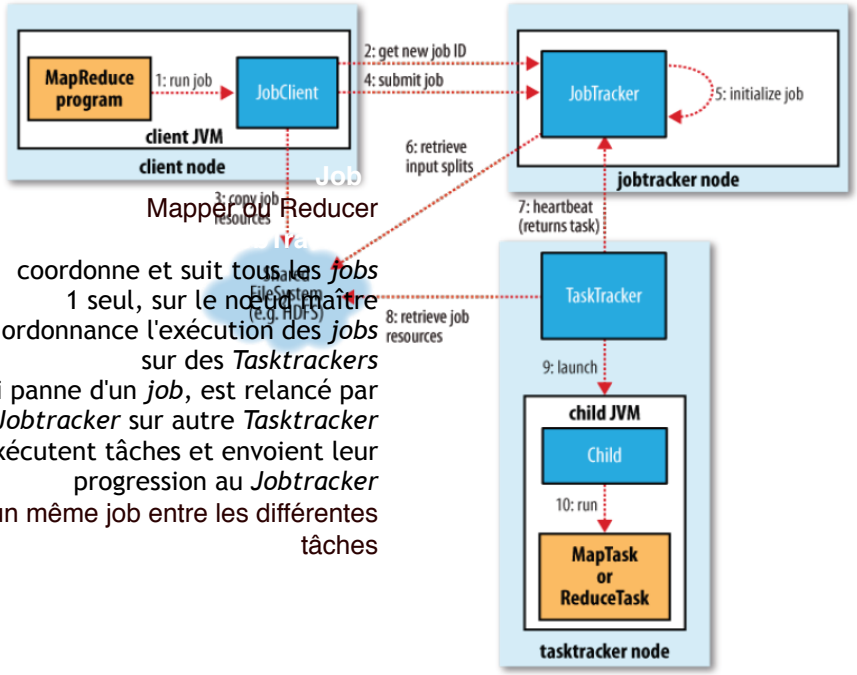



MapReduce : Proportion Mappers/Reducers

- Nombre de Reducers inférieur aux nombre de Mappers
- Bonnes performances
  - Garder un bon équilibre entre Mappers/Reducers
  - Nombre de Mappers dépend :
    - Taille données à traiter
    - Complexité du traitement appliqué
- Résultats intermédiaires sont triés avant envoie aux reducers
  - Si plusieurs Reducers, Mappers construisent partitions pour chaque Reducer
- Résultats des Reducers sont répliqués sur HDFS
  - dont une copie en locale

MapReduce : Fonctionnement

- Soumission d'un programme MapReduce



- Mapper ou Reducer
- coordonne et suit tous les *jobs*
- 1 seul, sur le nœud maître
- ordonnance l'exécution des *jobs* sur des *Tasktrackers*
- Si panne d'un *job*, est relancé par *Jobtracker* sur autre *Tasktracker*

lancent et exécutent tâches et envoient leur progression au *Jobtracker*

partage les fichiers d'un même job entre les différentes tâches

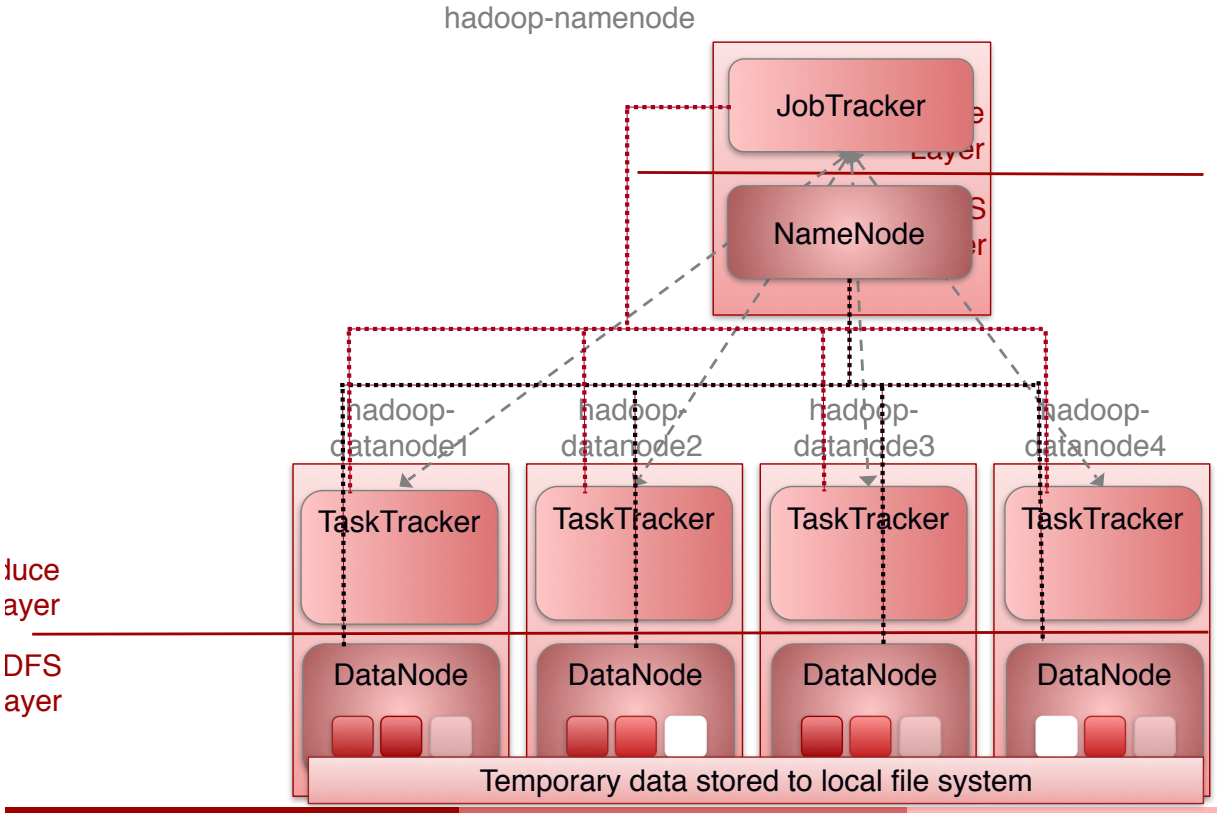
MapReduce : TaskTrackers

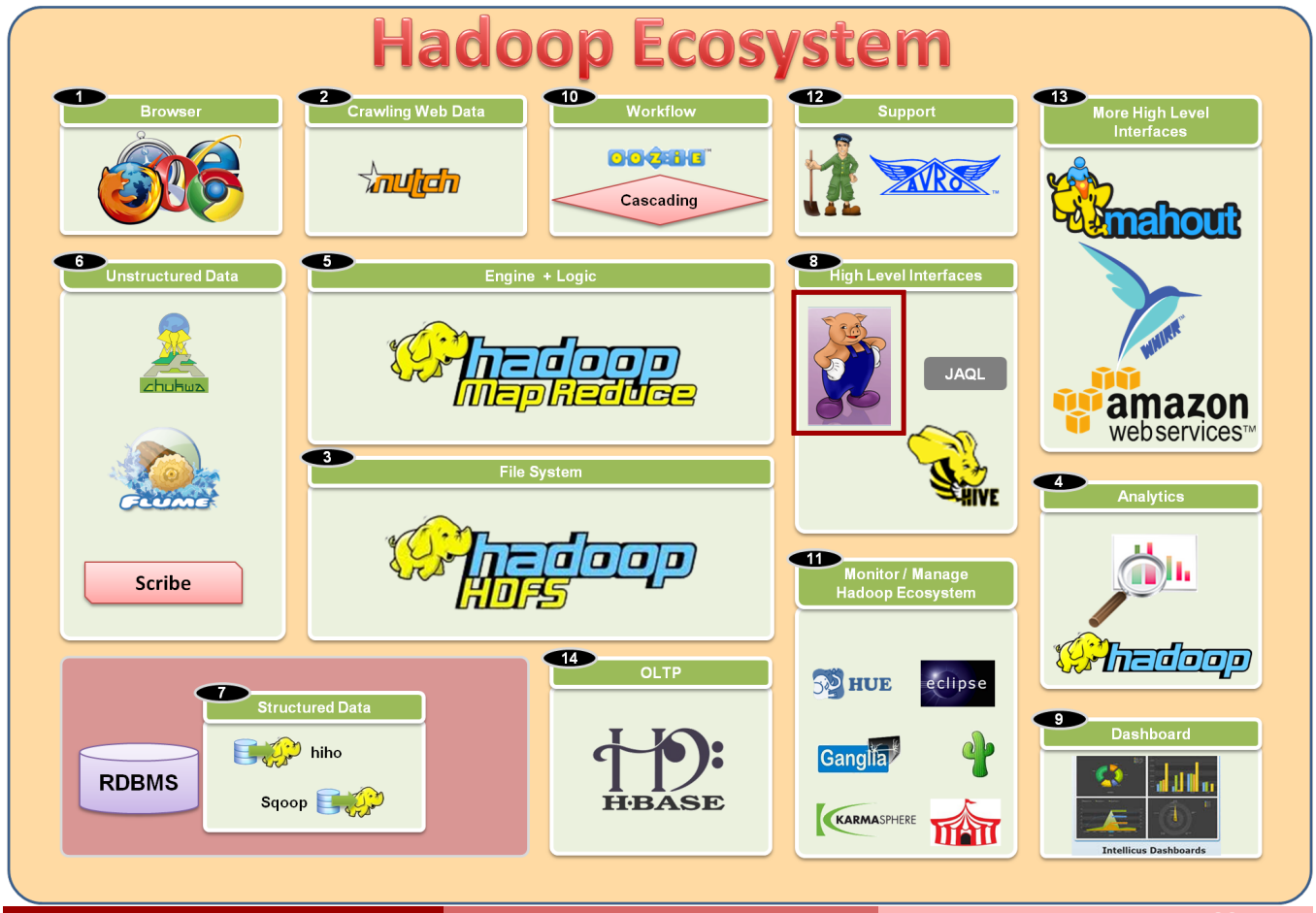
- Envoie périodique de messages heartbeat au Jobtracker
  - Au moins toutes les 5 secondes
  - Informe disponibilité pour exécuter nouvelle tâche
  - Indique l'état d'avancement des tâches en cours d'exécution
- *TaskTrackers* peuvent exécuter plusieurs tâches Map et Reduce
  - Nombre : dépend des ressources CPU et mémoire du noeud
  - Exécution en priorité tâche *Map*
- Assignement des tâches dépend de la localisation des données
  - Noeud avec slot disponible
  - Plus proche des données

MapReduce : Prétraitement et Optimisation

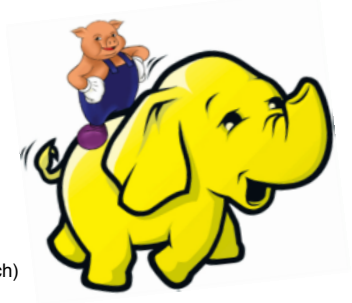
- Programmes MapReduce limités par débit réseau
- Utilisation TaskCombiner
  - Prétraitement sur résultats des Mappers avant envoie aux Reducers
  - Ex: Max(0,20,10,25,15) = Max(Max(0,20,10),Max(25,15))=25
    - Pas toujours possible d'utiliser fonction Combiner sur Mappers
    - Dépend de l'associativité et la commutativité du Reduce
- Vaut la peine de vérifier pour gagner en performance
  - gain temps et économie bande passante réseau

MapReduce et HDFS





Pig Latin



- **"Pig Latin: A Not-So-Foreign Language for Data Processing"**
  Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, Andrew Tomkins (Yahoo! Research)

- Langage de haut niveau pour Map/Reduce
  - Gestion de flux de données
  - Moteur d'exécution parallèlisé sur Hadoop
  - Compilateur produisant des séquences de programme Map/Reduce
  - Opérations sur HDFS
  - Métadonnées non obligatoires
- Avantages
  - Parallélisation triviale
  - Optimisation transparente
  - Extensibilité via création de fonctions

Script Pig

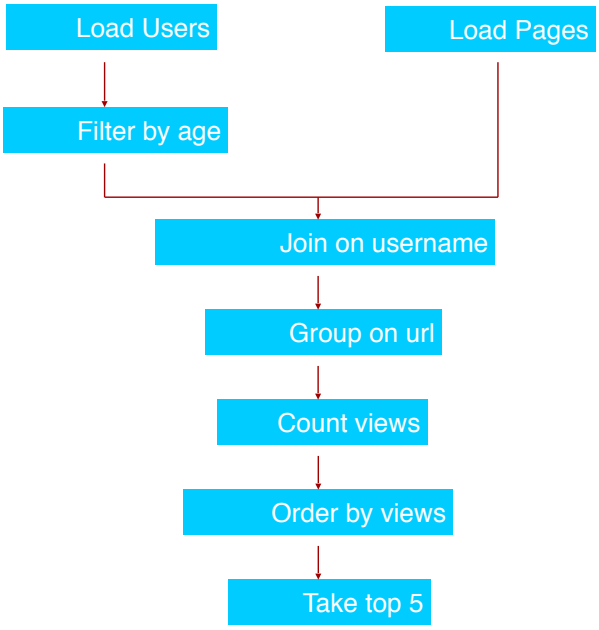
D:\1_TheFifthElephant_2012_Hands-on_Intro_to_Pig\top_5_sites.pig - Sublime Text 2

File Edit Selection Find View Goto Tools Project Preferences Help

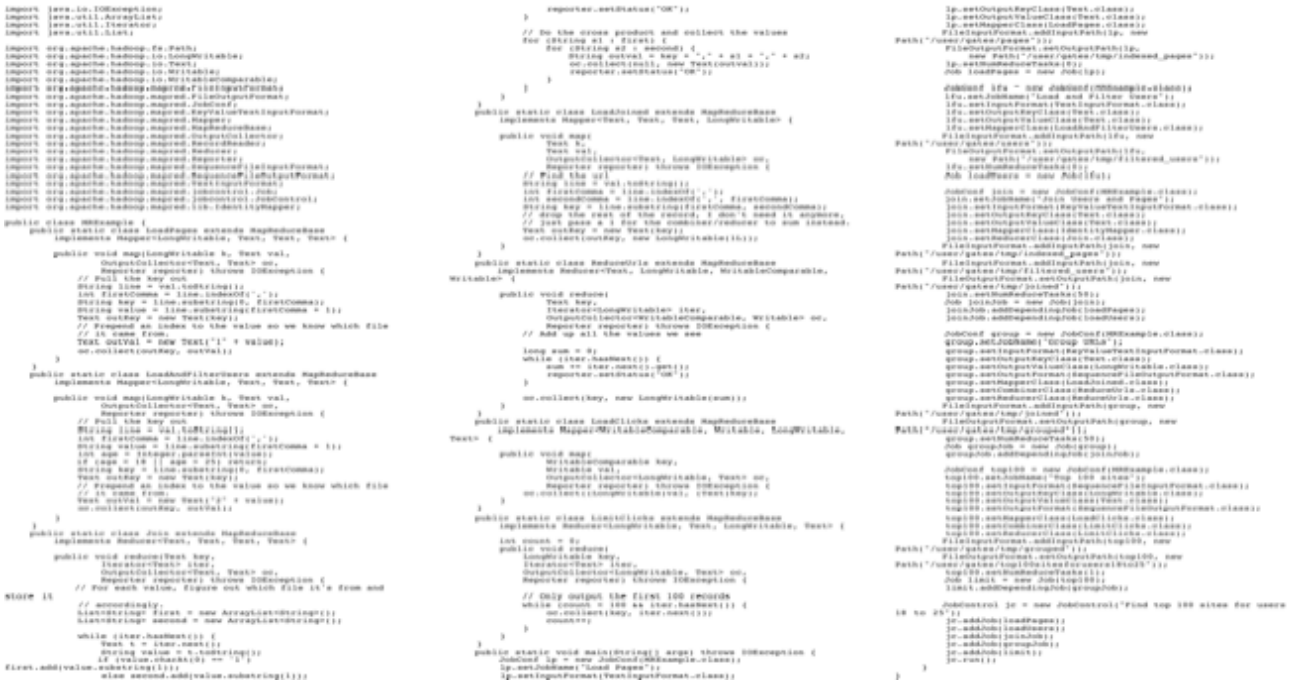
top_5_sites.pig

```
1  users = load 'users.csv' as (username:chararray, age:int);
2  users_1825 = filter users by age >= 18 and age <= 25;
3
4  pages = load 'pages.csv' as (username:chararray, url:chararray);
5
6  joined = join users_1825 by username, pages by username;
7  grouped = group joined by url;
8  summed = foreach grouped generate group as url, COUNT(joined) as views;
9  sorted = order summed by views desc;
10 top_5 = limit sorted 5;
11
12 store top_5 into 'top_5_sites.csv';
13
```

Pig Latin : plan compilé



Correspondance en Map/Reduce



Pig : Types de données

- Int, float (32 bits)
- Long, double (64 bits)
- Chararray
- Bytearray
- boolean

- Tuple
  - Attributs ordonnés
- Bag
  - Collection de tuples
- Map
  - Ensemble de pairs clés/ valeurs

Pig : commandes

- Load : Lecture de données
- Store : Enregistrement de données
- Dump : Ecriture dans sortie standard
- Foreach : Appliquer sur chaque enregistrement
- Filter : Prédicat sur chaque enregistrement
- Join, Order, Group, Cogroup : Sur la clé
- Distinct : Retire les doublons
- Union : Fusion de données
- Limit : Réduit la taille du résultat
- Split : Créé 2 ou + collections, sur condition

https://pig.apache.org/docs/r0.7.0/piglatin_ref1.htm

Pig : exemples

- Typer le fichier d'entrée

```
A = LOAD 'input' AS (name: chararray, age: int, gpa: float);
B = FILTER A BY $1 == 30;
C = GROUP A By $0;
```

- Compter & ordonner les accès par 'host'

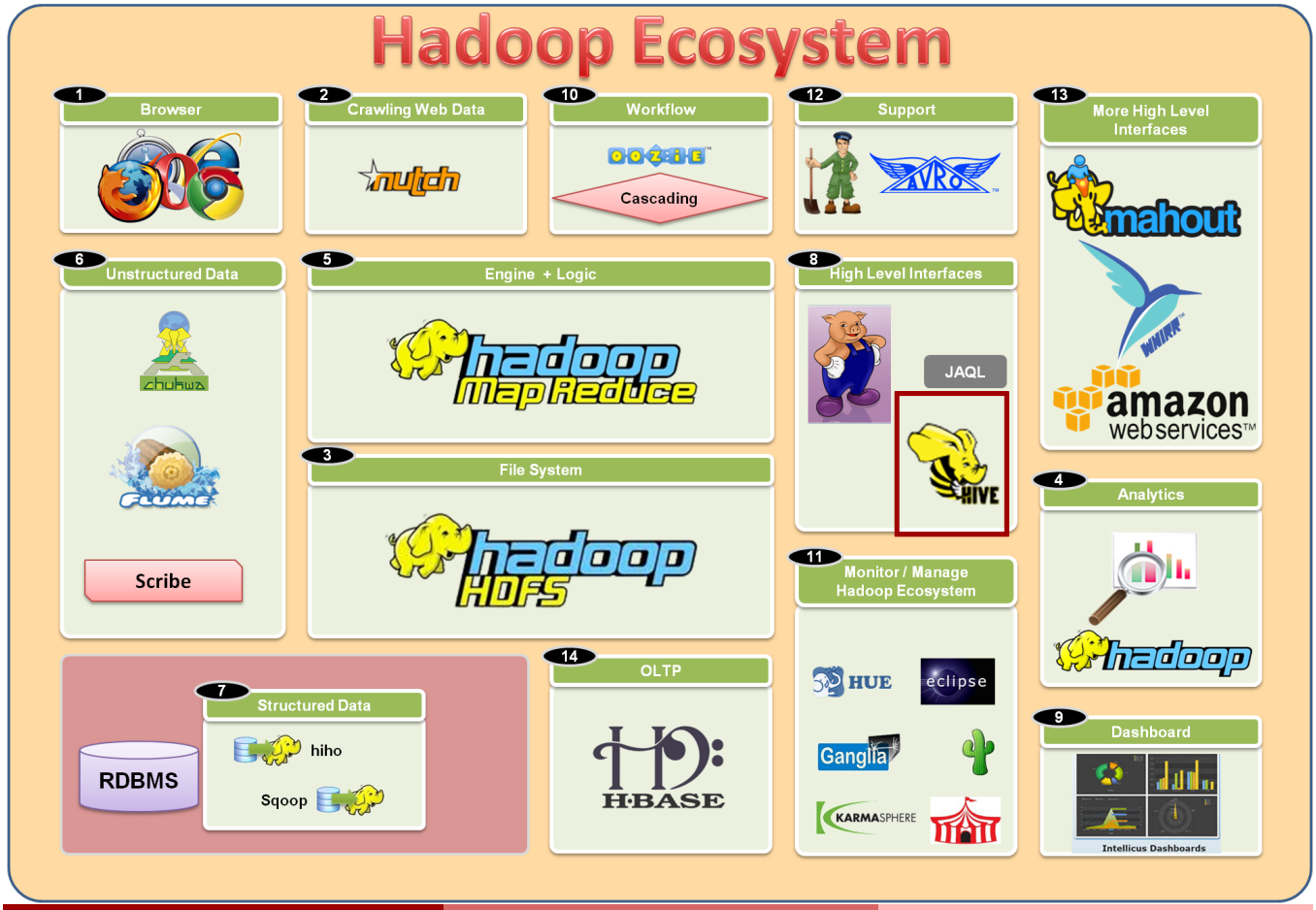
```
A = FOREACH www_access GENERATE host;
B = FOREACH (GROUP A BY host) GENERATE group AS host, COUNT($1) AS cnt;
                    //$1 : 2° colonne de 'A' (www_access)
OUT = ORDER B BY cnt DESC;
```

- Jointure multiple

```
big = LOAD 'big_data' AS (b1,b2,b3);
tiny = LOAD 'tiny_data' AS (t1,t2,t3);
mini = LOAD 'mini_data' AS (m1,m2,m3);
C = JOIN big BY b1, tiny BY t1, mini BY m1 USING "replicated";
                    //(Replicated – Memoire / Skewed – histogrammes / Merge – tri)
```

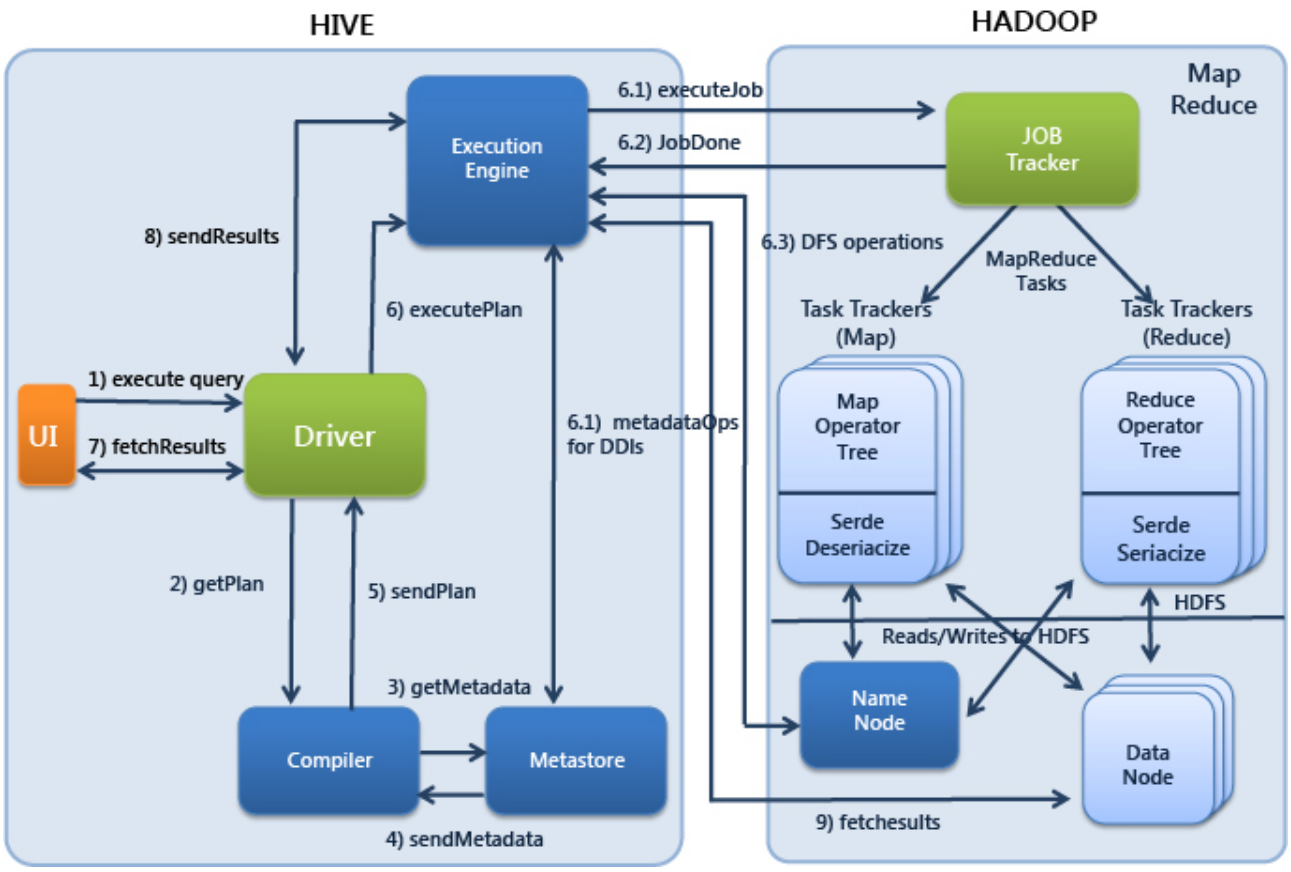

Composant de Pig

- Pig Latin
  - Création de scripts
- Grunt
  - Console pour Pig
- PigServer
  - Driver / Class Java pour Pig
  - Equivalent à JDBC



Hive

- Module Hadoop pour “SQL like”
  - Permet d’interroger et manipuler les données
  - SQL pour Map/Reduce
    - Collections
    - Filtres (Map)
    - Jointures (HashJoin en mémoire)
    - Agrégations (Reduce)
  - Gestionnaire de Méta-données (Metastore)



Hive : MetaStore

- Métadonnées Tables/Partitions
  - Schéma des tables
  - Librairie Sérialisation/Désérialisation : *SerDe*
  - Localisation HDFS
  - Partitionnement logique des clés
  - Autres
- API *Thrift*
  - Interface de programmation pour Hive/Hadoop
  - Php (Interface Web), Python (old CLI), Java (Query Engine and CLI), Perl (Tests)
- Stockage méta-données : texte ou SQL

Hive Cli (Command line)

- LDD :
  - create table/drop table/rename table
  - alter table add column
  - https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML
- Navigation :
  - show tables
  - describe table
  - cat table
- Chargement des données
- Requêtes

Hive : Langage de requêtes

- Requêtes SQL + Streaming Hadoop
- Opérateurs de bases :
  - Projections
  - Equijointures
  - Group by
  - Cogroups (= Cluster tables sous Oracle)
  - Sampling
- Sorties :
  - Données en flux aux mappers/reducers
  - Stockage dans une autre table
  - Fichiers HDFS
  - Fichier local

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select

Hive : Jointures

- Joins

```
FROM page_view pv JOIN user u ON (pv.userid = u.id)
INSERT INTO TABLE pv_users
SELECT pv.*, u.gender, u.age
WHERE pv.date = '2014-03-03';
```

- Outer Joins

```
FROM page_view pv FULL OUTER JOIN user u ON
        (pv.userid = u.id) INSERT INTO TABLE pv_users
SELECT pv.*, u.gender, u.age
WHERE pv.date = '2014-03-03';
```


Hive : Agrégation & Insertions multiples

```
FROM pv_users
INSERT INTO TABLE pv_gender_uu
  SELECT pv_users.gender, count(DISTINCT pv_users.userid)
  GROUP BY(pv_users.gender)
INSERT INTO TABLE pv_ip_uu
  SELECT pv_users.ip, count(DISTINCT pv_users.id)
  GROUP BY(pv_users.ip);
```

Hive : Intégration de scripts

```
FROM (
  FROM pv_users
  SELECT TRANSFORM(pv_users.userid, pv_users.date)
    USING 'map_script'
  AS(dt, uid)
  CLUSTER BY(dt)) map
INSERT INTO TABLE pv_users_reduced
  SELECT TRANSFORM(map.dt, map.uid) USING
    'reduce_script' AS (date, count);
```