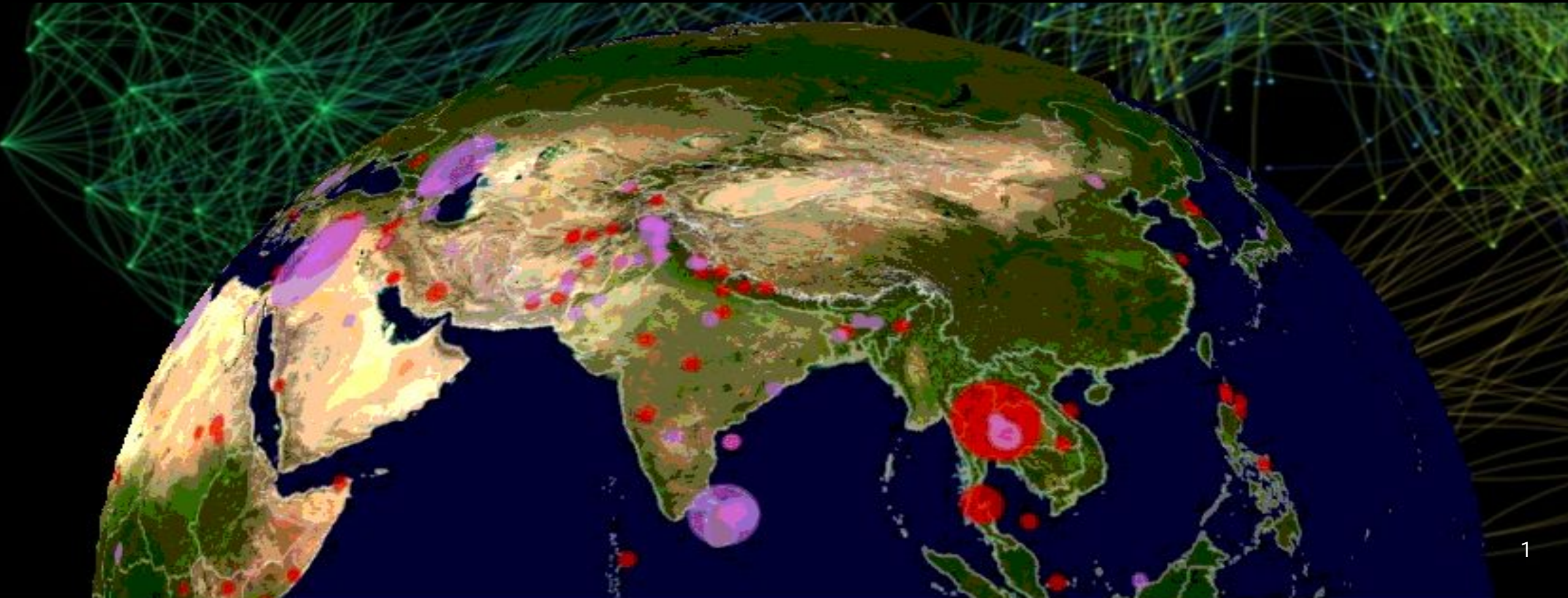


The GDELT Project

The data architecture
suited to the World of 2018





Équipe du projet



Sarra
Kazdaghli



Weijia
Du



Jean-
Baptiste
Scellier



Amine
Lemaizi



Paki
Parame-
swaran



Mickael
Lopes



Objectif

Proposer un **système de stockage** pour effectuer des requêtes sur les données de GDELT satisfaisant ces contraintes :

- Au moins **1 technologie vue en cours** utilisée
- Système **distribué** et **tolérant aux pannes**
- **Une année** de données
- Utiliser **AWS** pour déployer le cluster



Plan

- I. Modélisation & Architecture
- II. Implémentation, Volumétrie, Performance & Budget
- III. Requêtes
- IV. Démonstration

Modélisation & Architecture

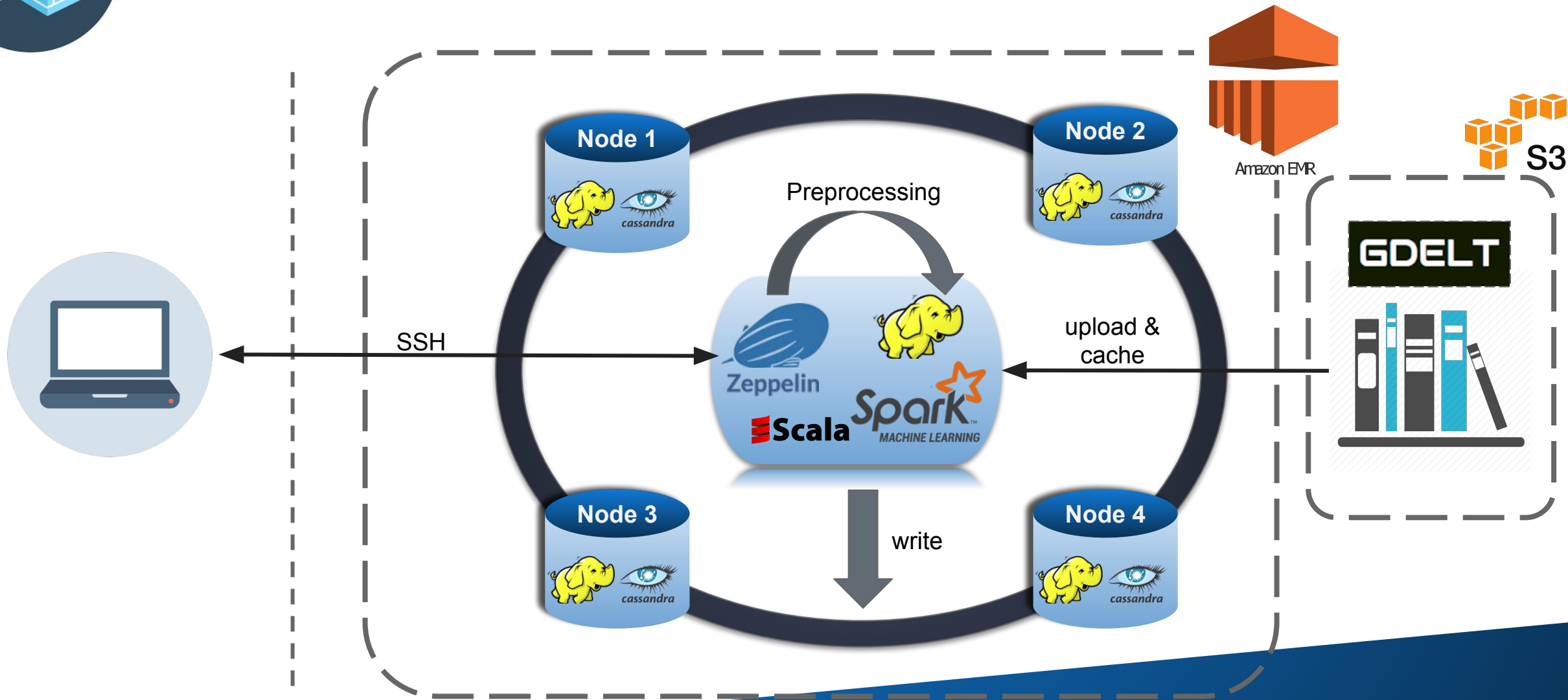
Cassandra MongoDB



	Cassandra	MongoDB
Modélisation	Structure de table traditionnelle	Orientée objet, riche et expressive
Noeud Maître	Multiple noeuds maître	Un seul maître
Mise à l'échelle	Plusieurs noeuds d'entrées et de sorties	Entrée unique par le noeud maître
Langage de requêtes	CQL similaire à SQL	Json
Agrégation	Nécessite des frameworks externes (Spark)	Agrégation intégrée
Schéma	Schéma de données statique	Modifiable et pouvant être non homogène
Performance	Supporte les grandes charges de données	Dépend du schéma de données utilisé



Architecture Générale



Implémentation



Configuration Cassandra

NOTRE OBJECTIF

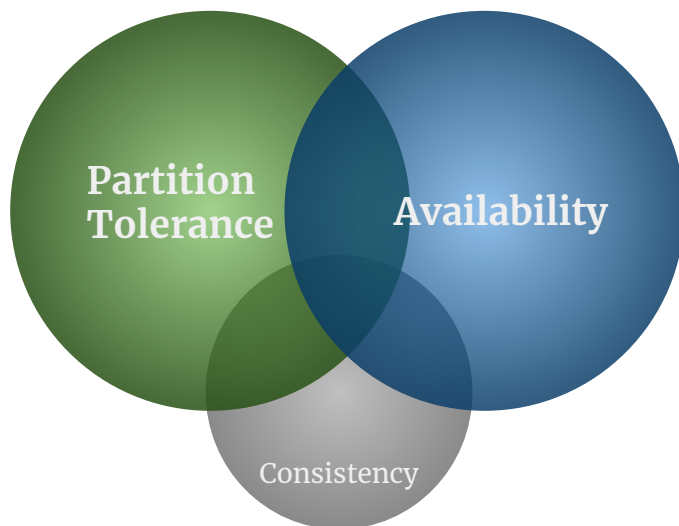
Disponibilité en écriture : max 1 noeud en panne
Disponibilité en lecture : max 2 noeuds en panne
Disponibilité globale : max 1 noeud en panne

Architecture AP

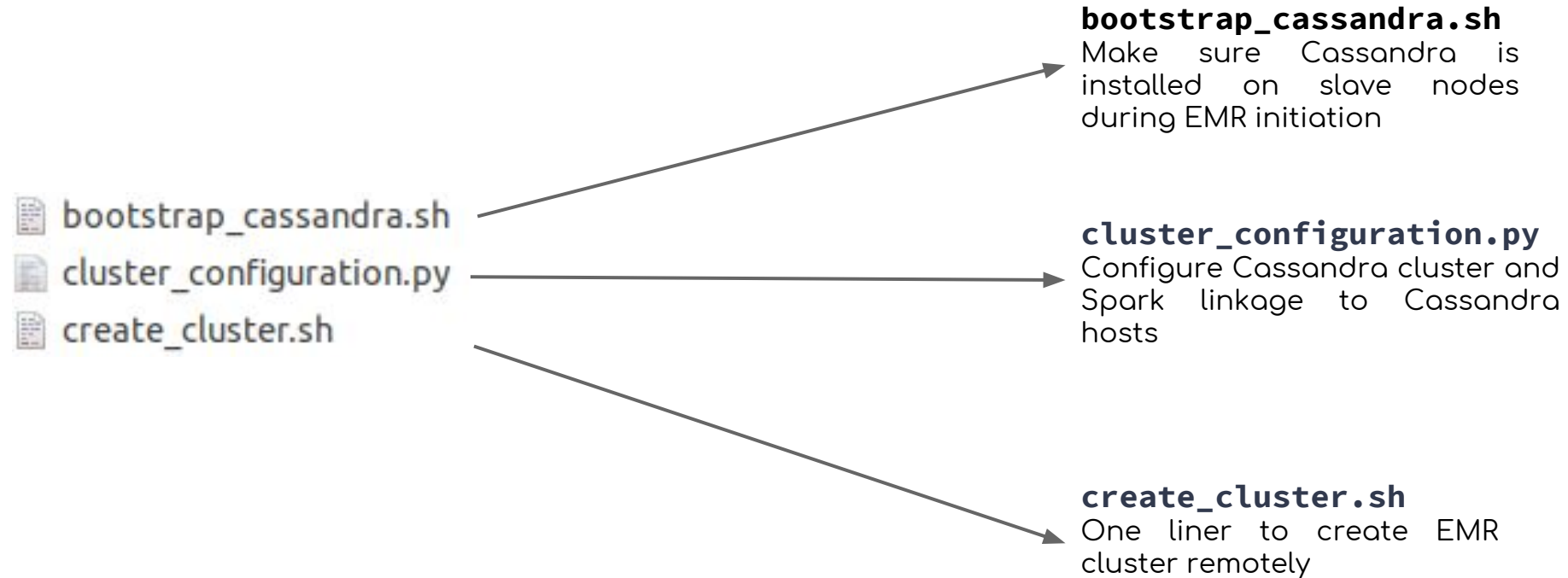
CONFIGURATION

- *Replication Factor* = 3
- *Écriture* = QUORUM (2)
- *Lecture* = ONE(1)

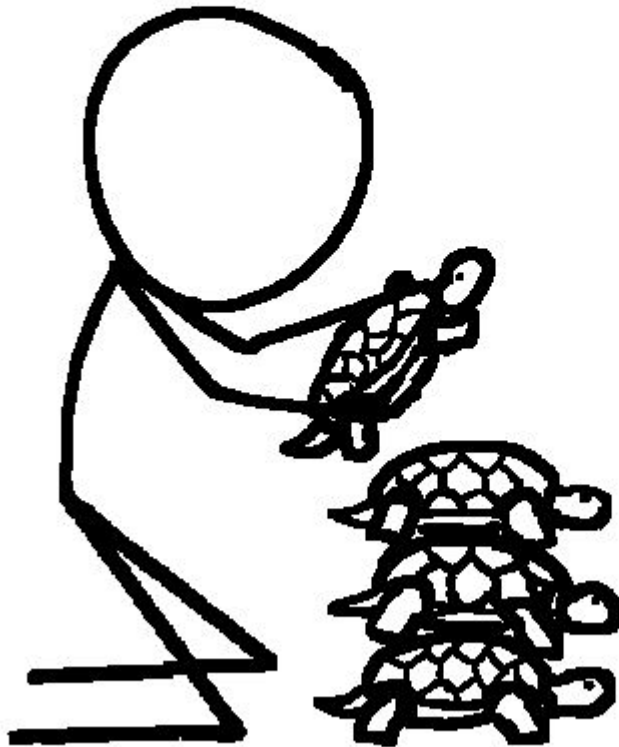
$W+R = RF$
COHÉRENCE À
TERME



EMR Automation



Working with EMR clusters be like...



- Premier cluster perdu...à 3h du matin avec toutes les données associées...
- Les jointures avec passage à l'échelle => Ressources consistantes nécessaires
- Théorie <-----> Pratique
- Prétraitement en Spark

Performances

Volumétrie

Tables sur S3:

- **events** : 32 millions de lignes
- **translation.mentions** : 78 millions de lignes
- **mentions (anglais)** : 220 millions de lignes
- **gkg** : 150 millions de lignes

Tables sur cassandra:

- **R1**, 1 mois: 906363 lignes
- **R2**, 6 mois: 16 millions de lignes
- **R3**, 1 mois: 906363 lignes
- **R4**, 1 mois: 4041 lignes
- **R5**, 6 mois: 602047 lignes

Vitesse de requêtage :

- Requête 1 - 5 : selon la fonctionnalité
10 secondes ~ 1 minute

Coût du système :

Amazon EMR

- m4.xlarge * 4 workers 1.92\$/h
~45h = 87\$

Requêtes

Prétraitement des données

1) Définition d'une classe pour chaque table :

```
case class Mention(GLOBALEVENTID: Int,  
  EventTimeDate: String,  
  MentionTimeDate: String,  
  MentionType: Int,  
  MentionSourceName: String,  
  MentionIdentifier: String,  
  SentenceID: Int,  
  Actor1CharOffset: Int,  
  Actor2CharOffset: Int,  
  ActionCharOffset: Int,  
  InRawText: Int,  
  Confidence: Int,  
  MentionDocLen: Int,  
  MentionDocTone: Double,  
  MentionDocTranslationInfo: String)
```

2) Création de DataFrame et concaténation pour les données mentions :

```
var df1 = cachedMentions.map(_.split("\t")).filter(_.length==15).map(  
  e=> Mention(  
    toInt(e(0)),parse_date(e(1),inputFormat_ymdhms, outputFormat_ymdhms),parse_date(e(2),inputFormat_ymdhms, outputFormat_ymdhms),toInt(e(3)),  
    e(4),e(5),toInt(e(6)),toInt(e(7)),toInt(e(8)),toInt(e(9)),toInt(e(10)),toInt(e(11)),toInt(e(12)),toDouble(e(13)), e(14).substring(6, 9))  
  ).toDF()  
  
var df2 = cachedMentions.map(_.split("\t")).filter(_.length==14).map(  
  e=> Mention(  
    toInt(e(0)),parse_date(e(1),inputFormat_ymdhms, outputFormat_ymdhms),parse_date(e(2),inputFormat_ymdhms, outputFormat_ymdhms),toInt(e(3)),  
    e(4),e(5),toInt(e(6)),toInt(e(7)),toInt(e(8)),toInt(e(9)),toInt(e(10)),toInt(e(11)),toInt(e(12)),toDouble(e(13)),"eng")  
  ).toDF()  
  
var mentions = df1.union(df2).toDF()
```

3) Transformation des colonnes à transcoder :

```
mentions = mentions.join(language_transco, mentions("MentionDocTranslationInfo")==language_transco("CODE"), "left").  
withColumn("Language", col("LABEL")).drop(language_transco.columns: _*)
```

4) Création d'une table Cassandra pour chaque requête :

```
val mentions_bis = mentions.select($"GLOBALEVENTID".alias("eventid"), $"MentionIdentifier".alias("documentid"), $"Language".alias("language")).distinct.where("language is not null")  
val events_bis = events.filter(month($"SQLDate_date")==="12").select($"GLOBALEVENTID".alias("eventid"), $"SQLDATE".alias("date"), $"ActionGeo_CountryName".alias("country")).where("country is not null")  
val articles_and_events = events_bis.join(mentions_bis, Seq("eventid"), "inner")  
  
articles_and_events.createCassandraTable(  
  "queries",  
  "articles_and_events",  
  partitionKeyColumns = Some(Seq("date", "country", "language")),  
  clusteringKeyColumns = Some(Seq("eventid")))  
  
articles_and_events.write.cassandraFormat("articles_and_events", "queries").save()
```

Query 1 - Nombre d'articles et d'évènements

articles_and_events		
eventid	Text	C
date	Date	K
country	Text	K
language	Text	K
documentid	Text	

Legend	
K	: Partition Key
C↓	: Sorting Key Desc
C↑	: Sorting Key Asc
<input type="text"/>	: CQL Type

```
SELECT date, country, language,  
       count(DISTINCT eventid / documentid)  
FROM articles_and_events  
WHERE language = "..."  
       AND date = "..."  
       AND country = "..."  
GROUP BY (date, country, language);
```

Query 2 - Évènements d'un acteur sur les 6 derniers mois

events_by_actor (main columns)		
actor_name	Text	K
event_date	Date	C↓
event_id	Text	C↑
actor_type1	Text	
co_actor_name	Text	
co_actor_type1	Text	
event_description	Text	
action_localisation	Text	
action_performed		
_by_actor	bool	

events_by_actor (other columns)	
actor_country	Text
co_actor_country	Text
actor_event_localisation	Text
co_actor_event_localisation	Text
event_quadclass	Text
event_nummentions	Text
event_articles	Text
event_avgtone	Text

Legend	
K	: Partition Key
C↓	: Sorting Key Desc
C↑	: Sorting Key Asc
	: CQL Type


Query 3 - Acteurs avec le plus d'articles positifs / négatifs

- Transposer column MentionDocTone -> Positif et Négatif
- 2 entrées : actor1 et actor2 -> union
- Sommer sur MentionDocTone pour chaque triplet

controversial_actors		
SQLMonth	Date	K
ActionGeo_CountryName	Text	K
Language	Actor	K
ActorCode *	Text	
ActorName *	Text	
GLOBALEVENTID	Text	C↓
Positive **	Float	
Negative **	Float	

- ActorCode = 1&2
- ActorName = 1&2
- Positive = if
MentionDocTone>0
- Positive = if
MentionDocTone>0


Legend

K : Partition Key
C↓ : Sorting Key Desc
C↑ : Sorting Key Asc
S : Static Column
 : CQL Type

Query 4 - Acteurs, pays, organisations qui divisent le plus

- Division, une action performé par un acteur divise le monde => on se restreint à l'acteur 1
- Acteurs (ou pays ou organisations) qui ont la plus grande variance de tone pour un event donné

division_by_actor		
EventDate	Date	K
ActorName	Text	C↑
Country	Text	C↑
Organization	Text	C↑
GlobalEventID	Text	C↑
Variance	Float	
Nb_article	Int	


Legend	
K	: Partition Key
C↓	: Sorting Key Desc
C↑	: Sorting Key Asc
S	: Static Column
	: CQL Type

Query 5 - Evolution des relations entre les différents pays

-- Part I

Evolution des relations entre **2 acteurs**
depuis **le ton sur les événements**

tone_actors_daily	
Actor1Name	K
Actor2Name	K
Date	C↑
Tone	


Legend	
K	: Partition Key
C↓	: Sorting Key Desc
C↑	: Sorting Key Asc
S	: Static Column
	: CQL Type

Query 5- Evolution des relations entre les différents pays

-- Part II

Evolution des relations entre **2 pays d'acteurs**
depuis le **ton** sur les événements

tone_country_daily	
Actor1Country	K
ActorCountry	K
Date	C↑
Tone	

Legend	
K	: Partition Key
C↓	: Sorting Key Desc
C↑	: Sorting Key Asc
S	: Static Column
	: CQL Type


Query 5- Evolution des relations entre les différents pays

-- Part III

Evolution des relations entre **2 pays** (pays du domaine de l'article, localisation de l'article)

depuis le ton sur les article (table gkg)

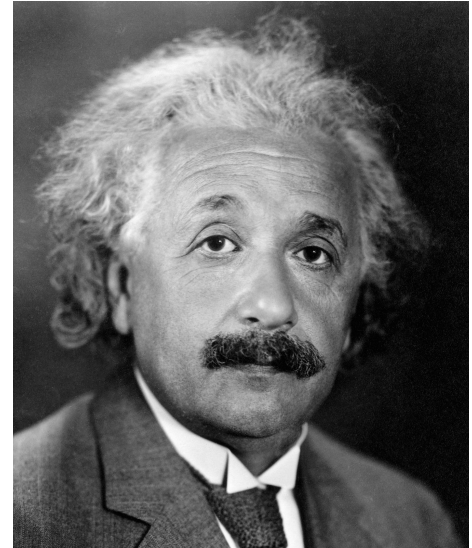
tone_country_daily_articles	
Country1	K
Country2	K
Date	C↑
Tone	

Legend	
K	: Partition Key
C↓	: Sorting Key Desc
C↑	: Sorting Key Asc
S	: Static Column
	: CQL Type

Conclusion

La **théorie**, c'est quand on sait tout et que rien ne fonctionne. La **pratique**, c'est quand tout fonctionne et que personne ne sait pourquoi. Ici, **nous avons réuni théorie et pratique** : Rien ne fonctionne... et personne ne sait pourquoi !

Albert Einstein



Démonstration

Merci pour votre
attention