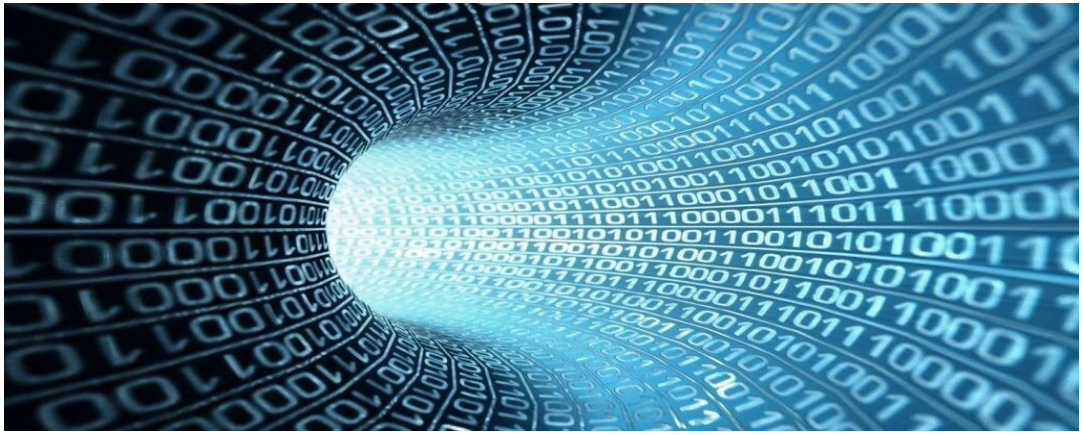
MIF18
Mise en œuvre de MapReduce

Damien Simonin Feugas



Présentations

Damien SIMONIN FEUGAS

- 2000 – IUT d'Informatique Aix en Provence
- 2002 – INSA Lyon, IF
- 2006 – Worldline, Unité Telecom Utilities Media
- 2010 – Worldline, R&D
- 2013 – Worldline, Unité BI & Big Data

Worldline

- Leader mondial des services transactionnels de paiement
- Construit et héberge les solutions de ses clients
- eCommerce : Auchan Drive, Carrefour, McDonald
- Banques : BNP, Société Générale, Boursorama
- Telecom & Média : Orange, SFR, M6, TF1
- Secteurs publics : IGN, SNCF, Météo France

Plan

1- Principes de scalabilité des traitements

2- Hadoop : HDFS, YARN & implémentation MapReduce

3- MongoDB : Topologie et MapReduce

4- Etude de cas


Scalabilité
des traitements

Problématiques liées au systèmes distribués

"Data Locality"

- Déplacer le code là où se trouve la donnée
- Limiter l'impact réseau
- Moins coûteux que de déplacer la donnée

Accès local séquentiel

- Traitement batch
- Limiter la latence disque
- Parcours systématique de toute la donnée

"Embrace failure"

- Inévitable dans un cluster
- Détection des pannes
- Stratégie de reprise

Recherche de scalabilité

- Parallélisation
- Plus de machines doit induire plus de puissance/stockage

Principes de la scalabilité des traitements

Décomposer l'algorithme

- En fonctions parallélisables

  ⇨ Fini la programmation séquentielle !

- Distribuer et chainer les fonctions

  ⇨ Exécution sur un sous-ensemble de la donnée

Restrictions de cette approche

- Parallélisation implique coordination

  ⇨ Plus il y a de programmes en parallèle, plus il faut coordonner leur exécution

- Tous les problèmes ne peuvent être décomposés de la sorte

worldline e-payment services

La séparation des responsabilités

Le rôle du framework d'exécution

- Distribuer les fonctions (le code) là où réside la donnée
- Lire la donnée et exécuter les fonctions dessus
- Stocker/Partager les résultats intermédiaires
- Synchroniser l'exécution des fonctions au sein du cluster
- Détecter les pannes et relancer les fonctions manquantes

Le rôle du développeur

- Maitriser la structure des données
- Décomposer son algorithme en fonctions parallélisables

  ⇨ Pas d'état persistent, pas de goulots d'étranglement
- Respecter (et comprendre) le contrat d'interface du framework

  ⇨ Rôle de la clé en sortie du map


hadoop

Simplified Data Processing on Large Clusters

Le papier (2004, Dean & Ghemawat)

- Google « open-source » son savoir faire
- 100 To traités en 2004
- 21 Po traités en 2010 (Facebook)
- +100 Po en 2011 (Yahoo)

Dates clés

- 2005 – création par Mike Cafarella & Doug Cutting (jouet)
- Octobre 2009 – ouverture du code et reprise par la fondation Apache
- 2011 – Yahoo donne naissance à HortonWorks
- Mai 2012 – refonte de MR et création de YARN

Pour quoi faire ?

- Page Ranking (liens vers les pages)
- Anti-spam
- Recommandations

Deux composantes fondamentales

HDFS (Hadoop Distributed File System)

- Est un ensemble d'exécutables Java
- Distribué sur plusieurs serveurs banals (Commodity hardware)
- Décompose les fichiers en blocks
- Assure la réplication des blocks (Durabilité)
- Ne fait pas d'accès aléatoire
- Optimisé pour le débit, pas pour la latence

YARN (Yet Another Resource Negociator)

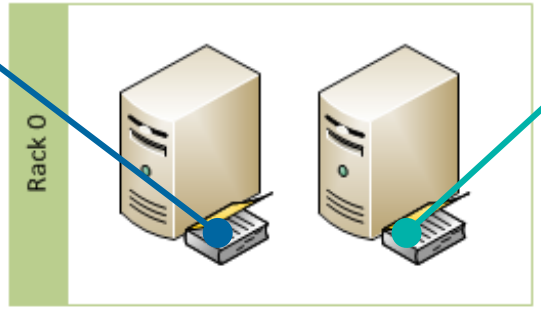
- Est un ensemble d'exécutables Java
- Distribué sur plusieurs serveurs banals (Commodity hardware)
- Distribue les fonction d'un algorithme là où est stockée la donnée manipulée
- Ordonnance l'exécution des fonctions
- Stocke et déplace les résultats intermédiaire le cas échéant
- Relance les fonctions ayant échouées

Topologies déployées sur les même serveurs

Organisation des exécutables HDFS

Namenode

- Annuaire de blocks
- Répond au clients
- Ordonnance la réplication

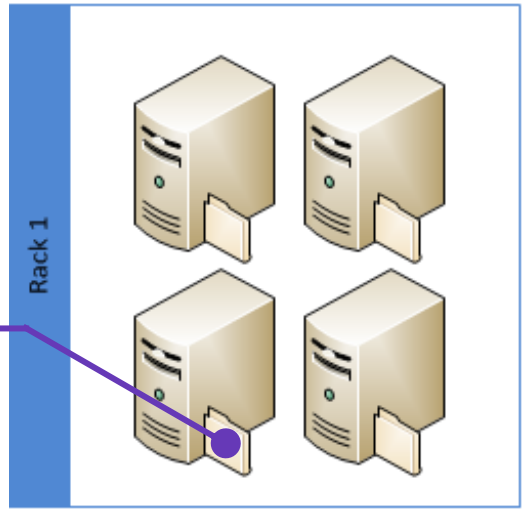


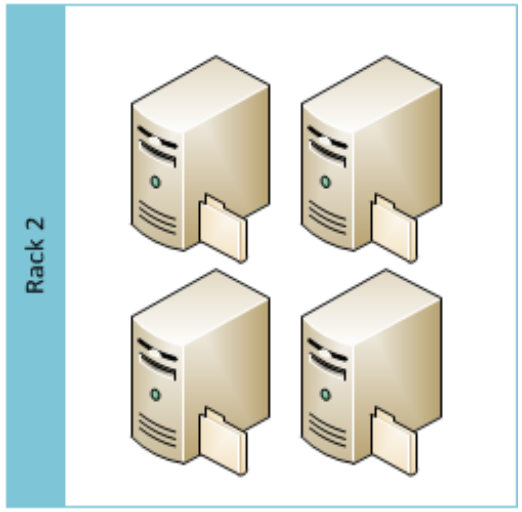
Secondary Namenode

- Instantannés de la mémoire du namenode

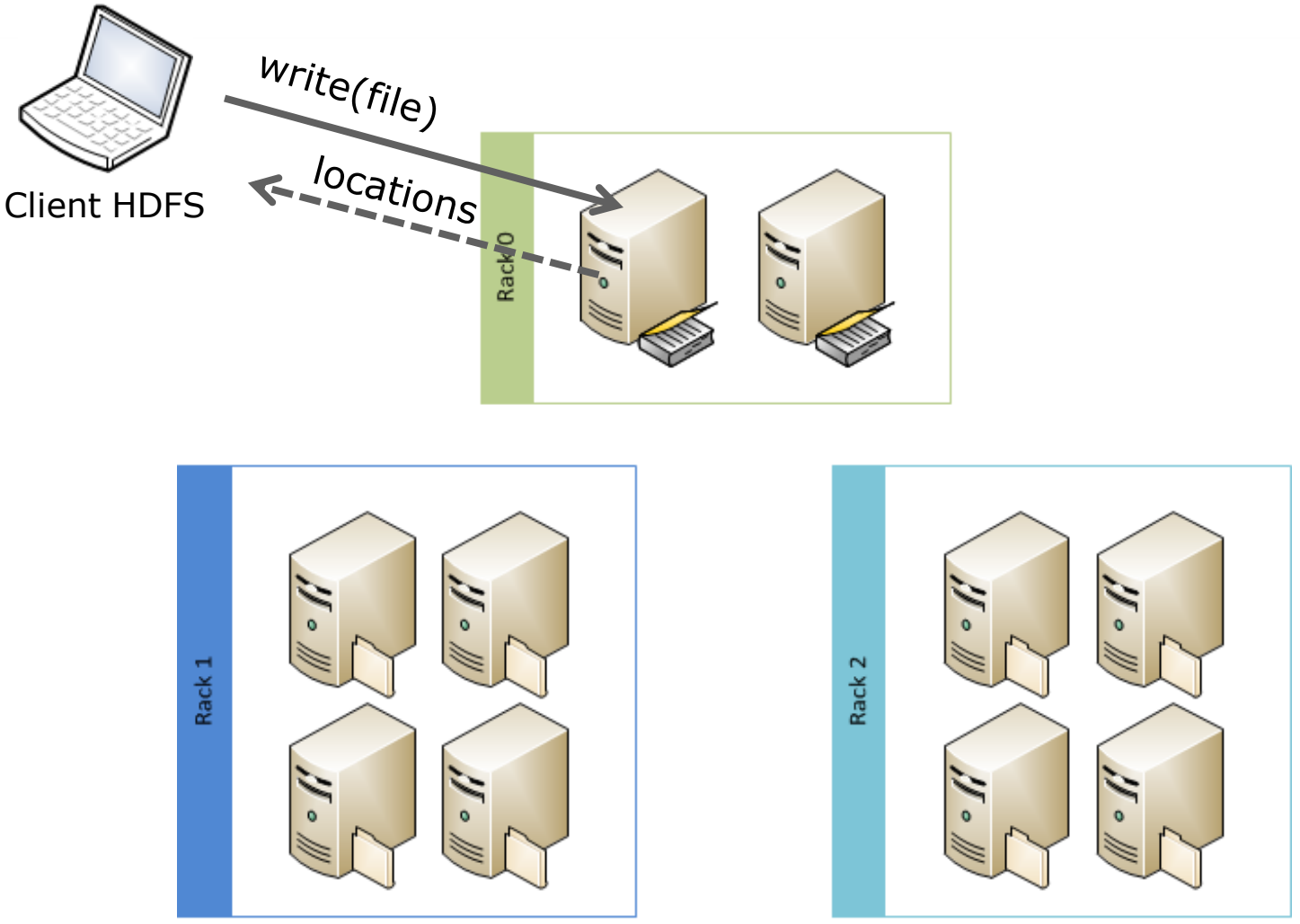
Datanode

- Stocke les blocks
- Sert les blocks
- Réalise la réplication

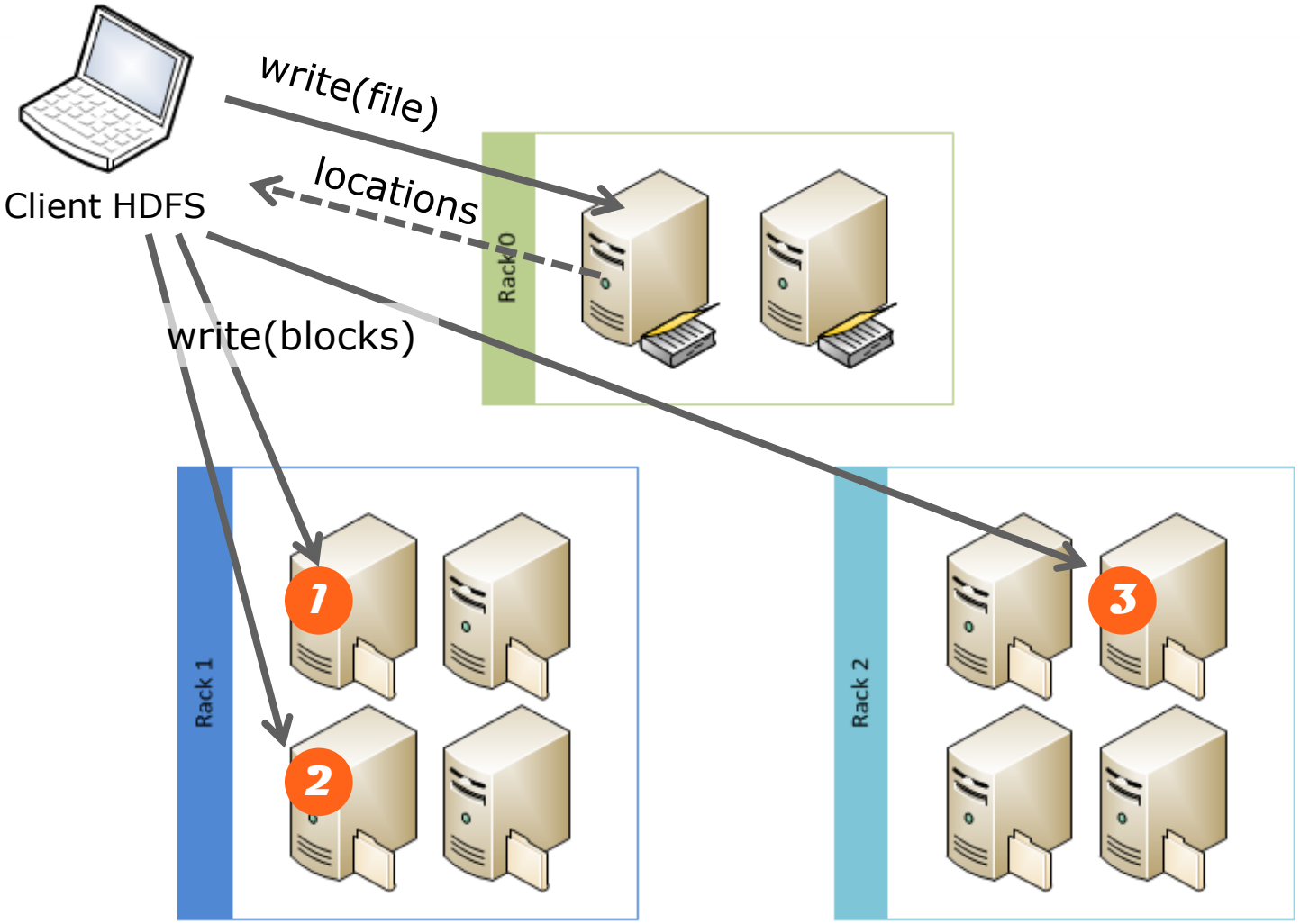




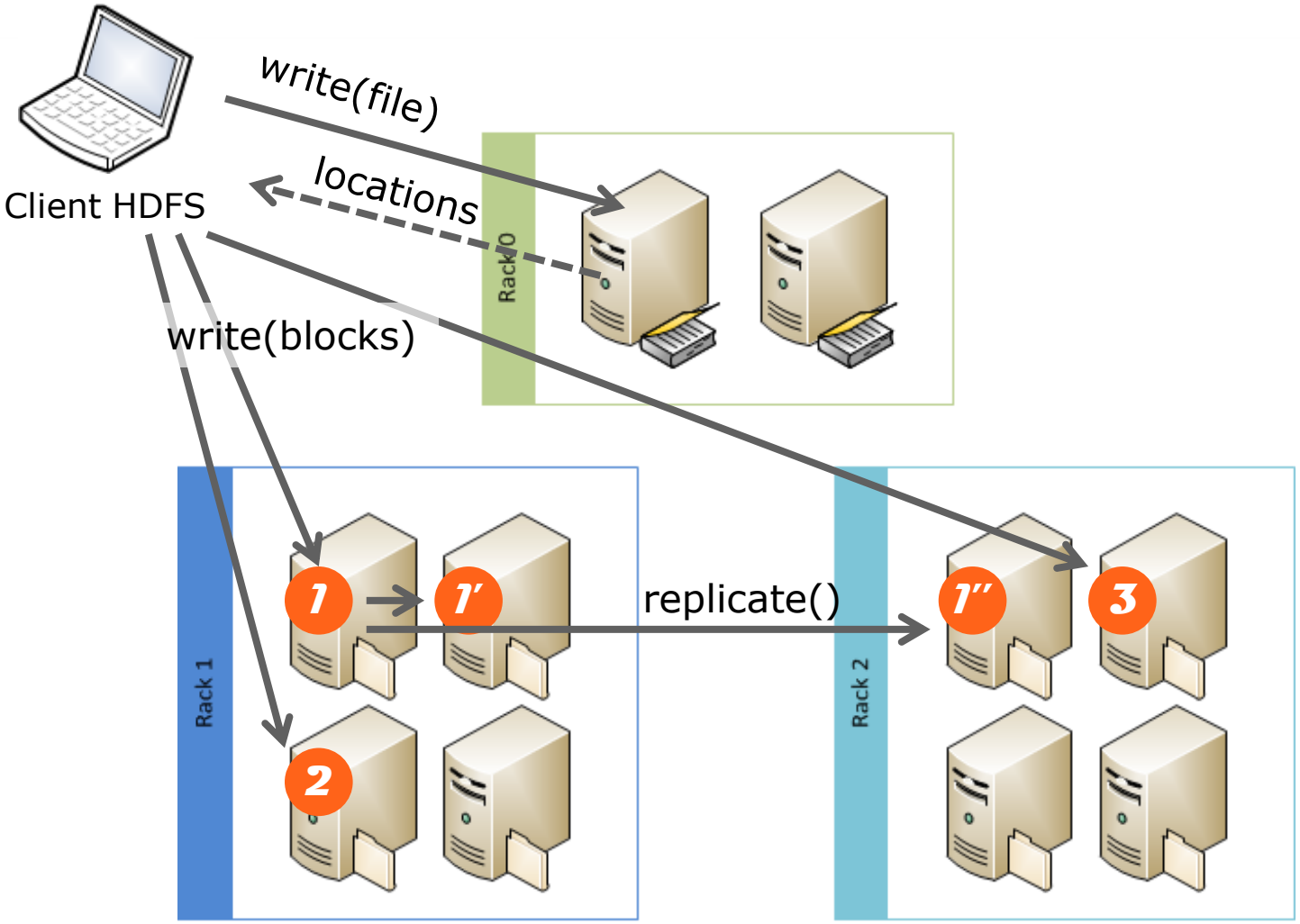
Usage d'HDFS



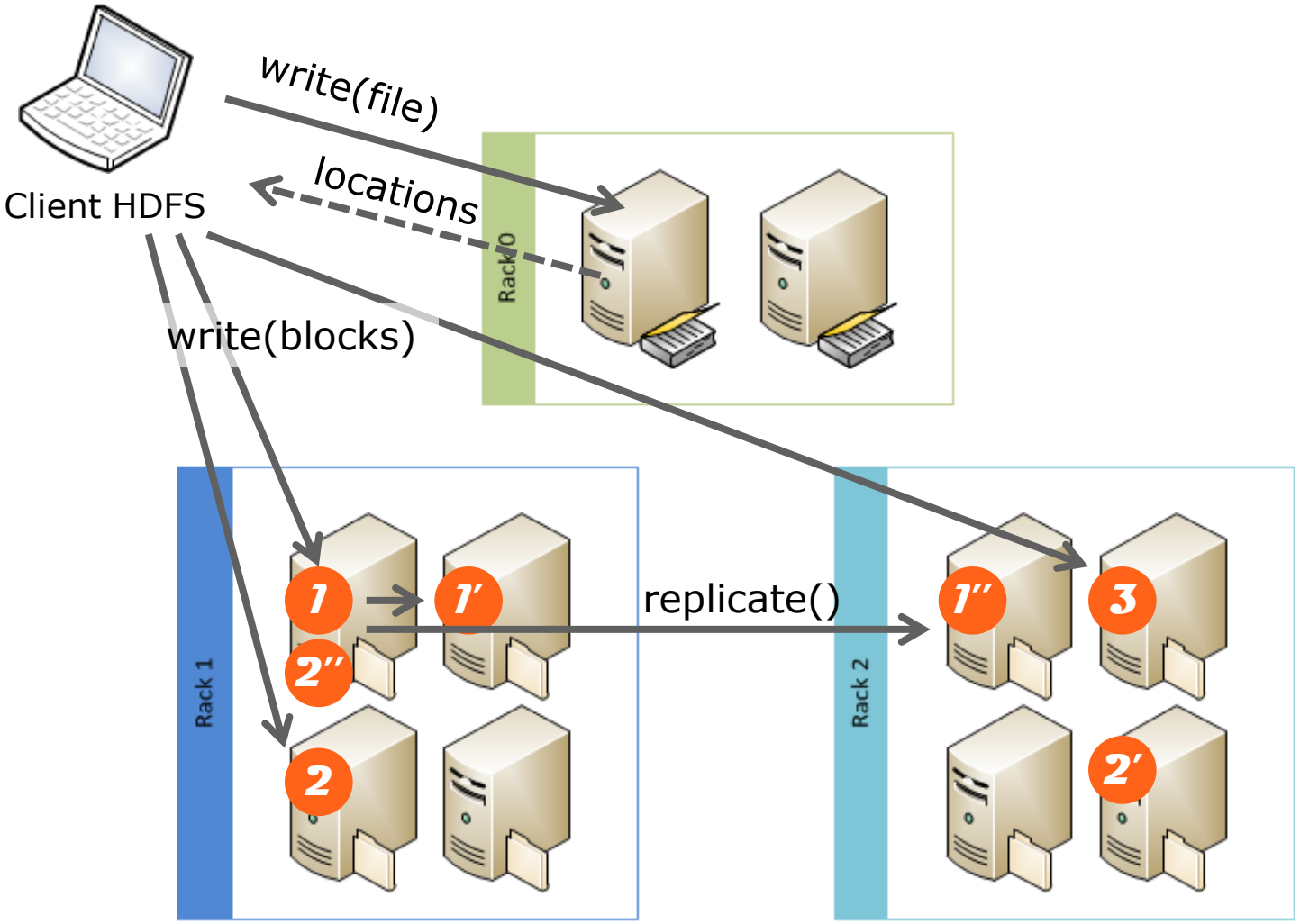
Usage d'HDFS



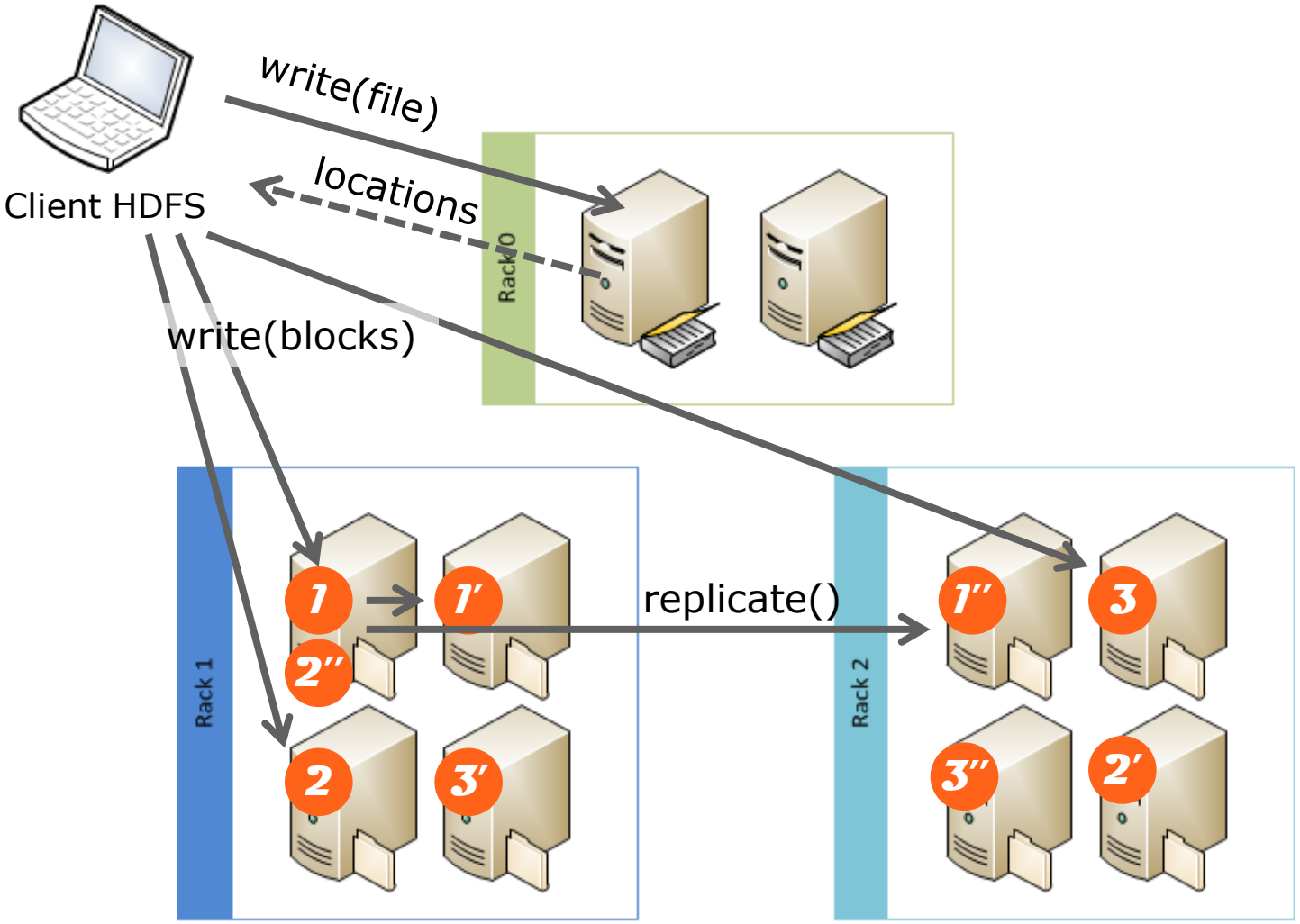
Usage d'HDFS



Usage d'HDFS



Usage d'HDFS



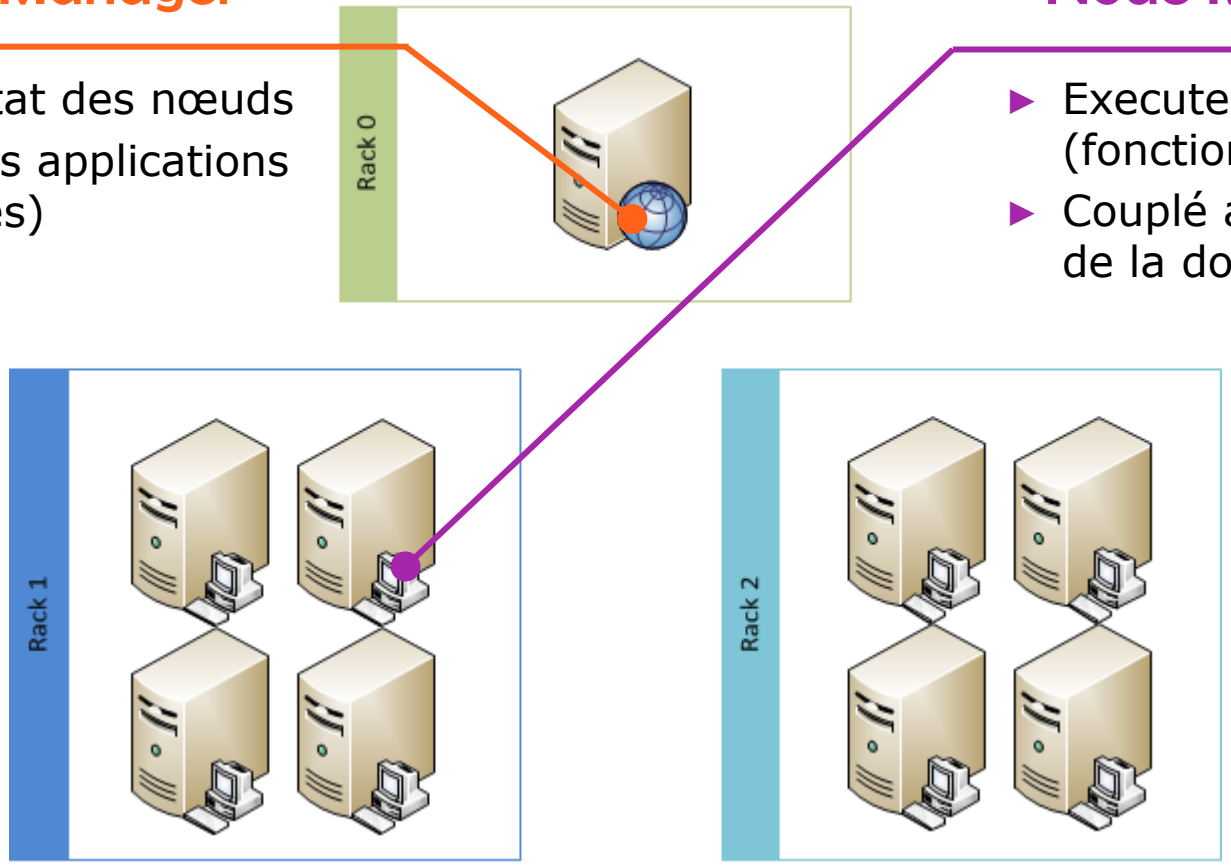
Organisation des exécutables YARN

Resource Manager

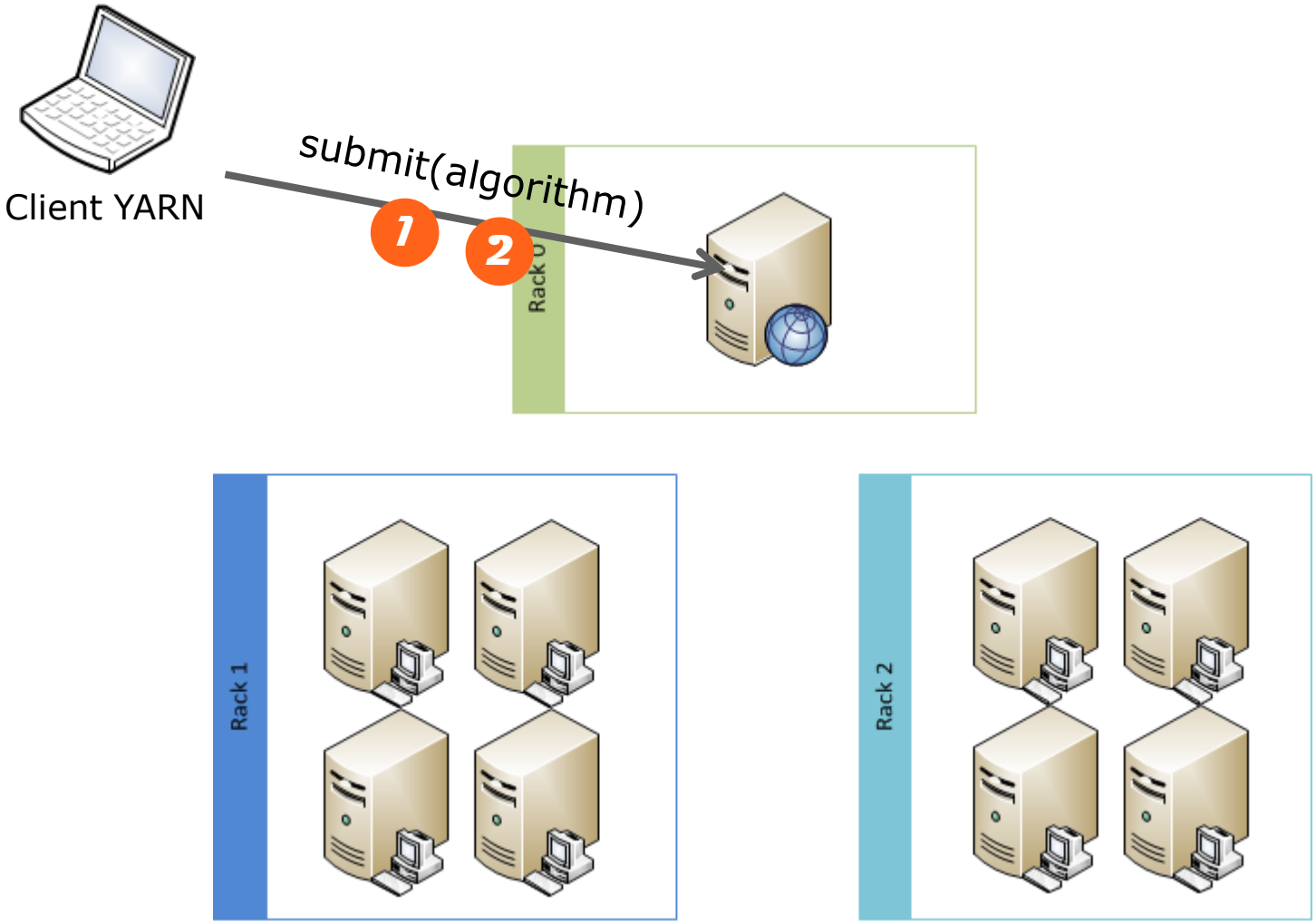
- Collecte l'état des nœuds
- Distribue les applications (algorithmes)

Node Manager

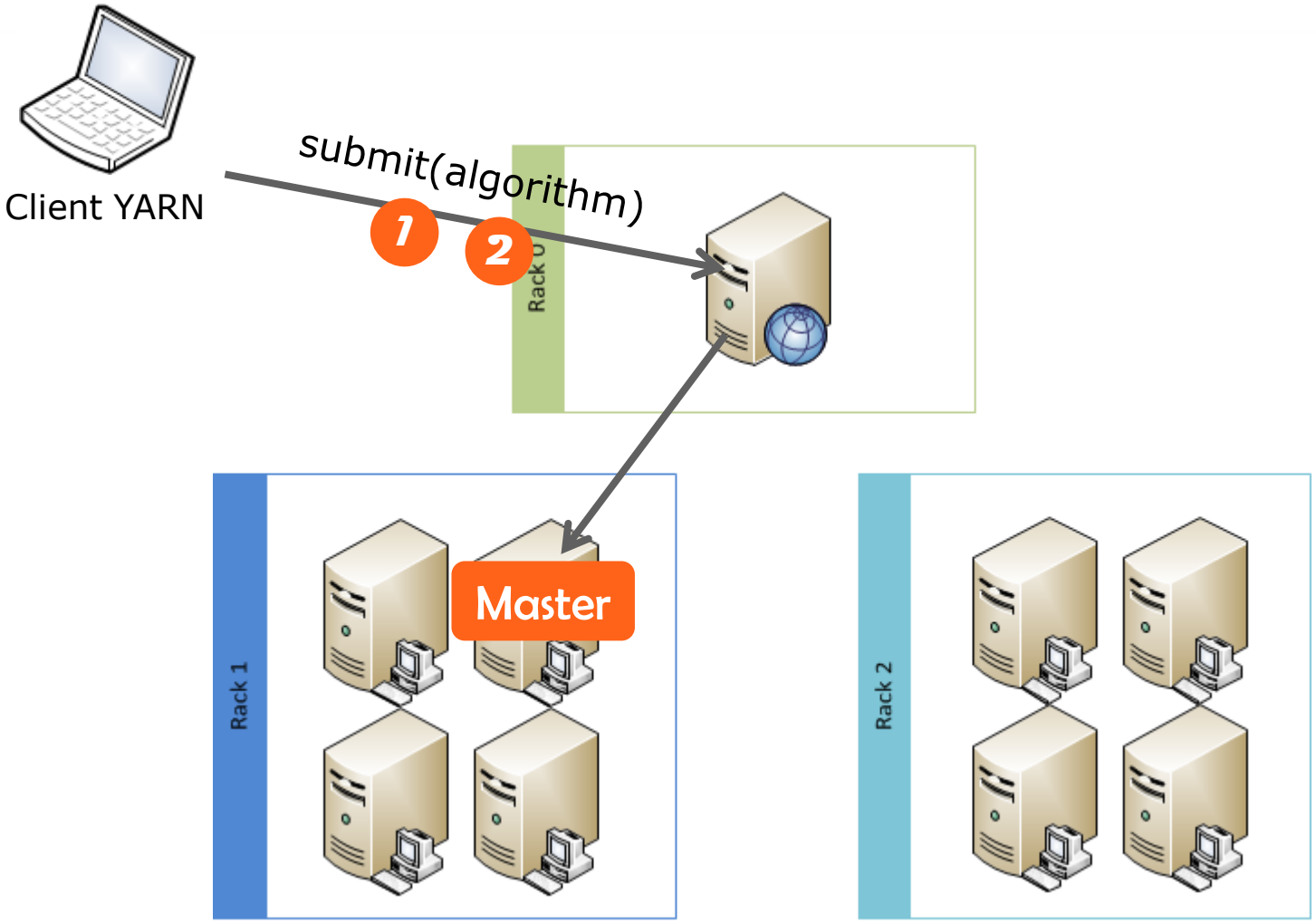
- Execute les tâches (fonctions)
- Couplé au stockage de la donnée



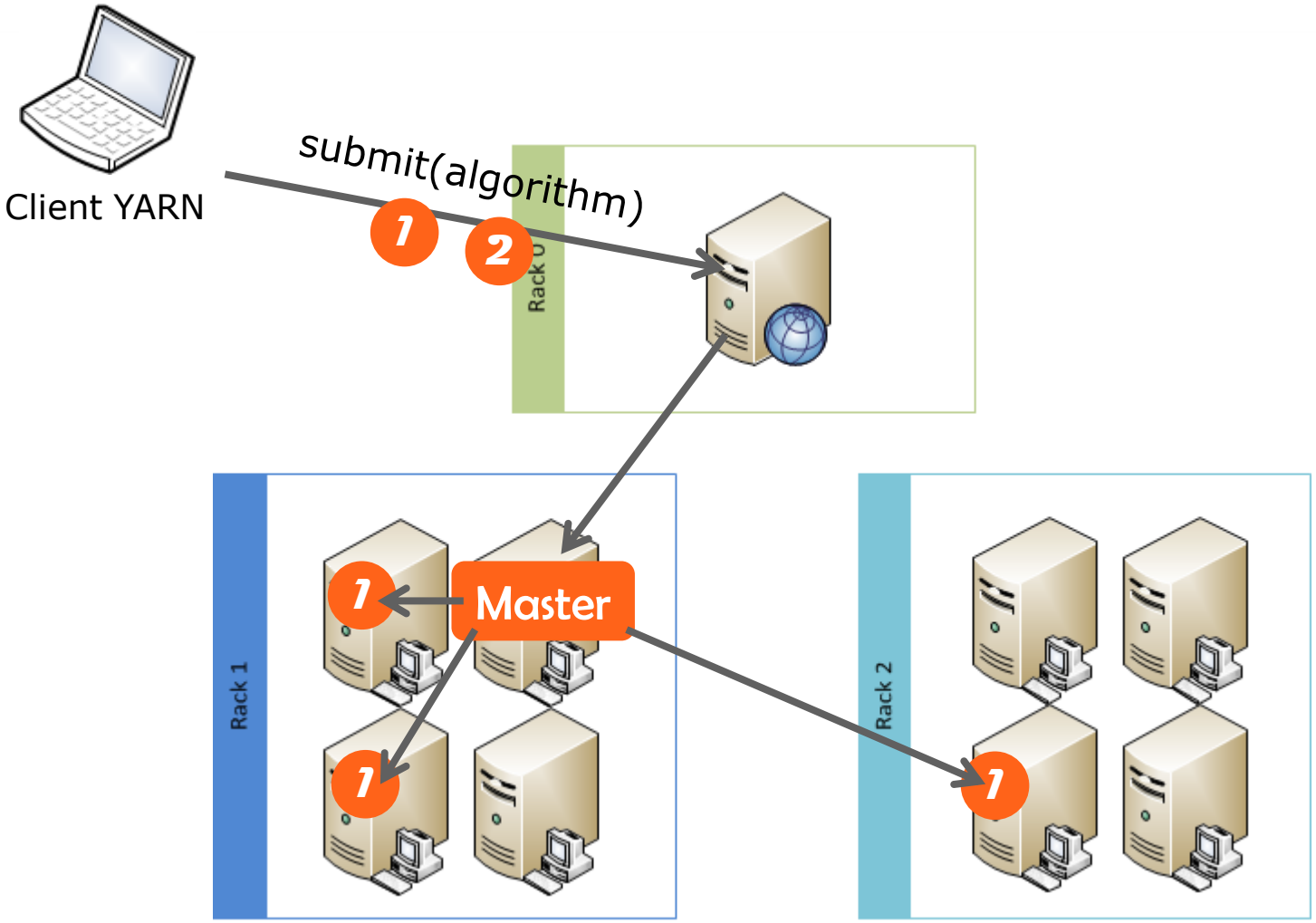
Usage de YARN



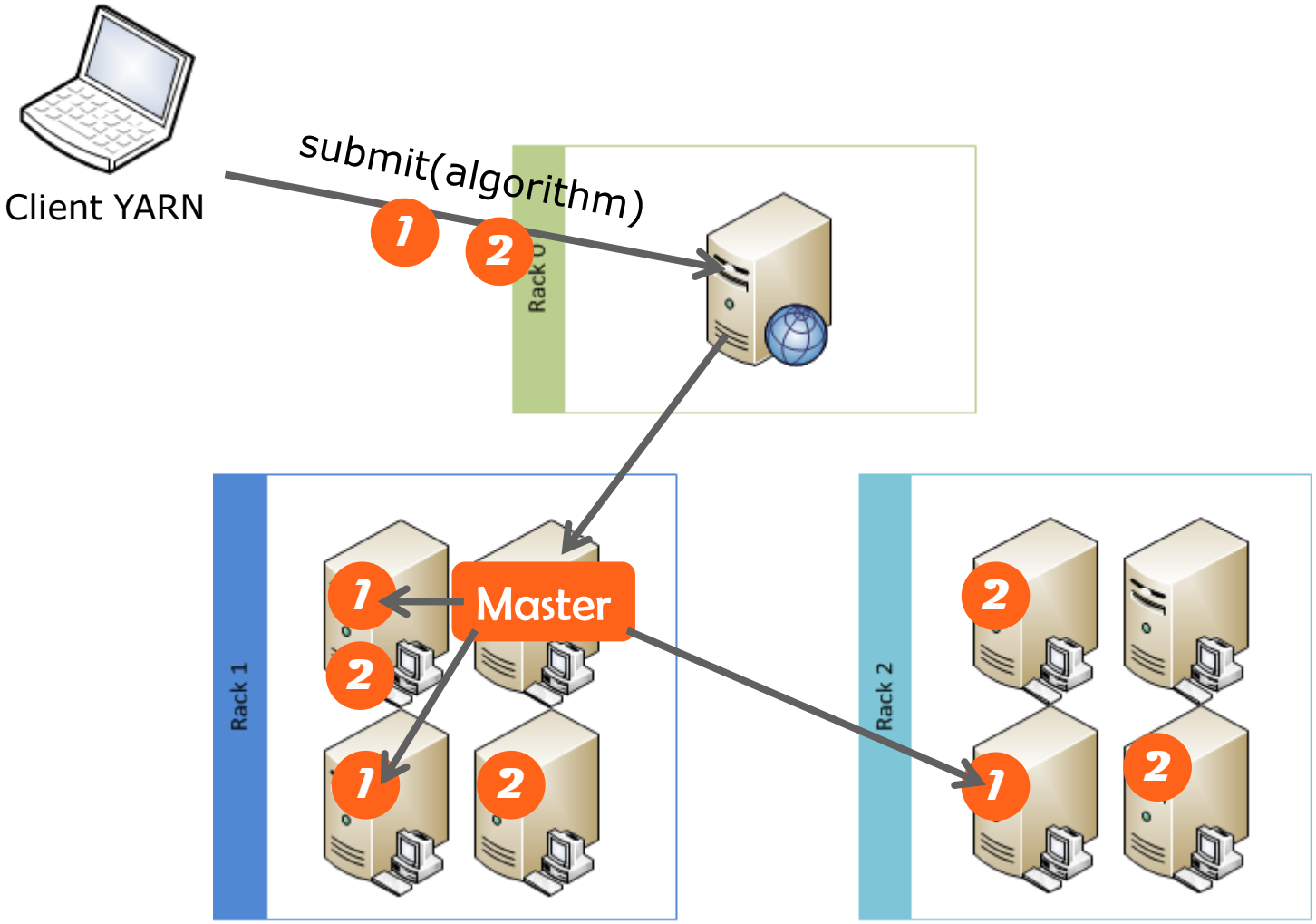
Usage de YARN



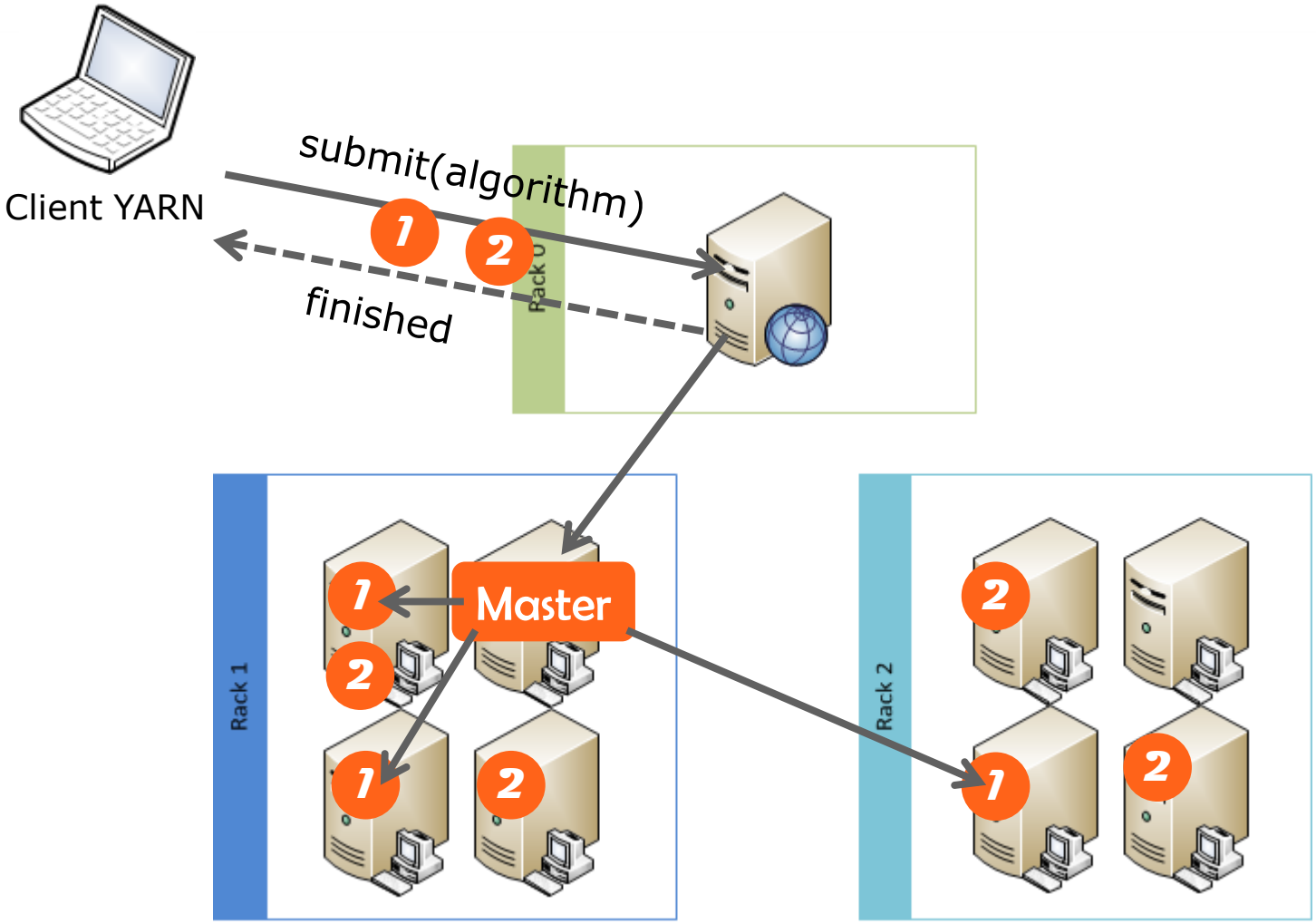
Usage de YARN



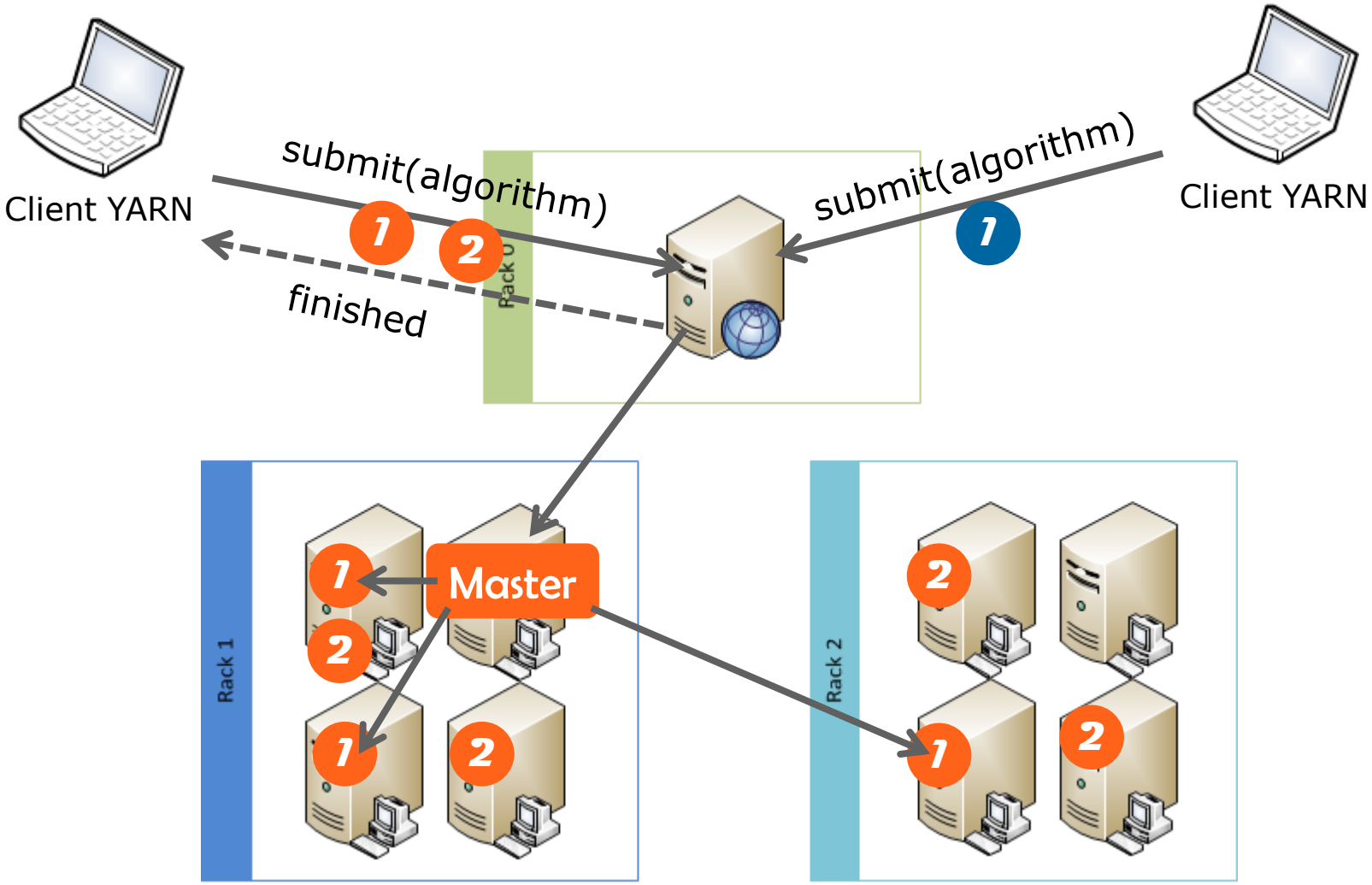
Usage de YARN



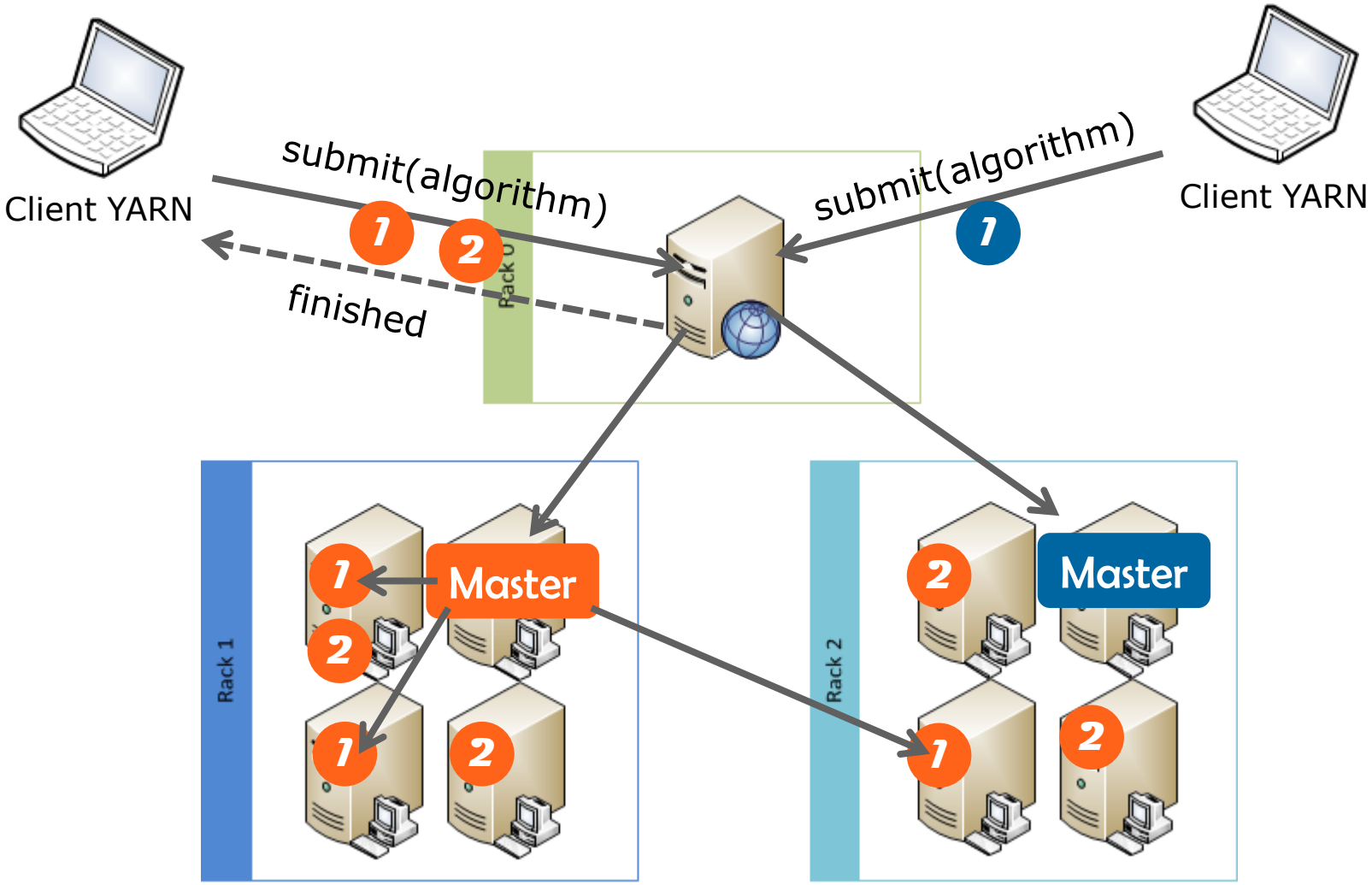
Usage de YARN



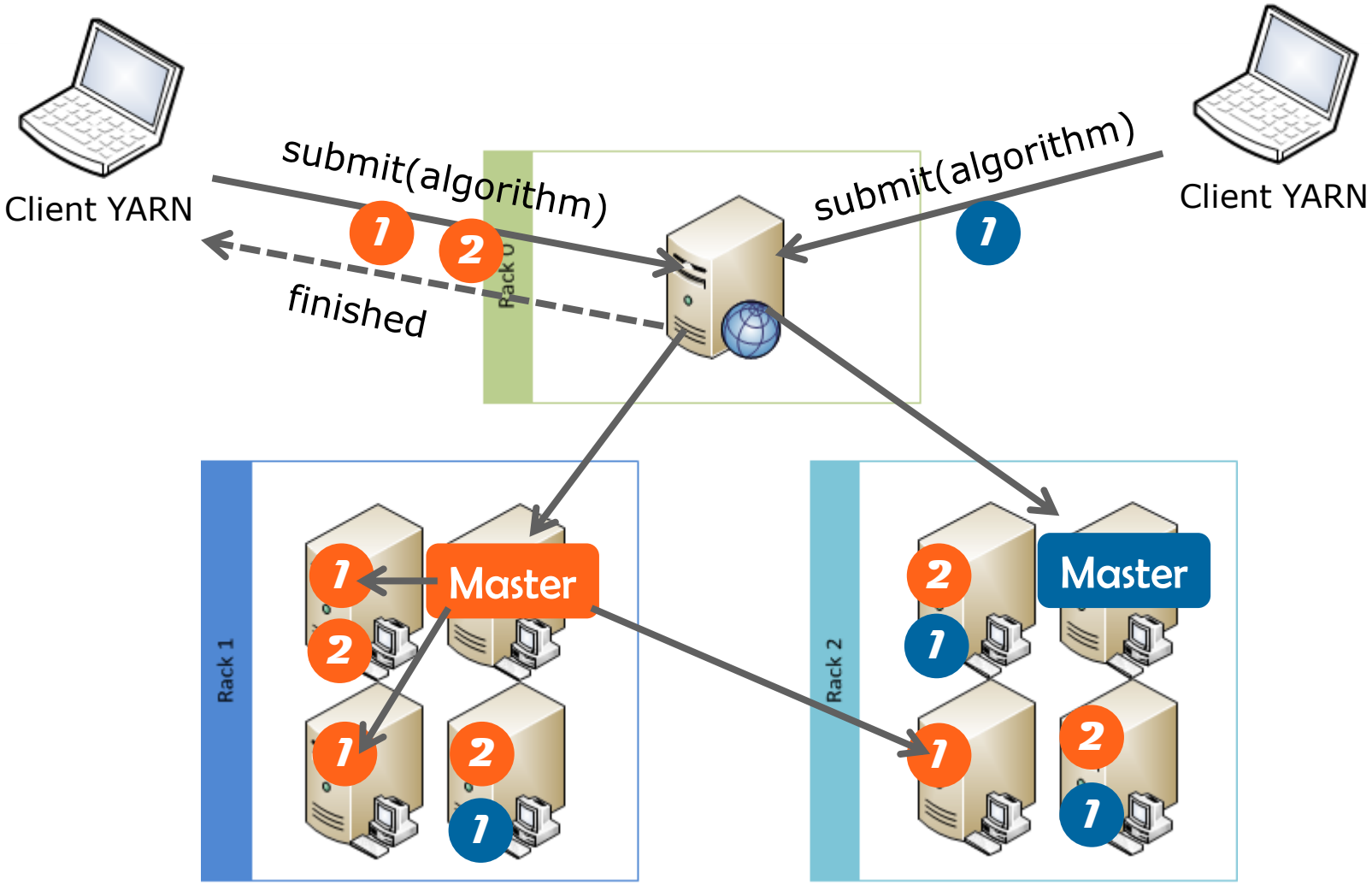
Usage de YARN



Usage de YARN



Usage de YARN



MapReduce en Java (new API)

Interfaces et classes de base

- InputFormat : utilisé par le système lors de la lecture du fichier d'entrée.
- Mapper : implémentation de la fonction map. Récupère la sortie de l'InputSplit, émet des couples clé-valeur pour les reducers
- Reducer : implémentation de la fonction reduce. Récupère l'agrégation des valeurs d'une clé donnée, retourne des couple clé-valeur (résultat)
- OutputFormat : utilisé par le système pour écrire les résultats des reducers.
- Job : classe de configuration du système (indique les formats d'entrée et de sortie, le Mapper et le Reducer…)

Exécution au sein du cluster

- Nécessite Java 1.6+
- Utilise des fichiers de configuration pour connaitre la topologie (Resource Manager)

MapReduce en Java (new API)

Mapper

```java
/**
 * InputKeyClass est la classe de la clé d'entrée, InputClass celle de la valeur d'entrée.
 * OuputKeyClass est la classe des clés émises, OutputClass celle des valeurs émises.
 */
public class MyMapper extends Mapper<InputKeyClass, InputClass, OutputKeyClass, OuputClass> {

  /**
   * La fonction est appellée pour chaque "split" (input) du fichier d'entrée.
   * Par exemple, inputKey pourrait être le numéro de ligne, et input la ligne elle même
   * Le contexte permet d'emettre les couples clé/valeur.
   * Dans l'exemple du "word count", la clé émise est un String (le mot) et la valeur est un
   * entier (le nombre d'occurence dans la ligne).
   */
  @Override
  protected void map(InputKeyClass inputKey, InputClass input, Context context) {
    // ...manipulation de l'entrée
    // emission d'un clé et de la valeur associée (peut être fait plusieurs fois)
    context.write(key, value);
  }
}
```

MapReduce en Java (new API)

Reducer

```java
/**
 * InputKeyClass est identique à OutputKeyClass du mapper, InputClass est identique à
 * OutputClass du mapper.
 * OuputKeyClass est la classe des clés émises, OutputClass celle des valeurs émises.
 */
public class MyReducer extends Reducer<InputKeyClass, InputClass, OutputKeyClass, OuputClass> {

  /**
   * La fonction est appellée pour chaque aggrégat de valeur (values) d'une clé possible (key).
   * Le contexte permet d'emettre les résultats.
   * Dans l'exemple du "word count", la clé résultat est un String (le mot) et la valeur est un
   * entier (la somme des occurences).
   */
  @Override
  protected void reduce(InputKeyClass key, Iterable<InputClass> values, Context context) {
    // ...manipulation des valeurs et de la clé
    // emission d'un clé et de la valeur associée (peut être fait plusieurs fois)
    context.write(key, value);
  }
}
```

MapReduce en Java (new API)

Déclaration et lancement de l'application

```java
// Point d'entrée
public class MyJob extends Configured implements Tools {

  @Override
  public int run(String[] args) throws Exception {
    // cré une application MapReduce avec le nom de votre choix
    Job myJob = Job.getInstance(getConf(), "MyJob");
    // indique le jar qui contient le Mapper et le Reducer, et leur classes
    job.setJarByClass(getClass());
    job.setMapperClass(MyMapper.class);
    job.setReducerClass(MyReducer.class);

    // Fichier(s) d'entrée
    TextInputFormat.addInputPath(job, new Path("/input.txt"));
    job.setInputFormatClass(TextInputFormat.class);

    // Fichier(s) de sortie et classe des clé/valeur résultats (voir le Reducer)
    TextOutputFormat.addOutputPath(job, new Path("/output"));
    job.setOuputFormatClass(TextOutputFormat.class);
    job.setOuputKeyClass(OuputKeyClass.class);
    job.setOuputValueClass(OuputClass.class);

    // Execution bloquante de l'application
    return job.waitForCompletion(true) ? 0 : 1;
  }

  // méthode main exécutable
  public static void main(String[] args) throw Exception {
    System.exit(ToolRunner.run(new MyJob(), args));
  }
}
```

Au dela du MapReduce

La galaxie Hadoop regroupe de nombreux produits Java

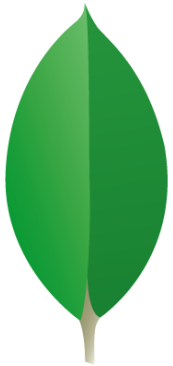
- Un SGBD NoSQL Colonne (Hbase)
- Un SGBD SQL (Hive)
- Un framework de machine learning (Mahout)
- Un framework de distribution de calcul (Spark)
- Un DSL pour combiner les MapReduce (Pig)
- Un outil de coordination (ZooKeeper)
- Un outil de supervision et provisionning (Ambari)
- Et bien d'autres encore…


mongoDB

Un système orienté documents

huMongous: gigantesque

- Ecrit en C++
- Interface en JSON, stockage en BSON
- Langage de requête
- Sharding automatique
- Exécution d'algorithme Map/Reduce



#1

Documents

Production-ready

- 10Gen commence en 2009, Open source mars 2010
- Focus sur la consistance des données
- Nombreux déploiements en production (SAP, Sourceforge, Foursquare, eBay)

Sharding et réplication avec MongoDB

Réplication des documents

- Au niveau d'une collection de documents
- Entre les instances MongoD du cluster
- Une instance primaire (la seule en écriture)
- Une ou plusieurs instances secondaires, éligibles en cas de perte du primaire

  ⇨ Nécessité d'un nombre impaire (utilisation d'un arbitre)
- Les réplicas sont utilisables en lecture

  ⇨ Attention au problèmes de consistance

Sharding d'une collection

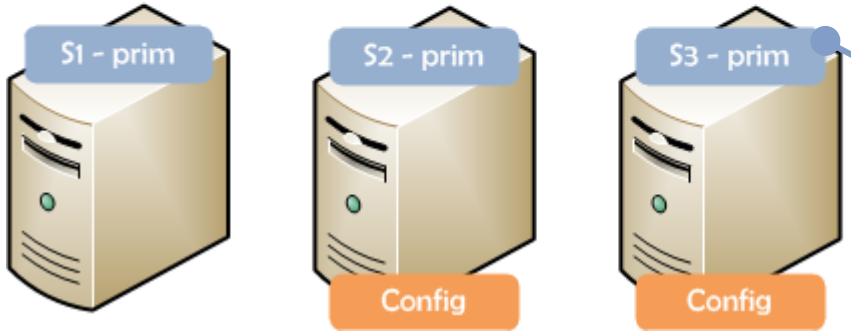
- Selon une clé définit par le développeur
- Nécessite un serveur de configuration (annuaire)

  ⇨ Sur une machine à part (SPOF), backupée par deux autres instances
- Permet de répartir les écritures et lectures sur plusieurs machines
- Chaque shard (MongoD) peut être ensuite répliqué (primaire)
- Rééquilibrage automatique par les serveurs de configuration

Topologie de production : instances

Configuration

- Annuaire des shards
- Balance automatique
- Au moins 1 sur une machine dédiée



Serveur Data

- Stocke les données
- Exécutes les traitements
- Machines puissantes

worldline
e-payment services

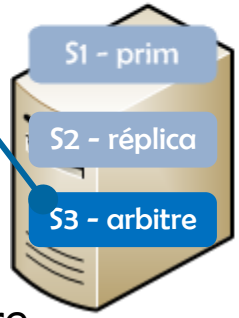
Topologie de production : instances

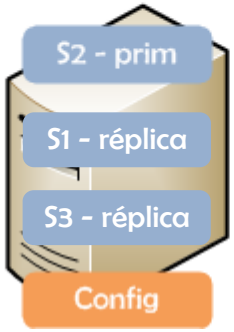
Configuration

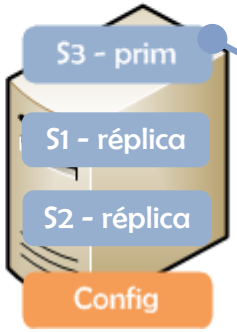
- Annuaire des shards
- Balance automatique
- Au moins 1 sur une machine dédiée

Arbitre

- Sans données
- Pour arbitrer l'élection du nouveau primaire







Serveur Data

- Stocke les données
- Exécutes les traitements
- Machines puissantes

worldline
e-payment services

Topologie de production : instances

Application utilisatrice

- C, Java, Ruby, JS, Go, Php…
- Connectée à plusieurs routeurs

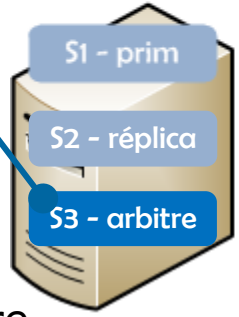
Routage

- Connectés aux configuration
- Au moins 2 pour la disponibilité
- Machines virtualisables

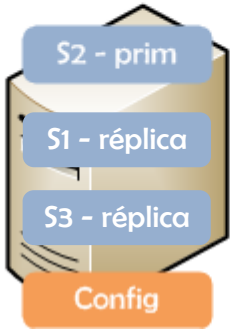
Configuration

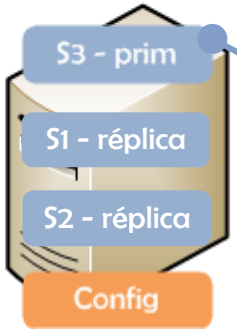
- Annuaire des shards
- Balance automatique
- Au moins 1 sur une machine dédiée

Arbitre



- Sans données
- Pour arbitrer l'élection du nouveau primaire

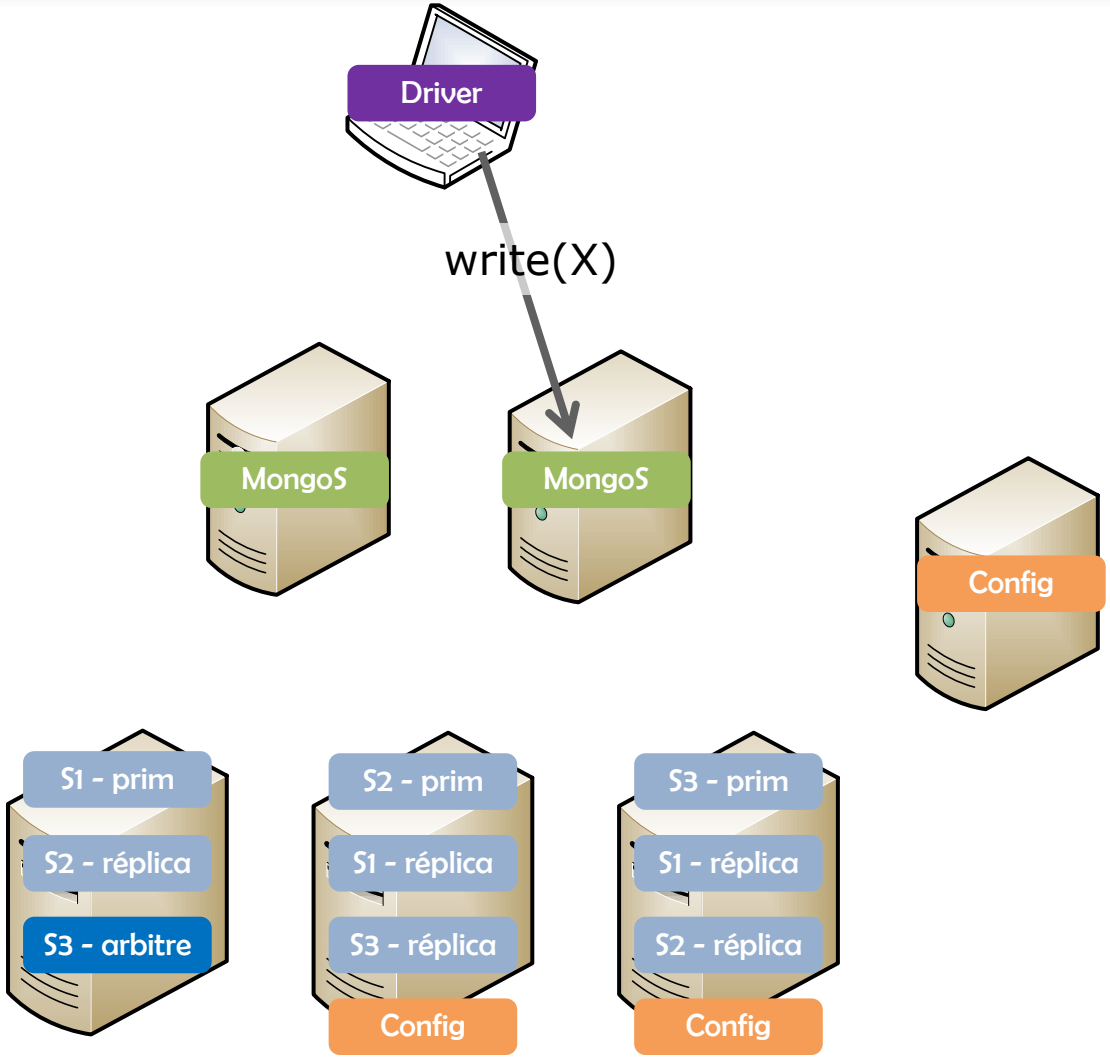




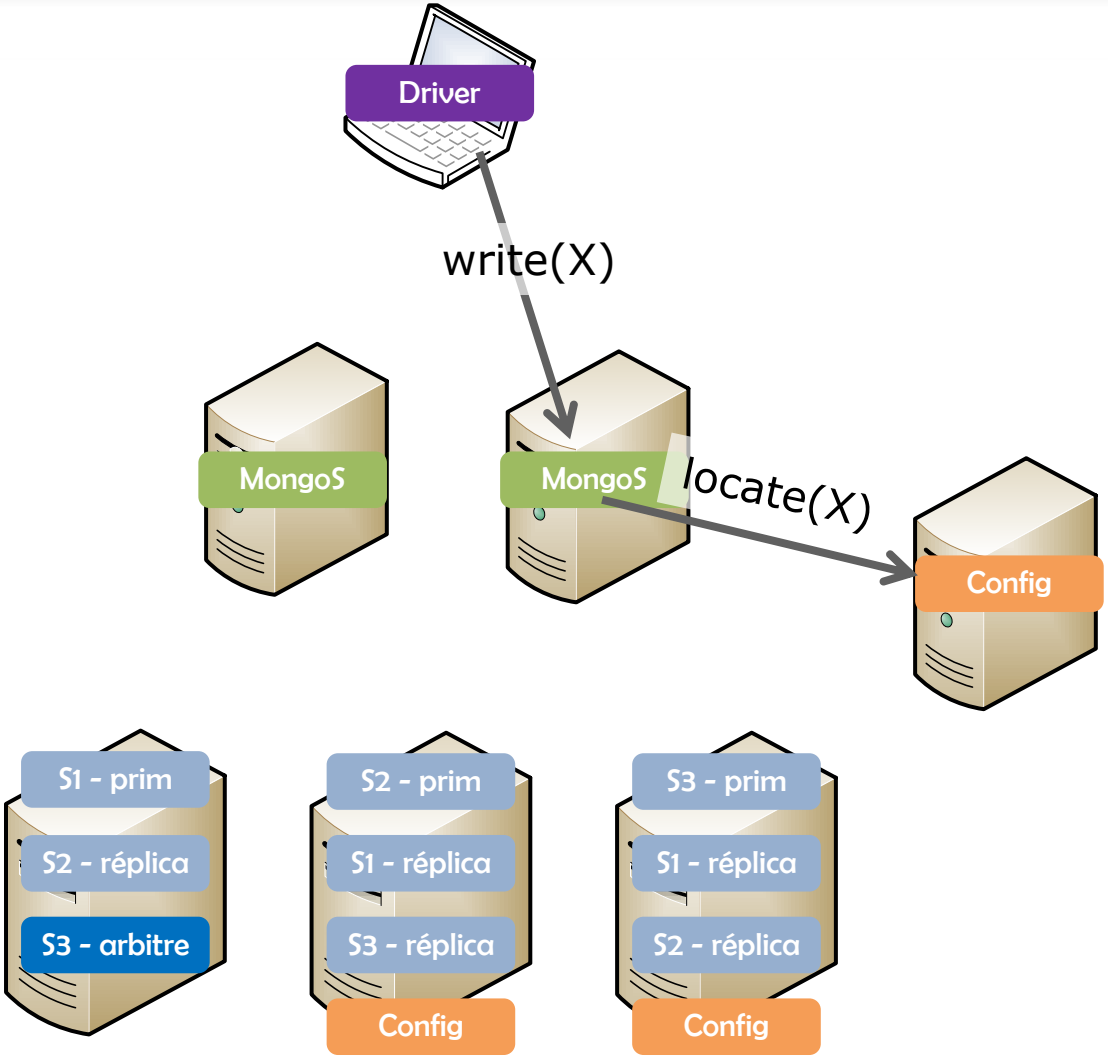
Serveur Data

- Stocke les données
- Exécutes les traitements
- Machines puissantes

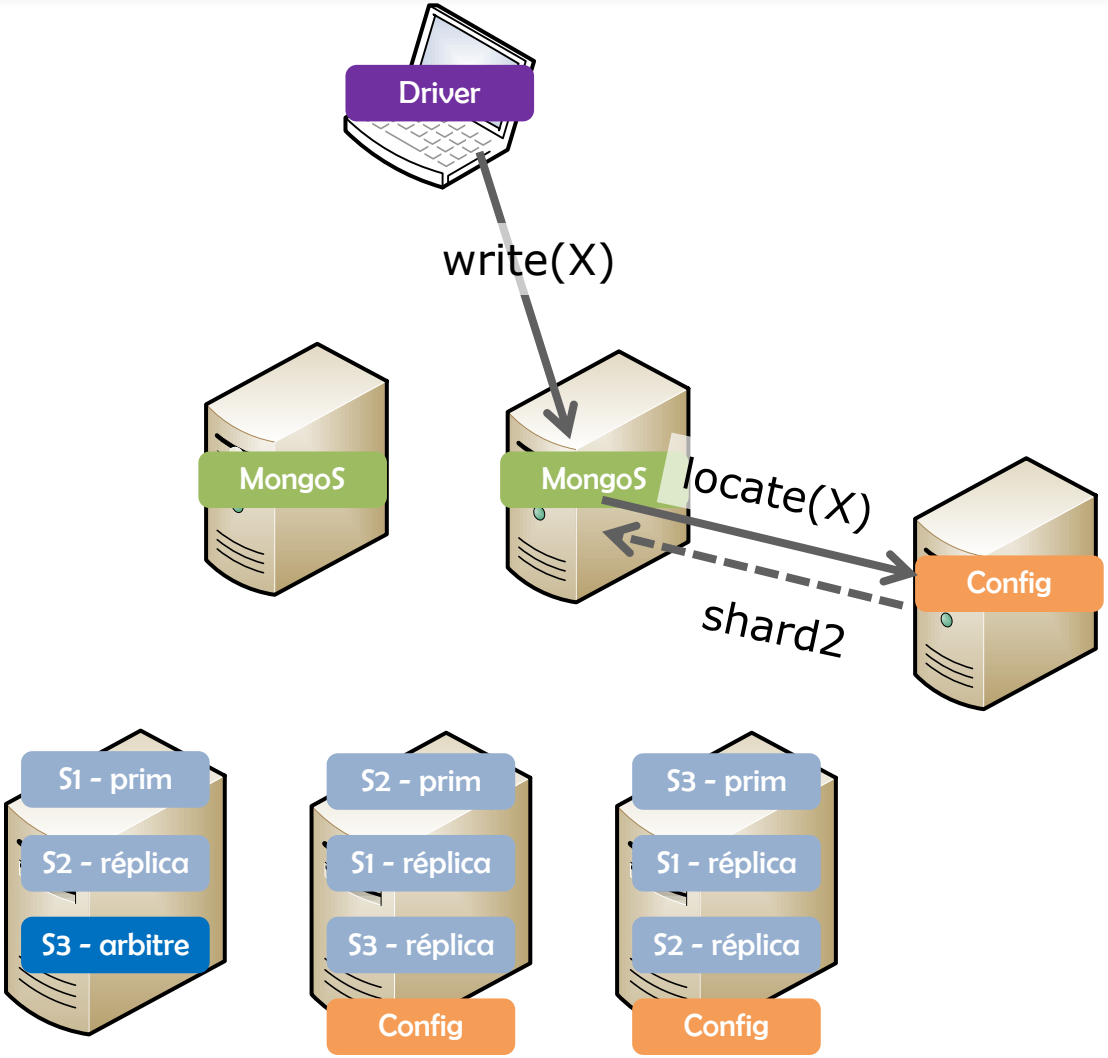
Topologie de production : flux



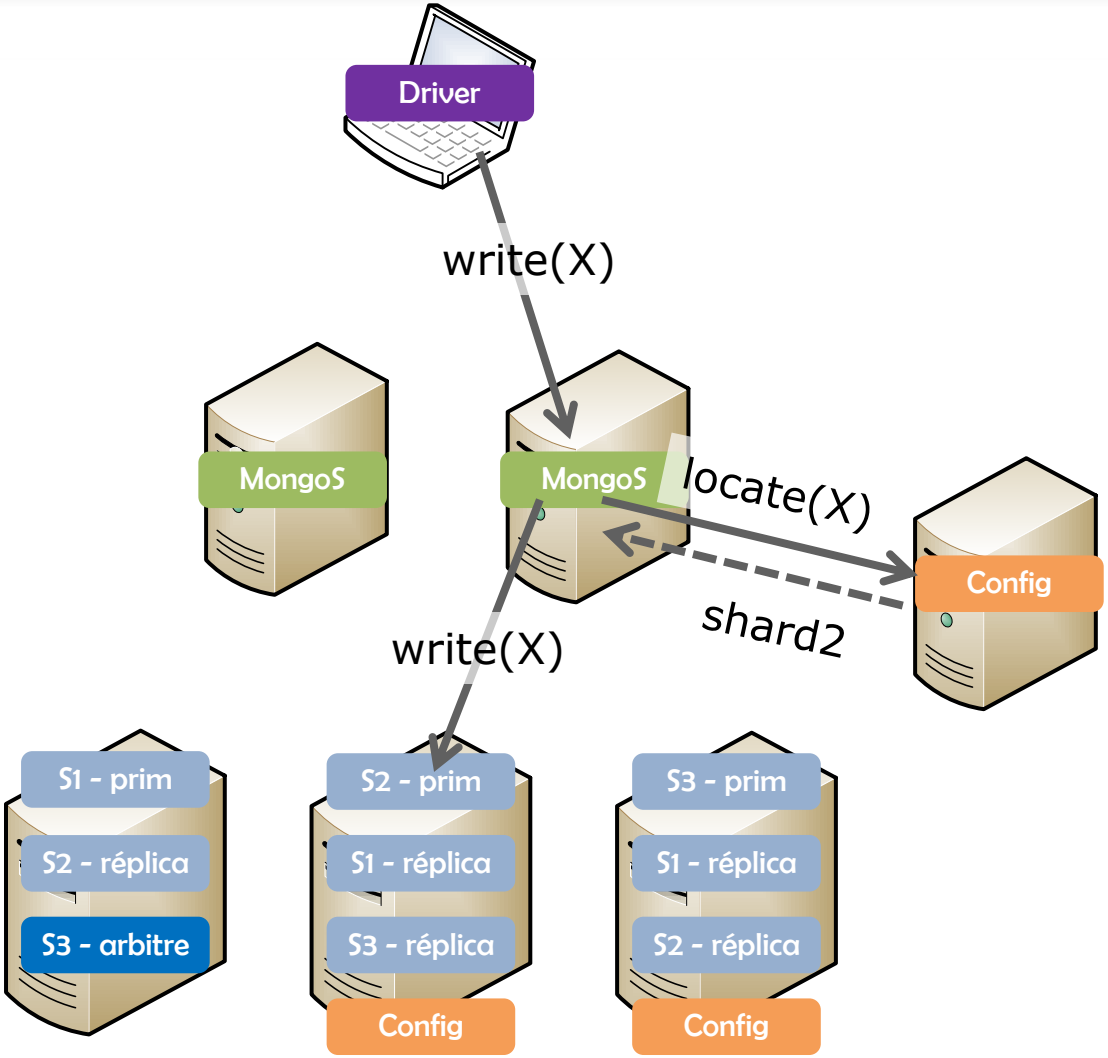
Topologie de production : flux



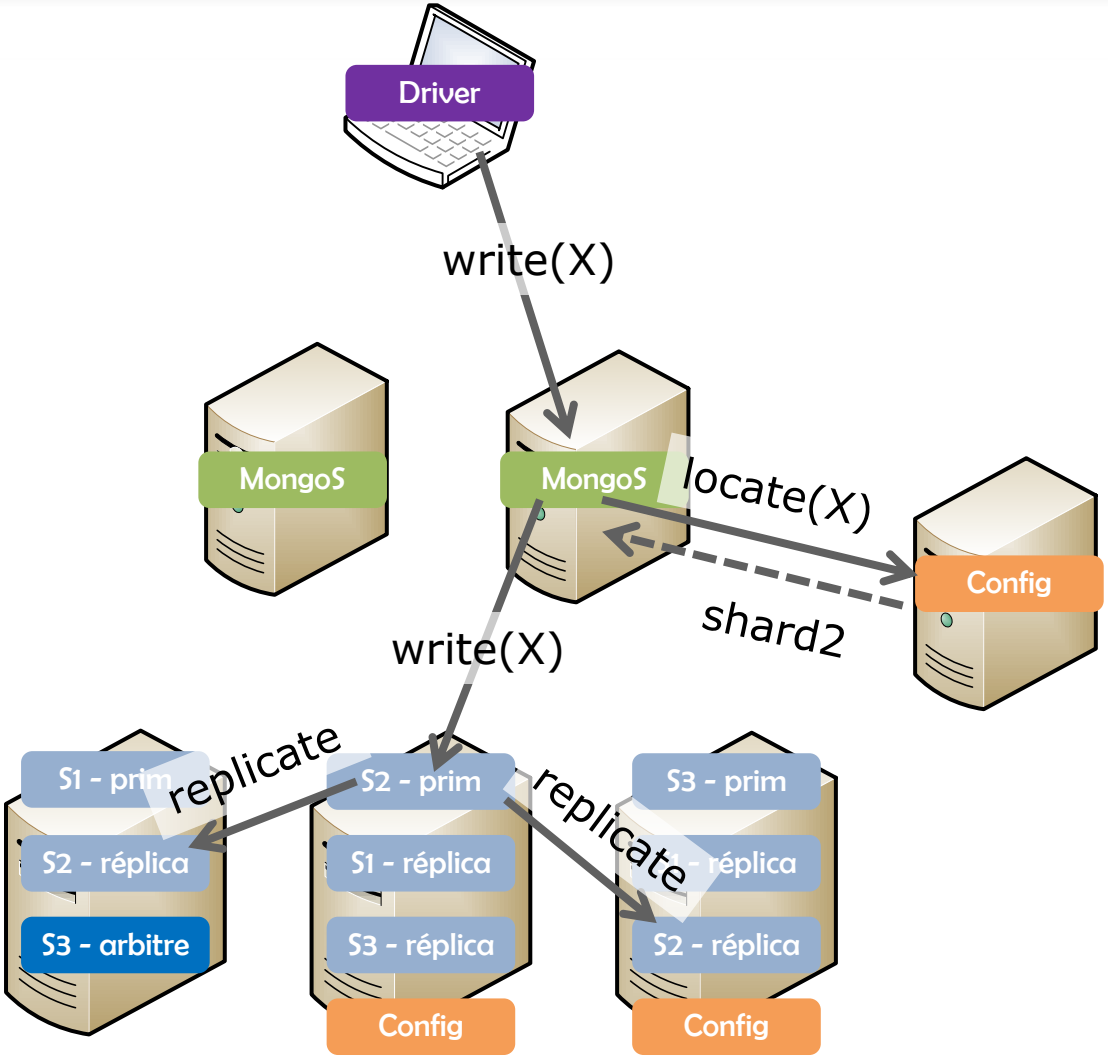
Topologie de production : flux



Topologie de production : flux



Topologie de production : flux



MapReduce en JavaScript (depuis la CLI)

Concepts de base

- Input: les documents **d'une collection** (requête optionnelle sans projection).
- Mapper : fonction qui traite un des document et émet 0 ou N paires clé/valeur
- Reducer : fonction qui récupère **une partie** de l'agrégation des valeurs d'une clé donnée, et retourne **1 valeur pour cette clé**
- Finalizer : fonction optionnelle qui traite le résultat d'une clé possible (après les reducers sur cette clé)
- Output : une collection, ou la CLI directement (inline)

Particularités

- Les valeurs émises par le mapper et retournées par le reducer doivent avoir le même type
- Le reducer doit être **associatif** et **idempotent**

```
reduce(key, [C, reduce(key, [A, B])]) == reduce(key, [C, A, B])

reduce(key, [reduce(key, valuesArray)]) == reduce(key, valuesArray)
```

- Pas d'invocation d'un reducer sur une clé n'ayant qu'une valeur

MapReduce en JavaScript

Mapper

```
// Execution d'un MapReduce depuis la CLI sur la collection 'input_collection'
db.input_collection.mapReduce(

  /**
   * A l'intérieur du mapper, le context (this) est le document traité
   * La fonction globale 'emit' permet d'emettre, 0, 1 ou plusieurs couple clé/valeur
   */
  function mapper() {
    // ... manipulation du document d'entrée (this)
    // emission d'une paire clé valeur (peut être appellée plusieurs fois)
    emit(key, value);
  },
```

MapReduce en JavaScript

Reducer

```
/**
 * Le reducer prend en entrée une clé et l'aggrégation (partielle) de ses valeurs
 * Il retourne une valeur pour cette clé (même type que les éléments dans 'values')
 */
function reducer(key, values) {
  // ... manipulation associative et indempotente des valeurs
  // renvoi d'un résultat pour cette clé
  return result;

},
```

MapReduce en JavaScript

Paramétrage et finalizer

```
{
  // sauvegarde des résultats dans une collection, ou affichage dans la console
  out: 'output_collection',
  out: {inline: 1},

  // (optionnels) requête, tri et limitation des documents de input_collection
  query: {qty: {$gt: 500}},
  sort: {date: 1},
  limit: 100000,

  /**
   * Le finalizer (optionnel) reçoit le résultat émis par un (ou plusieurs)
   * reducer pour une clé.
   * Il renvoi un résultat éventuellement modifié pour sauvegarde.
   */
  finalize: function(key, value) {
    // ...manipulation éventuelle de value
    return value;
  }
}
);
```

Un brouillon pour l'Aggregation Framework

Créé pour combler un manque

- MongoDB ne proposait pas de fonctions d'agrégation en 2010
- MapReduce avait particulièrement le vent en poupe
- Réalisé a partir de la VM Javascript (V8), et non en C++

The Aggregation Framework

- Introduit depuis la version 2.2 en 2012
- Pipeline de transformation appliqué sur les résultats d'une requête

⇨ Filtrage, transformation, regroupement, tri…

- Exploite entièrement le moteur de manipulation C++
- Plus naturel d'utilisation pour des développeurs avec un background SQL

Une étude de cas

Différents cas d'usage

Logs web de site grand public

- Entrée : horodatage + id session + article consulté + browser
- Plusieurs milliers par minutes
- Indicateurs calculés la nuit sur les logs de la journée
  - Classement des articles consultés
  - Durée moyenne de session sur la journée d'hier

Recommandation d'achat

- Entrée : panier avec la liste des produits achetés
- Historiques de millions de commandes
- Etablir la liste exhaustive des produits achetés en même temps qu'un autre, en tenant compte de la fréquence d'apparition

Identification de foyer

- Entrée : civilité + adresse postale + email + téléphone
- Donnée postale très hétérogènes
- Identifier les personnes vivant dans le même foyer (adresse postale similaire, dédoublonnage à partir de l'email, téléphone similaire)

Durée moyenne de session (1)

Evaluer la durée des sessions

- A partir de la collection logs qui contient un document par log
- En stockant le résultat dans une collection intermédiaire

```
// un document représente un log
// ce premier map reduce evalue la durée de session
db.logs.mapReduce(

  function mapper() {
    // pour cette session, emet un intervalle avec l'horodatage du log
    emit(this.sessionId, {start: this.timestamp, end: this.timestamp});
  },

  function reducer(sessionId, intervals) {
    var total = {start: Infinity, end: 0};
    // évalue l'intervale englobant tous les intervalle connus pour cette session
    intervals.forEach(function(interval) {
      total.start = total.start > interval.start ? interval.start : total.start;
      total.end = total.end < interval.end ? interval.end : total.total;
    });
    return total;
  },

  {
    out: 'sessions',
    // uniquement les logs d'hier
    query: {timestamp: {$gte: Date.now()-3600*24}}
  }
);
```

Durée moyenne de session (2)

Calculer la moyenne

- A partir de la collection intermédiaire
- En affichant le résultat dans la CLI

```
// un document représente une session
// ce second map reduce evalue la durée moyenne
db.sessions.mapReduce(

  function mapper() {
    // regroupe tous les session en emettant une clé constante
    emit('avg', {
      sum: this.value.end.getTime()-this.start.getTime(),
      total: 1
    });
  },

  function reducer(constant, durations) {
    var result = {
      sum: 0,
      total: 0
    };
    // réalise la somme, mais pas la moyenne, car ce n'est pas
    // indempotent
    durations.forEach(function(duration) {
      result.sum += duration.sum;
      result.total += duration.total;
    });
    return result;
  },

  {
    // affichage dans la console
    out: {inline: true},
    // uniquement les logs d'hier
    finalize: function(constant, result) {
      // réalise la moyenne uniquement dans le finalizer, en milliseconds
      return {avg: result.sum/result.total};
    }
  }
);
```

Merci de votre attention !

Tous les logos sont la propriété de leurs compagnies respectives

Diapositive 1, @ Nicolas Rouyer

Diapositive 4, © TechWorld

Diapositive 8, © YarnWithATwist

Diapositive 20, © Digidom

Diapositive 30, © CloudTimes
