Extension du framework SparkSQL pour une prise en charge efficiente des données spatiales massives dans les télécommunications.

El Hassane Nassif *,Hicham Hajji *,
Reda Yaagoubi *,Hassan Badir **

* Institut Agronomique et Vétérinaire Hassan II , Rabat
nassif.hassane@gmail.com, h.hajji@iav.ac.ma,r.yaagoubi@iav.ac.ma
http://www.iav.ac.ma
** Ecole de Sciences Appliqués de Tanger
hbadir@uae.ac.ma
http://www.ensat.ac.ma

Résumé. Nous assistons ces dernières années à une évolution explosive des données spatiales issues des activités : appels, sms et navigation internet des abonnés Télécoms. Ce type de données arrivent sous divers formats et à une fréquence de plus en plus élevée, à tel point que les exploiter relève du défi. En pratique, les opérateurs font face à un manque de solutions spécialisées dans l'analytique géospatiale sur des données massives. Ce qui rend difficile l'exploitation de la dimension spatiale dans les télécoms. Dans cet article, nous présentons une architecture Big Data intégrée pour la gestion des données spatiales massives dans les télécoms. Et nous expliquons les extensions effectuées sur le framework SparkSql afin de supporter les types et les requêtes spatiales.

1 Introduction

La concurrence et les nouvelles tendances du marché poussent les opérateurs de télécommunication à chercher des moyens innovants pour réaliser plus de marge. Un de ces moyens consiste à mettre en place une approche intégrée pour la géolocalisation de leurs clients. Ceci est possible à travers l'exploitation des traces géolocalisées des abonnés dans la réalisation des cas d'utilisations analytiques géospatiales. Pour ce faire et au vu des caractéristiques Big Data liées à l'évolution explosive du volume, de la vitesse et de la variété des données générées par les différentes activités des clients, il est nécessaire de s'orienter vers les technologies Big Data (Phaneendra et Reddy (2013)). Dans ce sens, nous avons proposé dans un précédent travail (E.Nassif et al. (2018)), une architecture Big Data intégrée pour la gestion des données spatiales massives dans le domaine des télécommunications. Vu que les outils de traitement Big Data qui existent sur le marché ne supportent pas le type spatial. Nous proposons dans ce papier d'étendre la nouvelle API SparkSQL (Armbrust et al. (2015)) du moteur de calcul parallèle Apache Spark [1] pour une prise en charge des données spatiales massives provenant des

1. https ://spark.apache.org

télécommunications. Contrairement à l'ancienne API Rdds [2] (Zaharia et al. (2012)), SparkSQL permet au développeur de lancer des requêtes SQL en mode parallèle sur des données massives et distribuées. Et offre la possibilité de transformer ces données en des tables en leur donnant un schéma bien défini.

2 Problématique et travaux liés

Grâce à l'avènement des réseaux de téléphonie mobile, toute personne munie d'un GSM allumé peut désormais être située dans l'ensemble des zones de couverture (Aguiléra et al. (2014)). Ce procédé permet à l'opérateur Télécom de géolocaliser les activités de ses abonnés afin de les exploiter dans plusieurs cas d'utilisations tel que le ciblage marketing.

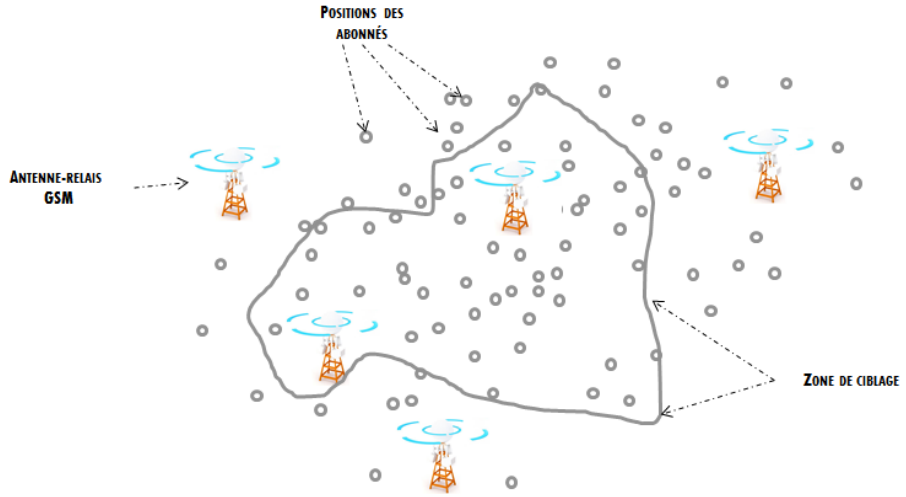

FIG. 1 – *Ciblage marketing des abonnés télécoms.*

Tel que indiqué sur la figure FIG. 1, le ciblage marketing consiste à cibler les clients se trouvant dans une zone d'intérêt en se basant sur leur dernière position géographique. C'est un cas d'utilisation qui est intéressant pour l'opérateur mais sa réalisation est confrontée à des contraintes liées à la nature Big Data des données traitées. En effet, les données issues des activités : appels, sms, internet ou simplement le fait d'allumer son téléphone sont très volumineuses (8 Téraoctets par jours, soit 97 Giga-octets par seconde), variées (réseaux sociaux et applications mobiles) et enfin la génération de toutes ces données se fait en continu. À ces contraintes s'ajoutent l'aspect temps réel ou presque des cas d'utilisations, l'aspect multidimensionnel des données de géolocalisation spatiale et la complexité des relations les reliant. Ainsi, pour identifier si un point se situe à l'intérieur d'un polygone par exemple, il faut faire un ensemble de calculs géométriques complexes et couteux en performance. Ce calcul est nécessaire à chaque événement récent des abonnés or nous en recevons 97 Go/s de donnés.

2. Resilient distributed dataset

Toutes ces caractéristiques nous amènent à dire que nous sommes face à des contraintes Big Data. Le seul moyen de les résoudre est de se tourner vers les solutions Big Data en espérant qu'elles supportent le type spatial. Malheureusement ce n'est pas le cas et le peu de travaux de recherche qui vont dans le sens d'étendre ces outils Big Data pour supporter les données spatiales ne prennent pas en charge les données télécom. En effet, nous n'avons pas trouvé dans la littérature des travaux de recherche qui se sont penchés sur la problématique de traitement des données spatiales massives dans les télécommunications. Par contre nous avons pu étudier un ensemble de papiers traitant la problématique de gestion des données spatiales massives en générale, sans se focaliser sur le contexte Télécom. Ces travaux peuvent être organisés en deux groupes : l'avant et l'après-Apache Spark. En effet, avant Spark nous avions SpatialHadoop (Eldawy et Mokbel (2015)), Hadoop-GIS (Aji et al. (2013)) et MD-Hbase (Nishimura et al. (2011)) qui ont implémenté des extensions spatiales sur la base de Hadoop MapReduce [3]. Ce qui leur a permis de profiter des avantages de Hadoop concernant la tolérance aux fautes et le calcul distribué, mais aussi d'hériter de ses problèmes. Car Hadoop est connu pour ses accès au disque en lecture/écriture à chaque itération, ce qui pénalise les performances.
Dès l'apparition de Spark, plusieurs travaux se sont fixés comme objectif de l'étendre pour qu'il puisse supporter les données spatiales. Spatial Spark [4] et GeoSpark (Yu et al. (2015)) par exemple sont des systèmes de gestion des données spatiales massives reposant sur Apache Spark. Ils utilisent la structure Rdds pour stocker les différentes géométries et sont plus performants que les trois précédents, car ils héritent de l'aspect in memory de Spark. Néanmoins ils n'offrent aucune interface d'interrogation en mode SQL, ce qui les rend difficilement exploitables. De plus et vu que c'est au développeur d'optimiser le code et non le moteur de calcul, cela génère des problèmes de performances.
D'où notre proposition d'étendre SparkSQL pour une prise en charge efficiente des données spatiales massives issues des activités des abonnés dans le domaine des télécommunications. En effet, notre extension de SparkSQL consiste en la création du type spatial à travers l'extension de l'interface native « UserDefinedType » offerte par Spark. Aussi, nous définissons un ensemble de fonctions Spark pouvant exploiter la structure spatiale dans des calculs topologiques en mémoire.
D'autres travaux de recherche se sont penchés sur le même sujet, mais en suivant une démarche différente. Il s'agit de GeoSparkSQL (Huang et al. (2017)). Ce dernier offre lui aussi une interface SQL d'interrogation de données spatiales massives, mais en se basant sur une structure spatiale stockée sur Hive [5] à travers l'utilisation d'une API ESRI [6] qui permet de charger les données depuis les fichiers de type Shapefile [7] vers des tables Hive contenant des colonnes dédiées pour l'indexation. De la même façon, GeoSparkSQL utilise une deuxième API d'Esri qui implémente les requêtes spatiales sur Hadoop [8]. Ce qui signifie que le rôle de Spark dans GeoSparkSQL est réduit à la conversion des requêtes SparkSQL en des requêtes Hive utilisant les opérateurs offerts par l'API d'ESRI. Ainsi, et vu que GeoSparkSQL offre une interface SQL de requettage spatiale via du Hadoop Mapreduce alors il est important de noter que l'on va rencontrer les mêmes problématiques de performance connues sur Hadoop à cause de la

3. http :// hadoop.apache.org
4. http ://simin.me/projects/spatialspark/
5. https ://hive.apache.org/
6. www.esri.com/
7. Format de fichier pour les systèmes d'informations géographiques
8. https ://github.com/Esri/spatial-framework-for-hadoop

sollicitation des disques pour chaque lecture/écriture.

3 Architecture de la solution

3.1 Intégration de la couche spatiale dans Apache Spark

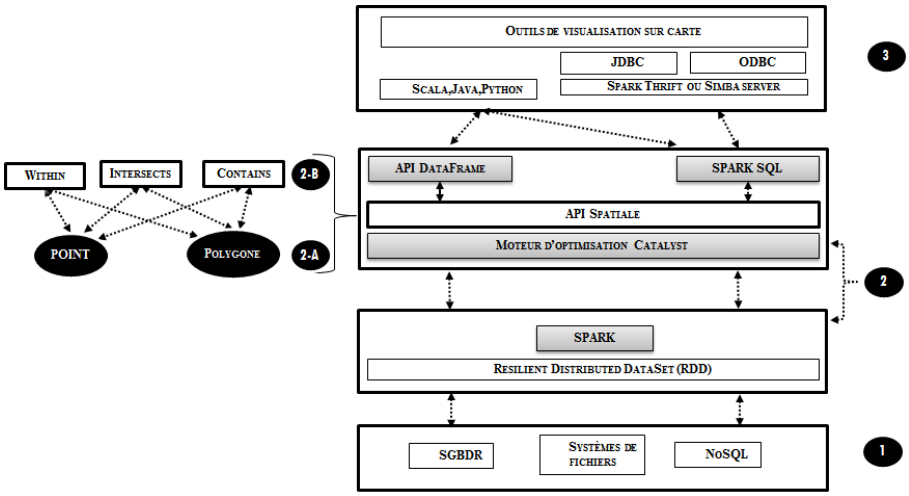

FIG. 2 – *Architecture de la solution.*

La solution que nous proposons se base sur le noyau du moteur de calcul parallèle apache Spark. Ce dernier comporte nativement des bibliothèques Scala, Python et Java et offre plusieurs API chacune adaptée à une problématique bien définie, telles que le traitement des graphes ou les traitements d'apprentissage automatique. Dans le cadre de cet article nous nous intéressons particulièrement aux deux API : SparkSQL et DataFrame (Armbrust et al. (2015)). Elles offrent une implémentation SQL-like et donnent la possibilité d'interroger les données en mode relationnel comme si elles étaient des tables avec un schéma bien défini. Ceci est rendu possible grâce à la nouvelle abstraction DataFrame qui est une collection distribuée d'enregistrements ayant le même schéma. Elle suit la même logique que les Rdds (Zaharia et al. (2012)), c'est-à-dire que les DataFrames sont immuables, distribuées, et sont évaluées paresseusement par les opérations Spark. Chaque Dataframe représente un plan logique de calcul qui est transformé par Spark en un plan physique d'exécution au moment de l'appel des opérations de type actions. Le DataFrame, peut-être construit depuis toutes les sources de données supportées par Spark, qu'elles soient distribuées ou non (zone 1 de la figure FIG. 2) avec la possibilité aussi de convertir un Rdd en Dataframe. De plus Spark offre la possibilité d'appeler ces requêtes SparkSQL depuis des applications tierces via Thrift [9] pour les connexions Jdbc [10] et Simba

9. https ://thrift.apache.org/
10. Java Database Connectivity, interface de connexions aux bases de données.

Server [11] pour les connexions Odbc [12] (zone 3 de la figure FIG. 2). Toutes ces fonctionnalités offrent aux développeurs la possibilité d'appeler via une simple requête SQL des bibliothèques de calculs complexes sur plusieurs types de données et de paralléliser leur exécution sur plusieurs serveurs. Ceci convient parfaitement aux données spatiales massives télécoms à condition de faire en sorte à ce que Spark puisse reconnaitre la structure multidimensionnelle de la composante spatiale. En réalité, rien ne nous empêche d'utiliser des structures classiques pour stocker une géométrie de type point par exemple, mais on sera dans l'incapacité d'effectuer des tests d'appartenance sur un polygone par exemple. Pour parer à cette limite, nous avons rajouté une couche spatiale (zone 2 de la figure FIG. 2) qui étend Spark. Et lui permet de reconnaitre le type spatial (zone 2-A de la figure FIG2) et de le manipuler de façon structurée comme si s'il était une table. De cette façon nous rendons Spark plus accessible aux développeurs du domaine spatial télécom en leur permettant de s'affranchir des complexités connues sur les autres travaux d'extensions spatiales de Spark telle que la programmation fastidieuse en Rdds (Tang et al. (2016)) que nous avons remplacée par une programmation plus productive basée sur SparkSQL.

3.2 Flux d'exécution des traitements

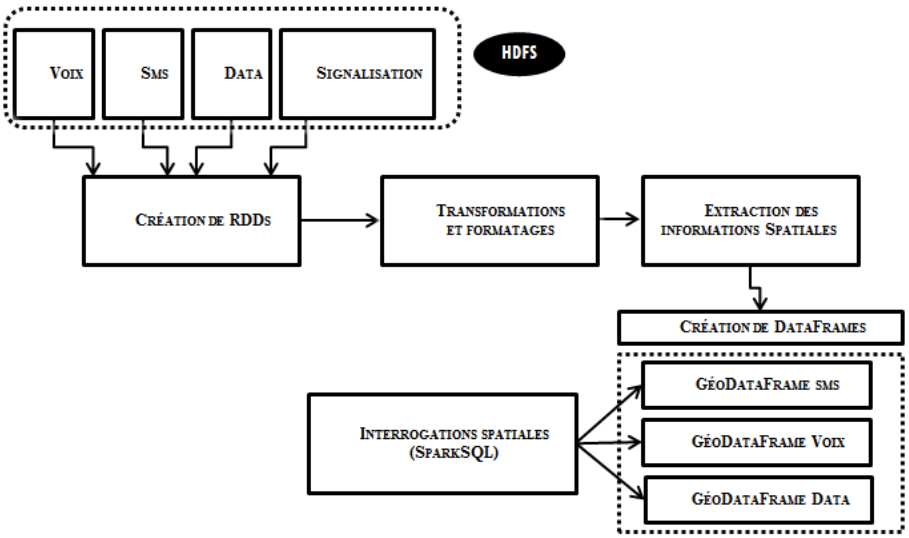

FIG. 3 – *Data pipeline spatial distribué.*

La figure FIG. 3 présente le modèle d'exécution distribué de notre solution. Ce dernier suit une logique pipeline adaptée aux cas d'utilisations nécessitant le traitement des données spatiales télécoms. La première étape d'exécution consiste à lire les fichiers de données via les APIs de la couche d'accès Spark. Dans cet article, nous considérons que toutes les don-nées qui

11. https ://www.simba.com/
12. Open Database Connectivity, interface de connexions aux bases de données.

arrivent depuis les sources télécoms sont stockées de façon distribuée sur HDFS [13] . Cette opération de lecture permet de générer des objets RDDs, sur lesquels nous agissons pour appliquer un ensemble de transformations telles que le formatage des dates, des numéros de téléphone et des coordonnées GPS. L'étape qui suit consiste à ne garder que les informations utiles et qui sont l'horodatage de l'événement, sa géolocalisation, l'identifiant de l'abonné et puis d'autres informations qui servent à caractériser l'activité telles que les attri'buts mesurables (montant, durée..). Par la suite nous créons nos Dataframes de type Point sur la base d'un schéma qui reprend la structure spatiale et qui est défini à l'avance. Et ce sont ces DataFrames qui servent à répondre à nos requêtes spatiales sur la base des activités géolocalisées des abonnées. Ceci est possible dans SparkSQL, car il permet de rajouter un type personnalisé en étendant la classe UserDefinedType (Armbrust et al. (2015)) et de l'utiliser pour créer des Dataframes qu'on peut interroger en mode SQL. Cette fonctionnalité nous permet de définir une sorte de mapping bidirectionnel entre notre type spatial et ceux spécifiques à Spark. De cette façon nous sommes capables dans notre code d'interroger des objets ayant un type spatial tels que les Points ou les Polygones et ces derniers sont reconnus par le moteur SparkSQL. De plus SparkSQL pourra les stocker, si on le souhaite, sous un format en colonne en compactant individuellement chaque attribut dans une colonne dédiée.Une fois le type spatial personnalisé : PointUDT créé, nous pouvons l'utiliser pour créer notre Dataframe spatial.

3.3 Description de l'implémentation

La figure suivante FIG. 4, illustre la création du nouveau type spatial Point dans SparkSQL en étendant la classe UserDefinedType.

```
import org.apache.spark.sql.catalyst.InternalRow
import org.apache.spark.sql.SQLContext
import org.apache.spark.{SparkContext, SparkConf}
import org.apache.spark.sql.catalyst.InternalRow
import org.apache.spark.sql.types._
case class Point(val GeometryType: Int, val Longitude: Double, val Latitude: Double) { }
class PointUDT extends UserDefinedType[Point] {
 override def sqlType: DataType = ArrayType(DoubleType, false)
 override def serialize(obj: Any): Seq[Double] = {
    obj match {      case p: Point =>         Seq(p.GeometryType,p.Longitude, p.Latitude)     }  }
 override def deserialize(datum: Any): Point = {
    val row = datum.asInstanceOf[InternalRow]
    require(row.numFields == 3)
  Point(row.getInt(0),row.getDouble(1),row.getDouble(2))
  }
@SQLUserDefinedType(udt = classOf[PointUDT])
 override def userClass: Class[Point] = classOf[Point]
}
```

FIG. 4 – *Extension de SparkSQL et création du nouveau type spatial Point.*

La figure FIG. 5(zone1) illustre l'opération d'extraction des données depuis les fichiers sources et leur utilisation pour instancier notre type spatial Point. De cette façon nous créons notre Dataframe GeoDataVoix qui peut être interrogé en mode SQL tel que le montre la zone «

13. Hadoop Distributed File System

1 » de la figure FIG. 5. Nous l'utilisons aussi pour créer notre deuxième Dataframe Polygone qui représente une zone de ciblage de type carré que nous pouvons interroger aussi en mode SQL (voir la zone 2 de la figure FIG. 5).

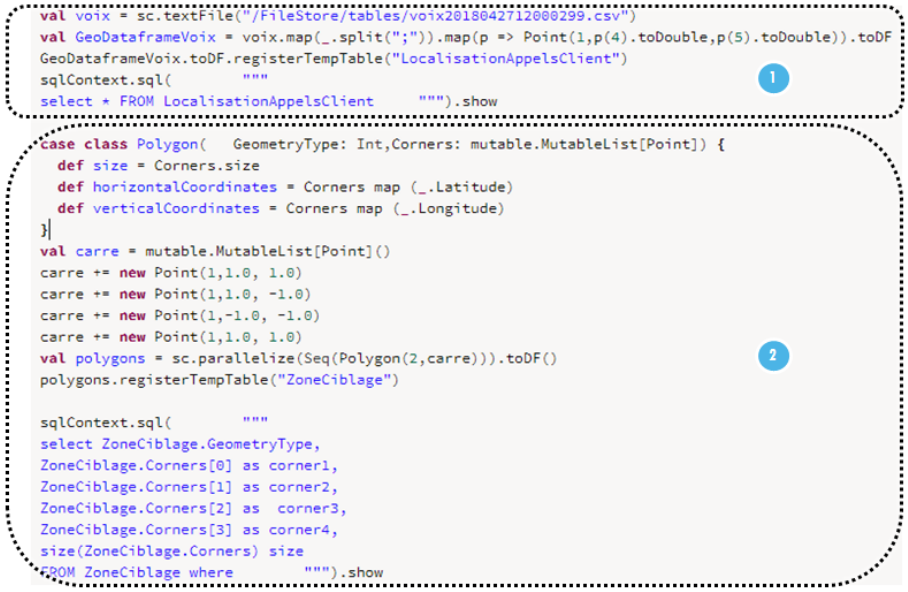

FIG. 5 – *Création des DataFrames Point et Polygone.*

Il ne reste qu'à implémenter les différents prédicats géométriques pour consommer les données disponibles à travers nos différents Dataframes. Cela peut se faire par SparkSQL directement dans le code SQL, mais ce n'est pas la bonne approche. En fait, SparkSQL met à notre disposition un autre moyen plus pratique d'implémenter nos prédicats. Il s'agit des fonctions personnalisées UDF [14] qui offrent la possibilité d'encapsuler toute notre logique spatiale dans des fonctions que nous pouvons appeler dans des requêtes SQL.

```
sqlContext.sql(
""" select LocalisationAppelsClient.*
FROM
LocalisationAppelsClient,
ZoneCiblage
where
PointWithinPolygon(LocalisationAppelsClient, ZoneCiblage) = true """).show
```

FIG. 6 – *Appel d'un prédicat spatial à travers un UDF dans SparkSQL*

14. User defined function

La figure FIG. 6 illustre un exemple d'utilisation d'un prédicat spatial « Within » dans une requête SparkSQL afin de vérifier si un point est inclus dans un polygone.

4 Conclusion

La combinaison de trois contraintes principales caractéristiques des Big Data spatiales (volume, variété, vitesse) rend inadéquates les architectures techniques existantes des opérateurs Télécom. En effet, l'exploitation des données spatiales massives à leur juste valeur requiert la mise en place d'architectures basées sur de nouveaux outils Big Data. Ces outils jusque-là inadaptés pour le spatial offriraient aux opérateurs un champ nouveau d'exploitation, celui de l'exhaustivité et du temps réel. Dans ce sens, nous nous intéressons notamment au moteur big data de calcul parallèle (Zaharia et al. (2012)). Et plus particulièrement à l'API SparkSQL (Armbrust et al. (2015)) que nous avons étendue. Nous avons donc montré à travers un exemple pratique en relation avec le ciblage marketing qu'il est tout à fait possible d'exploiter les données spatiales massives dans les télécommunications. Ainsi, l'extension que nous avons effectuée sur Spark permet à l'analyste télécom de confectionner des requêtes spatiales de façon pratique et productive. Par exemple, pour répondre à la requête spatiale complexe décrite dans la figure FIG. 5, notre extension de SparkSQL permet de lancer une simple requête SQL sur l'ensemble des localisations clients pour retrouver celles qui correspondent au critère PointWithinPolygon. Bien entendu, les performances dépassent largement celles des architectures classiques. Cependant, rechercher dans toutes les positions géographiques est couteux en performance même dans le cas d'un moteur performant comme Spark. C'est pour cette raison que nous avons rajouté une couche d'indexation basée sur les courbes qui remplissent l'espace : Lebesgue (BÖxhm et al. (1999)). Nous n'avons pas pu développer ce deuxième volet de nos travaux dans ce papier [15], nous le ferons dans un prochain article.

Références

Aguiléra, V., S. Allio, et C. Milion (2014). Territory analysis using cell-phone data. *Proc. of the 5th Transportation Research Arena, Paris, France*.

Aji, A., F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, et J. Saltz (2013). Hadoop gis : a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment 6*(11), 1009–1020.

Armbrust, M., R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. (2015). Spark sql : Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 1383–1394. ACM.

BÖxhm, C., G. Klump, et H.-P. Kriegel (1999). Xz-ordering : A space-filling curve for objects with spatial extension. In *International Symposium on Spatial Databases*, pp. 75–90. Springer.

15. Besoin de plus de pages pour développer le volet indexation.

Eldawy, A. et M. F. Mokbel (2015). Spatialhadoop : A mapreduce framework for spatial data. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pp. 1352–1363. IEEE.

E.Nassif et al., Hajji Hicham, R. Y. H. B. (2018). Vers une architecture intgre pour la gestion des donnes spatiales massives en tlcommunications. *Communications of the Big data and Applications 12th edition of the Conference on Advances of Decisional Systems*.

Huang, Z., Y. Chen, L. Wan, et X. Peng (2017). Geospark sql : An effective framework enabling spatial queries on spark. *ISPRS International Journal of Geo-Information 6*(9), 285.

Nishimura, S., S. Das, D. Agrawal, et A. El Abbadi (2011). Md-hbase : A scalable multi-dimensional data infrastructure for location aware services. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, Volume 1, pp. 7–16. IEEE.

Phaneendra, S. V. et E. M. Reddy (2013). Big data-solutions for rdbms problems-a survey. In *12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)(Osaka, Japan, 19 Apr 2013)*.

Tang, M., Y. Yu, Q. M. Malluhi, M. Ouzzani, et W. G. Aref (2016). Locationspark : a distributed in-memory data management system for big spatial data. *Proceedings of the VLDB Endowment 9*(13), 1565–1568.

Yu, J., J. Wu, et M. Sarwat (2015). Geospark : A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 70. ACM.

Zaharia, M., M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, et I. Stoica (2012). Resilient distributed datasets : A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 2–2. USENIX Association.

Summary

In recent years, we have been witnessing an explosive evolution of the spatial data coming from Telecom subscriber's activities, such as calls, sms and the internet browsing. This type of data comes in various formats and with a high velocity, which makes their usage a real challenge. In practice, operators face a lack of specialized solutions in geospatial analytics on massive data. In this paper, we present an integrated Big Data architecture for the management of massive spatial data in telecoms and our extensions made on the SparkSql framework to support spatial types and queries.