PARALLEL POINT CLASSIFICATION INTO GEOGRAPHICAL REGIONS

by

Sanver Tarmur

B.S., Computer Engineering, Boğaziçi University, 2011

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computational Science and Engineering

Boğaziçi University

2018

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor Prof. Can Özturan for his continuous support to my thesis study and related research. His guidance, support and motivation helped me to finalize this thesis.

ABSTRACT

PARALLEL POINT CLASSIFICATION INTO GEOGRAPHICAL REGIONS

The amount of data generated by social media, social networks and distributed platforms such as blockchain, have reached quite high levels. There are various use-cases to process this huge amount of data. One is to classify the geo-tagged data which is produced by social networks into geographical regions. We propose an efficient parallel classification approach and implement a classifier tool which is capable of processing huge geographical point data in parallel. Twitter data from five major cities of Turkey is used as classification test set considering the density of the regions. There are important factors effecting the classification performance such as spatial indexing and parallelization strategy. Hierarchical Triangular Mesh (HTM) and R-Tree spatial indexes are used for indexing regions and open-source Apache Spark and Kafka platforms are used to implement our classification application in a distributed and scalable environment. The mentioned platforms are designed to handle huge data streams and quickly respond varying volume of data traffic. Benchmarks are provided in thesis to show effectiveness of our approach against built-in spatial index of Microsoft SQL Server and approach of Kondor et al. [1] in which HTM is applied on SQL Server. Our method has significant advantage since it is built upon Apache Spark platform which is crafted for processing chunks of data stream in real-time, however other approaches are based on SQL Server which cannot efficiently process massive streaming data. 1.6-4.5 fold speed-ups have been obtained in classification performance. The speed-up factor may change according to the query set size. Since our system has a scalable architecture it is possible to expand query set to billions of records. Apart from improved performance, our method is cost-effective since Twitter data collected over a month can be processed on cloud in around 3 hours with a small cost.

ÖZET

NOKTALARIN COĞRAFİ BÖLGELERE PARALEL SINIFLANDIRILMASI

Sosyal medya, sosyal ağlar ve blokzincir türevi dağıtık platformlar tarafından üretilen veri miktarı son yıllarda hissedilir seviyede artmıştır. Veri miktarının bu derecede artması ile, toplanan verinin analiz edilmesi ve işlenmesi ile ilgili uygulamalar ortaya çıkmıştır. Bu uygulamalardan bir tanesi coğrafi olarak etiketlenmiş olan verinin hangi coğrafi bölgeye ait olduğunun efektif ve hızlı bir biçimde bulunmasıdır. Bu çalışmada, coğrafi noktaları coğrafi alanlar üzerine paralel sınıflandıran bir metod önerilip, bir yazılım aracı olarak kodlanmıştır. Kodlanan yazılımı test etmek amacıyla Twitter üzerinden, nüfus yoğunluğunu hesaba katarak Türkiye'nin en yoğun bölgelerinin beşinden veri toplanmıştır. Coğrafi sınıflandırma performansını etkileyen en önemli faktörler kullanılan coğrafi endeks ve parallelleştirme stratejisidir. Uygulamamız, Hierarchical Triangular Mesh (HTM) ve R-Tree coğrafi endekslerden ve açık kaynak kodlu, dinamik veri miktarina göre uygulama ihtiyaçlarına cevap veren Apache Spark ve Kafka platformlarından faydalanılarak ölçeklenebilir ve dağıtık yapıda geliştirilmiştir. Microsoft SQL Server'in sunduğu coğrafi endeks ve Kondor et al. [1] tarafından HTM ile SQL Server'da geliştirilen metod, önerdiğimiz metod ile performans yönünden karşılaştırılmıştır. Uygulamamız, veri akımlarını hafıza üzerinde işleyen Spark üzerine inşa edildiği için, akımları efektif olarak işleyemeyen İlişkisel Veri Yönetim Sistemi bağımlı yaklaşımlara göre yüksek performans göstermektedir. Geliştirdiğimiz metod ile sorgu kümesinin büyüklüğüne bağlı olarak sınıflandırma süresi 1.6 ila 4.5 kat arasında hızlanma göstermiştir. Ayrıca tasarladığımız sistem, ölçeklenebilir mimarisi sayesinde milyarlar mertebesinde veriyi işleme olanağı sunmaktadır. Metodumuz performans artırmanın yanında maliyeti azaltmaktadır. Zira üç saat gibi kısa bir sürede bulut üzerinde bir aylık Twitter verisini çok düşük maliyet ile sınıflandırır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES vii
LIST OF TABLES ix
LIST OF ACRONYMS/ABBREVIATIONS x
1. INTRODUCTION 1
1.1. Contributions of Thesis 5
2. BACKGROUND WORK 10
2.1. Hierarchical Triangular Mesh Internals 10
2.2. Interval Skip List Internals 13
2.3. Overview of Apache Kafka 20
2.4. Overview of Apache Spark 22
2.5. Overview of Amazon Cloud and EMR 24
3. RELATED WORK 25
3.1. Summary 28
4. METHODOLOGY OF PARALLEL POINT CLASSIFICATION 30
4.1. Data Collection and Data Cleaning 30
4.2. Mechanism 33
4.3. Software Package Details 40
5. EXPERIMENTS AND RESULTS DISCUSSION 42
5.1. Results of HTM Indexing on Apache Spark with small infrastructure configuration 43
5.2. Results of HTM Indexing on Apache Spark with large infrastructure configuration 44
6. CONCLUSION 46
REFERENCES 47
APPENDIX A: KAFKA CLOUDFORMATION SCRIPT 51

LIST OF FIGURES

Figure 1.1. Basic classification example of randomly selected simple polygons and points. 1

Figure 1.2. Polygon of Istanbul on HTM index grid with HTM hierarchy projection. 2

Figure 1.3. HTM index grid with partial trixels between two neighboring regions marked as green. 4

Figure 1.4. Application Input/Output example and template. 6

Figure 2.1. HTM index in ascending resolution level. 11

Figure 2.2. Level 20 HTM ID intervals generated for randomly selected polygons. 14

Figure 2.3. Interval Skip List Visual Representation. 15

Figure 2.4. Interval skip list search algorithm. 17

Figure 2.5. Interval skip list algorithm to add new interval. 18

Figure 2.6. Overall Spark architecture of parallel point classifier. 24

Figure 4.1. Tweeter Stream response JSON object. 31

Figure 4.2. Reverse geocoding response of Nominatim contains OpenStreetMap-Data. 32

Figure 4.3. Polygon specification. 34

Figure 4.4. Kafka topic partitioning and Spark executors processing data stream. 35

Figure 4.5. End-to-end process scheme of parallel point classifier with all components. 38

Figure 4.6. Point classification algorithm. 39

Figure A.1. Amazon Cloudformation script to setup Kafka cluster. 51

LIST OF TABLES

Table 2.1. Level 20 HTM ID intervals covering Ankara region which are stored inside interval skip list nodes. 16

Table 3.1. Comparison of research studies for spatial indexing and point classification against our dissertation. 29

Table 4.1. Ratio of location correctness for Tweeter data collected via Tweeter public stream across regions. 33

Table 5.1. Classification accuracy of indexing strategies measured for regions over Turkey. 42

Table 5.2. Average classification time of HTM on Apache Spark with snappy compression against SQL server built-in and HTM indexing. 43

Table 5.3. Average classification time of HTM on Apache Spark without using Kafka message compression. 44

Table 5.4. Average classification latency values of HTM on Apache Spark with snappy compression against SQL server built-in and HTM indexing. 45

Table 5.5. Average classification latency values of HTM on Apache Spark without compression. 45

LIST OF ACRONYMS/ABBREVIATIONS

RDBMS	Relational Database Management System
HTM	Hierarchical Triangular Mesh
RDD	Resilient Distributed Dataset
SQL	Structured Query Language
CPU	Central Processing Unit
RAM	Random Access Memory
K	Thousand
M	Million
ID	Identifier
AWS	Amazon Web Services
GIS	Geographic Information System
EMR	Elastic Map Reduce
HEALPix	Hierarchical Equal Area isoLatitude Pixelisation
JTS	Java Topology Suite
API	Application Programming Interface
JAR	Java Archive
JVM	Java Virtual Machine
OS	Operating System
GPS	Global Positioning System
BNF	Backus–Naur form
HTTP	Hypertext Transfer Protocol
VPC	Virtual Private Cloud
URL	Uniform Resource Locator
HDFS	Hadoop Distributed File System
MBR	Minimum Bounding Rectangle
TCP	Transmission Control Protocol
REST	Representational State Transfer

1. INTRODUCTION

Point classification addresses the problem of mapping geographical points into the regions on Earth surface where the points are contained in. In context of point classification, the region and points could be defined with various coordinate systems however most commonly used one is latitude/longitude based spherical coordinate system which is used in this thesis work. A point corresponds to a latitude/longitude pair showing a location on sphere. On the other hand, regions are represented with polygon shapes that correspond to an array of latitude/longitude pairs defining an enclosed area on sphere. In Figure 1.1 shows basic polygons on an arbitrary region where points inside these polygons and their corresponding classification mappings are shown.

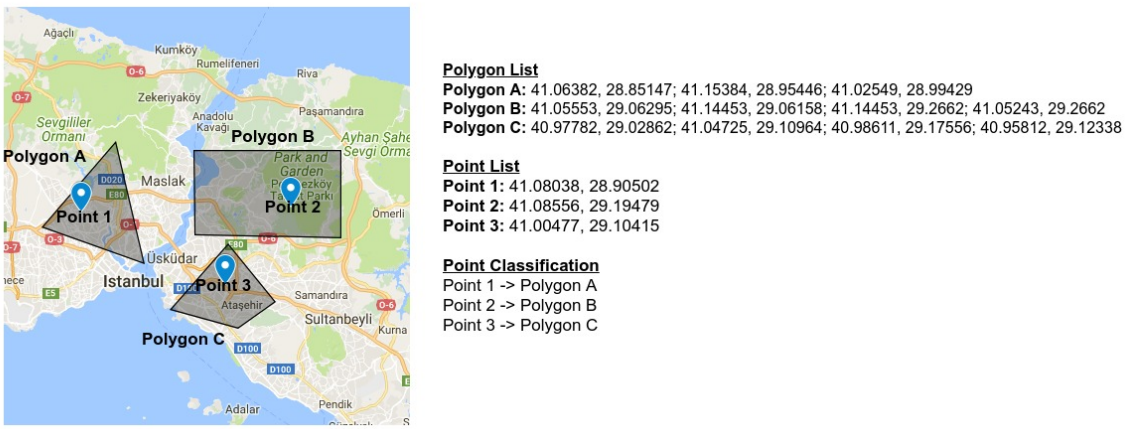

Figure 1.1. Basic classification example of randomly selected simple polygons and points.

At first glance, point classification problem looks straightforward. However, if we consider that the volume of spatial data generated by social networks like Twitter is increasing rapidly, classification of big volume of spatial data in reasonable time and resources becomes a quite complex challenge. According to the recent statistics, on Twitter platform there are around 330M active users and the total number of tweets produced in a day is around 500M which corresponds to an enormous amount of data on the network. On the other hand, point classification and spatial search operations such

as range or k-nearest-neighbour queries have been increasingly popular since extracting information from raw location data proposes valuable analytic outcomes. To overcome the complexity of problem and effectively classify the large data sets, the general approach of research studies is to apply spatial index structures. Also, processing the large sets in parallel improves performance and community Cloud environments provide economical infrastructure to the parallel classification. In this study, we propose a parallel point classification method that works on Cloud environment and exploits from spatial indexing strategies. The spatial indexing strategies that we use are Hierarchical Triangular Mesh (HTM) [2] and R-Tree [3] indexing. In Chapter 2, we provide details of HTM and in Chapter 4 we get into details of performance effect of HTM and R-Tree structures and architectural and procedural details of our method.

In context of our approach, the first step is to generate HTM spatial index structure over administrative regions which are loaded as polygons. In Figure 1.2, we provide a visual illustration of HTM index grid and a complex polygon residing over the grid.

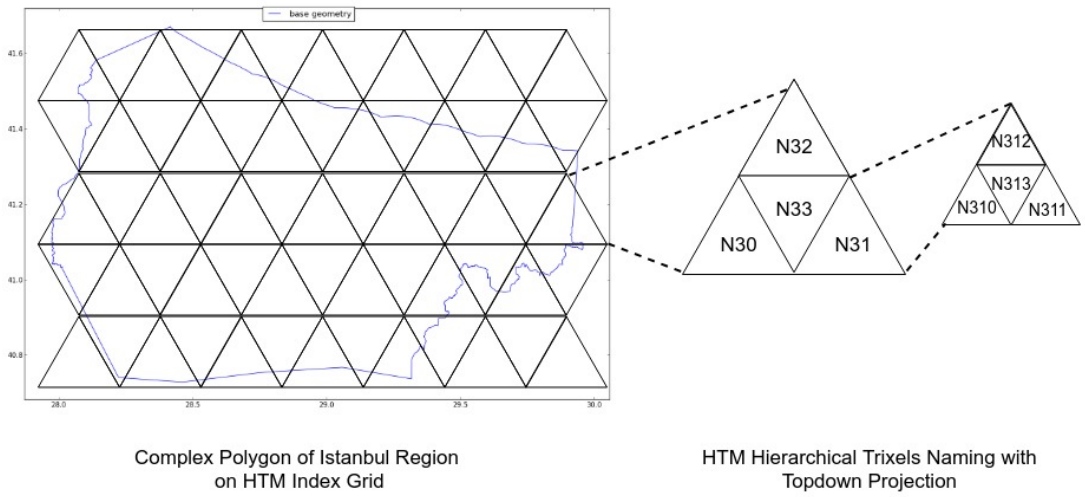

Figure 1.2. Polygon of Istanbul on HTM index grid with HTM hierarchy projection.

Besides the HTM index we generate an R-Tree index with the very same polygons. The reason behind the R-Tree index generation is to resolve partial trixel problem of HTM indexing structure. In further chapters, HTM internals and partial trixel problem

are discussed in detail. Briefly, HTM subdivides regions into trixels and there are three different trixel types which are full, partial and outer trixels. The full and outer trixels are straightforward since the prior one is completely inside the indexed region and the latter one is outside the region. However, partial trixels partially intersect with the region. That is why for points that are located inside partial, trixel creates ambiguity from classification task perspective. Figure 1.3 depicts an HTM index grid covering a region which includes partial trixels marked with green color. In case, we have two neighboring regions both having the same partial trixel in their index structure, this situation affects classification accuracy slightly. Even if the effect is not major, ambiguity could cause flaw in results. To overcome partial trixel problem, we exploit supportive R-Tree index which adds constant time complexity to the overall time complexity of method since partial trixels can be part of two or at worst-case three regions.

As it is mentioned before, spatial indexing techniques are important and necessary with regards to geospatial data processing performance. Besides HTM and R-Tree approaches, there are wide variety of index structures created in research studies about Hierarchical Equal Area isoLatitude Pixelisation (HEALPix) [4], quad-tree and more R-Tree variations. Among indexing approaches, HTM provides a uniform, globally continuous and hierarchical representation of regions on sphere, so that it is found to be beneficial for many academic studies. With the uniformly generated trixels by the HTM indexing approach, regions in polygon shape are represented by a sequence of trixel IDs namely HTM IDs. As of the continuous structure, it is possible to keep these IDs as intervals, basically pairs of 64-bit integer numbers.

HTM ID intervals can form a basis of representation for regions as well as geographical points so that points could be mapped to unique HTM IDs. Our technique is to store the generated HTM IDs for all regions in a space efficient and serializable data structure called interval skip list which copes with distributed processing environment.

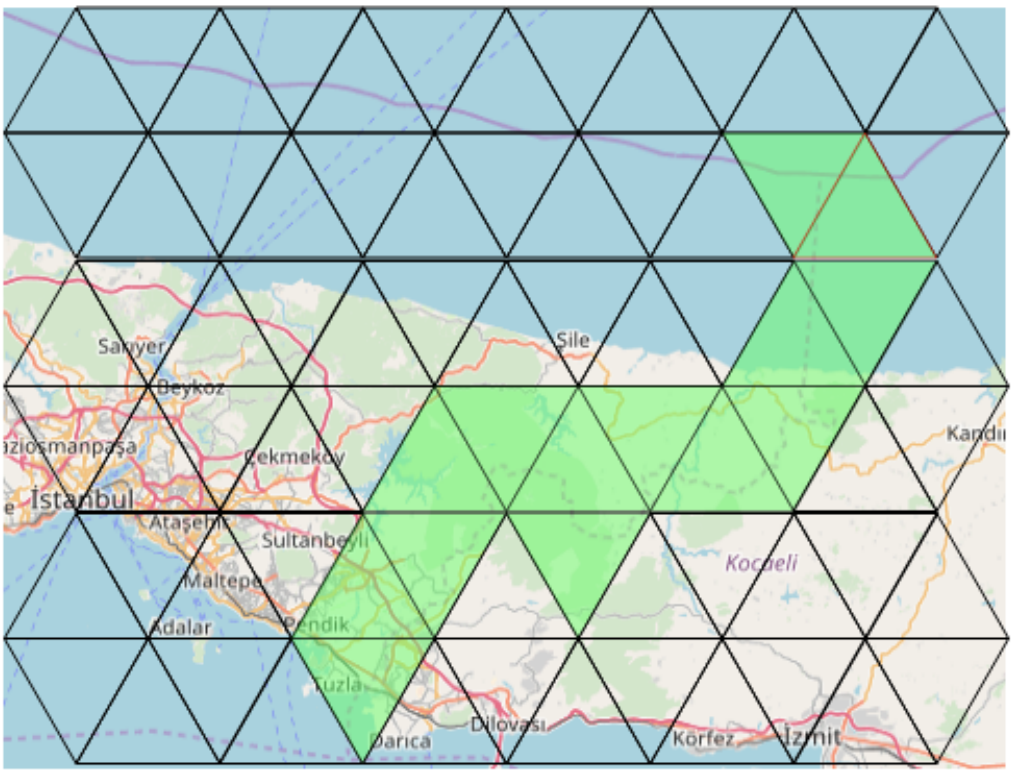

Figure 1.3. HTM index grid with partial trixels between two neighboring regions marked as green.

Interval skip list is a variation of the well-known data structure skip list. For our method, standard interval skip list is not sufficient hence a customized version is implemented. Besides HTM index, R-Tree indexes are stored in a serializable data structure which can be shared across cluster workers.

Streaming huge amount of data and distributing the individual blocks of stream among backing workers to classify points is a complex operation. In order to cope with this challenge, we use Apache Kafka cluster as a mediator layer between client and backing Apache Spark [5] workers. Kafka is one of the most scalable, robust and open-source messaging platform among various queuing solutions. Architecture of Kafka is organized around a few concepts such as topics, producers, consumers and brokers. Kafka topic is a structure that holds stream of records. Producers by analogy are clients writing messages to the topics and consumers read these messages from the topics. Lastly, brokers are workers inside the cluster and they keep topics or small portion of topic records which will be explained later. For each and every client that communicates with point classifier server should send messages and expect response in a specific format. On the other hand, for each client, a unique producer and a unique consumer Kafka topic are defined so that any client could pump data stream upon system through an exclusive channel. "ProducerRecord" object of Kafka is taken as a communication template for our program. In Figure 1.4, an overview of asynchronous communication between client and Kafka is depicted with input, output templates and example message content. Basically, the query input to our system is composed of a unique key and an array of latitude/longitude pairs so that the system responds the query with the same unique key and classification result of points which is an array of corresponding region IDs.

1.1. Contributions of Thesis

Spatial indexing on geographical data helps applications processing spatial data to achieve high performance on classification and query operations. RDBMS vendors provide built-in spatial index support for increasing response time of SQL join operations. However, RDBMS systems do not perform well in streaming data operations

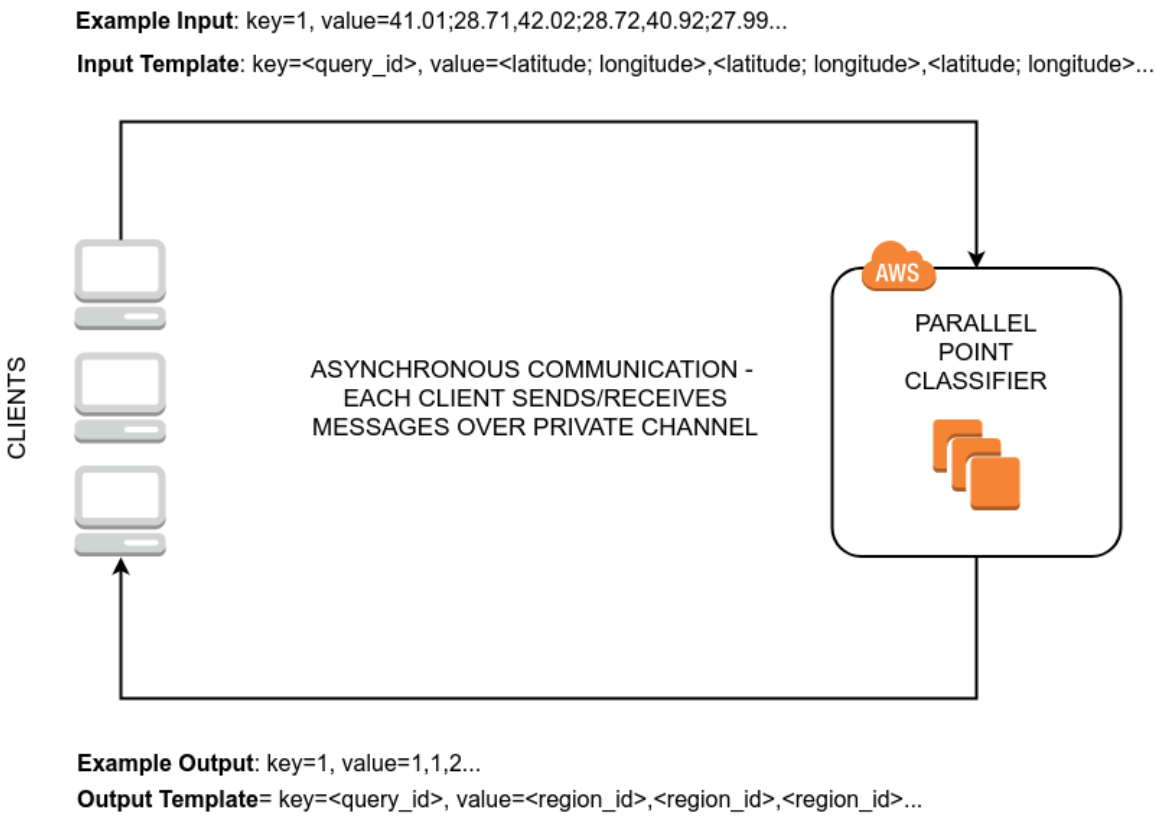

Figure 1.4. Application Input/Output example and template.

because of their disk I/O bounded nature and necessity to have data in a fixed model or schema.

The aim of this study is to develop an efficient parallel point classification mechanism using HTM and R-Tree indexes and allow streaming point data to be classified in a fast and scalable parallel execution environment. Scalability and accessibility of a system are significant requirements for being able to process this high volume of streaming data. In this context, programs that are working on Apache Spark platform perform well against high data volumes given that Spark is specialized to process data in-memory. What is more, the data to be processed is allowed to be schemaless unlike it is in RDBMS. By using Spark platform in our method, our approach outperforms conventional RDBMS based systems. In order to show performance improvement, we conduct tests specifically against Microsoft SQL Server which is a well-known RDBMS by using its built-in quad-tree based spatial index.

SQL Server has limitations on the aforementioned built-in index scheme besides having restrictions in processing data streams. Spatial index of SQL Server constructs a hierarchical decomposition of a surface with a four-level grid hierarchy. Each level decomposes the predecessor level so each-level contains a complete grid of successor levels. On a given level all grids have the same number of cells along columns and rows of each grid and each cell in a level are equal sized. Thus, a 4×4 grid produces 65,536 cells which is a hard limit for 4×4 configuration. Number of grid cells can be set 4×4, 8×8 or 16×16. Thus, it provides 32-bit index depth at most, which is less than 40 bit deep hash limit of the HTM indexing. As another constraint, SQL built-in index has a hard limit of 8192 index entries per geography object. These limitations of SQL built-in index cause less effective classification performance and decrease accuracy of results. Limitations on indexing and inappropriate nature for processing streaming data are main problems of RDBMS based systems. Hence, our method works well with streaming data and outperforms the Microsoft SQL Server built-in spatial index and HTM based indexing strategy employed on SQL Server. All in all, contributions of thesis can be summarized as follows.

- A parallel and scalable point classifier is implemented with HTM Indexing approach.
- Performance improvement is obtained on spatial join operation with a modified version of customized interval skip list data structure which stores regional HTM indexes.
- Partial trixel problem of HTM indexing strategy is resolved by exploiting supportive R-Tree index structure.
- Benchmarks are collected against Microsoft SQL Server 2012 built-in spatial index and Kondor et al. [1] HTM indexing approach on SQL Server.
- Performance metrics of parallel point classifier for Tweeter data are collected by using variable sized resource configurations on AWS Cloud.
- Benchmarks for parallel point classifier by applying snappy [6] compression algorithm to messages travelling on communication channel between query client and Kafka cluster that streams spatial data to be classified, are provided.

The organization of the remainder of this thesis is as follows.

Chapter 2 details background information for index and data structures and technologies used in our thesis scope. HTM index structure and interval skip list are analyzed. In addition, implementation details expressed as pseudocodes are given.

Next, relevant academic works are investigated through in Chapter 3. Research methods conducted by using HTM Indexing, Hadoop and Apache Spark are presented in this chapter.

Chapter 4 gets into more detail on our proposed method by providing more implementation details and explaining how platforms and layers interact each other, meanwhile giving information about flow of application.

Chapter 5 introduces the benchmarking strategies and provides results of our tests with different resource configurations and strategies. We also present comparison of our results with those of other approaches.

Chapter 6 concludes the thesis with a wrap-up of topic. Possible areas of research as extensions to our current approach are discussed and several ideas are given in this chapter.

2. BACKGROUND WORK

In this section, internals of HTM Indexing strategy and customized interval skip list data structure which is implemented as part of this thesis are explained. Implementation details and platform overviews related to Apache Kafka and Spark platforms are provided. Finally, a discussion on Amazon Cloud and EMR (Elastic Map Reduce) is provided superficially.

2.1. Hierarchical Triangular Mesh Internals

Spatial indexing has been a complex problem so that various geographical indexing strategies are proposed in research studies to store and access spatial data in most optimized way. Several database providers such as Microsoft SQL Server and MongoDB employ spatial index structures with built-in support. Besides the well-known database vendors, JTS topology suite library [7] provides geometry operations, algorithms in computational geometry and spatial index structures such as R-Tree and quad-tree. HTM indexing strategy that indexes a sphere which is employed in our study is created by Szalay and Grey [8] who are working for Microsoft Research. HTM indexing approach is applicable on both geographical objects on the Earth and celestial sphere which is an abstract sphere having an arbitrarily large radius taking Earth as center. Therefore, HTM indexing is well-suited on astronomical applications as well as spatial data processing applications such as classification of huge data generated by social networks like Twitter in our study and blockchain applications having geographical tags on transactions. The HTM indexing is initially linked to Microsoft SQL Server which is a relational database server within a study of Gray et al. [9].

HTM is a multilevel and recursive decomposition of the sphere. In order to build a quad-tree based HTM index, the strategy is to subdivide sphere into evenly sized triangles. The triangulation of sphere begins with an octahedron having eight faces. Edges of the octahedron are projected on sphere and four spherical triangles are generated on Southern hemisphere and the remaining four are on Northern hemisphere.

Initial level of index is 0 which corresponds to the octahedron having eight faces and can be subdivided repeatedly in a recursive manner. Number of triangles at a desired depth d can be calculated with the formula 2.1. HTM index structures starting from level 0 which embodies 8 triangles up to level 5 which has 8192 triangles are illustrated in Figure 2.1. In the limit, the subdivision operation recursively approaches to the sphere with finest detail.

$$\textbf{NumOfTriangles}(d) = 8 \times 4^{d-1} \tag{2.1}$$

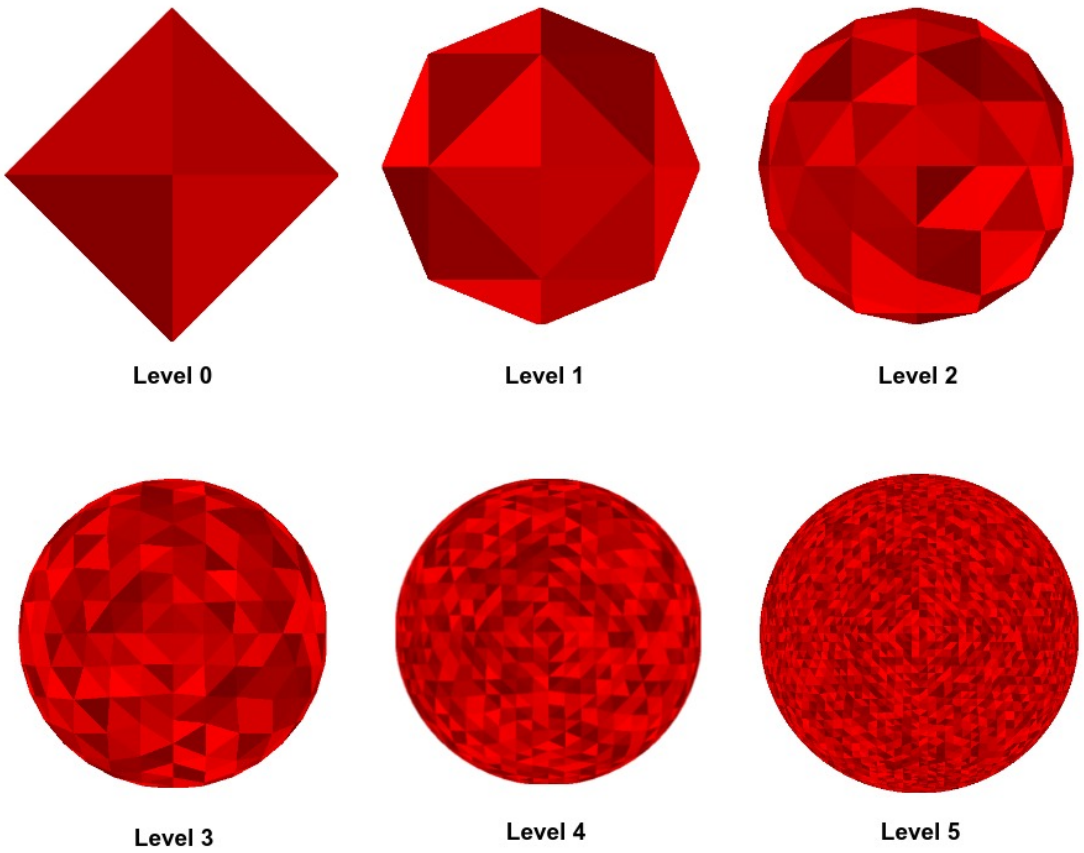

Figure 2.1. HTM index in ascending resolution level.

Triangles of HTM index which are also called as trixels and next hierarchical level is obtained by subdividing trixels into four triangles of nearly equal size. Subdivision operation could be repeated infinitely, but the practical limit is level 30 since a 64-bit integer can hold at most in level 30 trixel ID. Trixed ID is a unique identifier which maps into triangles at different resolution level. Trixel ID which can also be called as HTM ID, is an integer value encodes the depth and location of trixel ID. The encoding of regions or address points is provided by HTM library [10]. The HTM naming strategy is as follows. Level 0 trixels that forms the 8 faces of octahedron are named according to the hemispheres they belong to. Naming of trixels on Northern hemisphere has a prefix of N and the ones on Southern hemisphere has prefix of S. Initial four trixels on north has names of N0, N1, N2, N3 and similarly the ones on south has names S0, S1, S2, S3. As HTM is a quad-tree variant, each trixel has four children. The children of each trixel is named by appending 0, 1, 2, 3 numbers to the parent trixel name. The labeling implies that smaller trixels have longer labels and the length of the label indicates its HTM ID level. Encoding strategy of the HTM node names into integer HTM IDs is done by appending two bits for each level and initially encoding prefix N as 11 and S as 10. As an example N11 encodes as binary 110101 and decimal 53. Child trixel numbers are added as a decimal number between 0 and 3 to the name and binary values between 00 and 11 to the HTM ID. The ID corresponds to a 64-bit integer value from Java perspective. 64-bit integer can hold up to level 30 trixel IDs, but for practical purposes level 20 trixel IDs which corresponds to a precision around 25 meters are used within the latest HTM library and in our study. The whole sphere can be represented with $8 * 4^{20} = 8.796e + 12$ trixels (more than eight and three quarter billion) in level 20 resolution.

Trixels at each level can be represented with a name or an interval of descendant trixel names. As an example the four descendants of trixel with name N33 are N330, N331, N332, N333, form the decimal interval 252-255. By using interval representation, regions with variable sized trixels can be stated uniformly. Since trixels subdivide the sphere uniformly, any location is inside a single trixel so if we consider the resolution of level 20 trixels, HTM ID of a point is an accurate approximation to the real position. Strategy of mapping points to the trixels is exploited in classification

algorithm. Similar to the locations, regions can be expressed with trixels, but with a set of them intersecting with region. HTM library executes an algorithm to generate HTM ID intervals namely HTM index for a region. Algorithm starts with octahedron having eight level 0 trixels and recursively analyze unprocessed descendants and mark them with a specific tag. The trixels marked with inner tag are completely inside that region, the reject ones are outside the region and partial ones are on the frame of polygon. Inner trixels are added directly to the regional index and reject ones are discarded immediately. Partial trixels are split into four children which are processed further. Eventually the algorithm reaches out to a specified depth and trixels are tagged as inner or partial at desired level of the HTM index of region. In Figure 2.2, level 20 HTM ID intervals generated for randomly selected simple polygons are depicted.

HTM library supports three different coordinate systems. One of which is LatLon spherical coordinate system which takes prime meridian (Greenwich) as initial point and represents points with latitude and longitude parameters. The second one is an equatorial system which is a celestial coordinate system having right ascension and declination parameters (ra-dec) and which is mostly used by astronomers. The last one is a cartesian coordinate system which is based on unit vectors representing the axes of points, i.e., the unit vectors in the direction of the x, y and z define points.

2.2. Interval Skip List Internals

Skip list is a data structure that provides quick search within an ordered nodes sequence. In this study, we use a custom implementation of skip list to store intervals of HTM IDs instead of a single value on each node. Interval skip list allows fast search by preserving a hierarchy of linked subsequences of intervals with each successive interval skipping over less elements than the previous one as it is illustrated in Figure 2.3. Search operation starts in the sparsest interval subsequence until two consecutive intervals have been found. One of these intervals could be smaller and the other one could be larger than search key, or as a terminating condition it contains the key searched for. Via the linked hierarchy, these two elements link to elements of the next sparsest subsequence, where searching is continued until finally we are searching in

Polygon A	Polygon B	Polygon C
17372913139711 17372913074176	17372931457023 17372931440640	17372913729535 17372913664000
17372918841343 17372918317056	17372931538943 17372931522560	17372931489791 17372931162112
17372919037951 17372918972416	17372931670015 17372931653632	17372931751935 17372931555328
17372919365631 17372919103488	17372939812863 17372939567104	17372932472831 17372931817472
17372920479743 17372920414208	17372940140543 17372940075008	17372932997119 17372932538368
17372925919231 17372925657088	17372940238847 17372940206080	
17372926705663 17372926181376	17372940271615 17372940255232	
17372928409599 17372928344064	17372940304383 17372940288000	
17372930965503 17372930899968	17372940763135 17372940730368	
17372939354111 17372939288576	17372940795903 17372940779520	

Figure 2.2. Level 20 HTM ID intervals generated for randomly selected polygons.

the full sequence. There are two known approaches to skip over elements which are probabilistic [11] and deterministic [12]. Skip lists are formed with layers containing linked lists and arrays. Each node contains an array of reference to the nodes with larger values and structure contains ordered sequence of intervals. Each higher layer is sparser than the predecessors and skip over more nodes. Nodes residing in ith layer resides in $i+1th$ layer with a defined probability value p in probabilistic version. The generally used p values are $\frac{1}{2}$ and $\frac{1}{4}$. On average, each element appears in $\frac{1}{1-p}$ lists and the head element which is used as an initial dummy node appears in all lists. Totally skip lists contains $\log_{\frac{1}{p}} n$ lists on average.

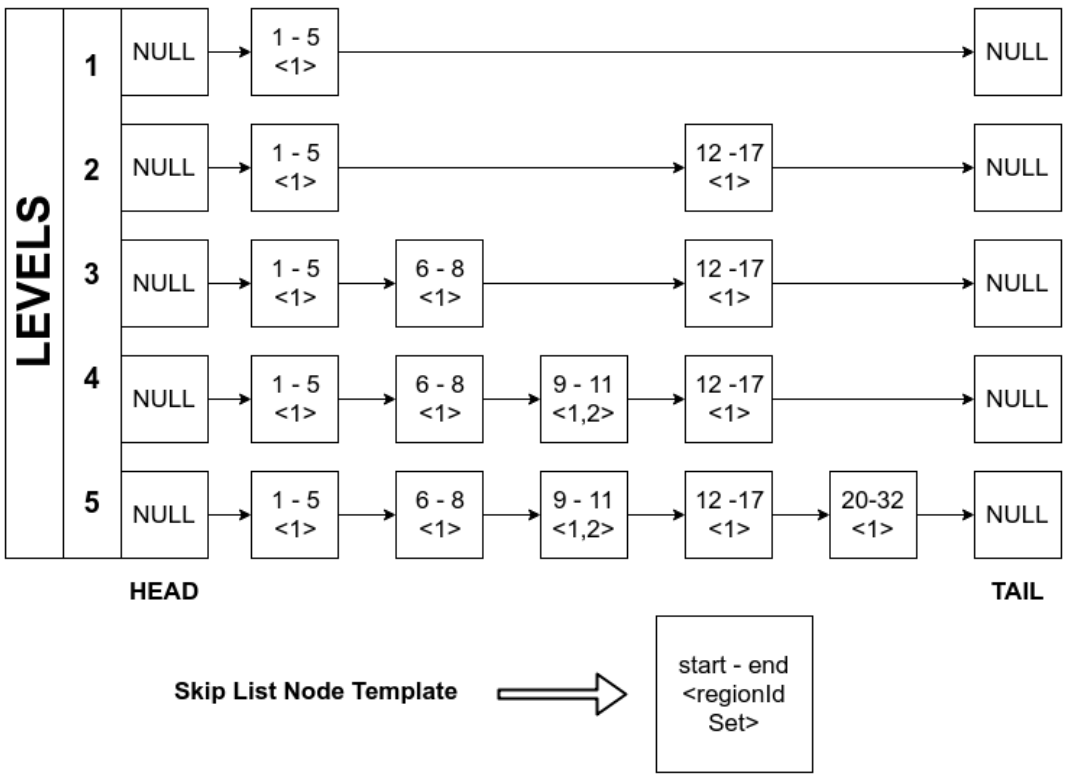

Figure 2.3. Interval Skip List Visual Representation.

Interval skip list implementation has significant advantages over conventional skip lists in terms of space and time as it contains fewer elements by keeping intervals. To be more accurate, HTM IDs given in Table 2.1 which represents a polygon covering Ankara (one of the densest regions in Turkey), correspond to 805306349 nodes in a traditional skip list. By modification into interval skip list the node count decreased

to 19. If we calculate the space allocation of the traditional skip list including the 805306349 nodes by taking into account the values kept in nodes that are of long data type, we reach out 805306349 * 8 bytes that corresponds to 6143 MB space. In contrast, interval skip list needs only 2 * 19 * 8 byte corresponds to 304 bytes space.

Table 2.1. Level 20 HTM ID intervals covering Ankara region which are stored inside interval skip list nodes.

HtmID Start	**HtmID End**
17338782973952	17338917191679
17338933968896	17338950746111
17338984300544	17339051409407
17360257810432	17360274587647
17360291364864	17360324919295
17360408805376	17360425582591
17360475914240	17360492691455
17368847745024	17368914853887
17368931631104	17369116180479
17369720160256	17369787269119
17369820823552	17369837600767
17369854377984	17369871155199
17369887932416	17369921486847
17544941404160	17544958181375
17566416240640	17566449795071
17566466572288	17566483349503
17566516903936	17566533681151
17566651121664	17566667898879
17575006175232	17575022952447

Before conducting analysis on interval skip list search algorithm complexity, we briefly explain how search operation is working and give pseudocode of interval search in Figure 2.4.

```
Algorithm1 intervalSkipListSearch(skipList, key)
Input: List skipList in interval skip list form,
       Key to search in skip list in number type.
Output: Node x of interval skip list contains key inside interval or NIL if not found.

result := Clone of skipList→head
currentNode := skipList→head
level := head→level (Loop count is number of longest level)
while level >= 0 do
  while interval upper bound of currentNode link to next one at level position > key
  do
    currentNode := currentNode→links[level]
  end while
  result→links[level] := currentNode
  level := level - 1
end while
currentNode := result[1]
if interval upper bound of currentNode >= key and interval lower bound of currentNode
<= key then
  return currentNode
else
  return NIL
end if
```

Figure 2.4. Interval skip list search algorithm.

Interval skip list operation of searching for a specific element starts at the top element in the uppermost list. Then, it proceeds horizontally until the desired element is found inside the interval of a node or greater than the end of interval. If the target is inside the interval of node, that means the target node is found and the found interval is returned. Otherwise, the operation is repeated with cursor moving down to the list one level below and continue to compare interval values with target value until it ends up with final node of bottom list. Expected number of searching an element in each piece of linked list is at most $\frac{1}{p}$. If we consider the expected total cost of an overall

search, that becomes $\frac{\log_{\frac{1}{p}} n}{p}$ which is $\mathcal{O}(\log n)$, in case p is constant. [11] Skip list bears trade off between storage costs and search costs associated to the value of constant p.

Average search complexity for both interval skip list and skip list is $\mathcal{O}(\log n)$,, but interval skip list provides great operational efficiency as it contains far fewer nodes than normal one. If we consider p as $\frac{1}{2}$ and calculate number of search steps with formula $\frac{\log_{\frac{1}{p}} n}{p}$ is 59,169 for skip list and 8,643 for interval skip lists which is approximately seventh one for list covering Ankara.

Apart from search algorithm of interval skip list, we put an effort on algorithm that is adding new intervals for specific region IDs. Challenging part of adding interval is that conflicting intervals arising from partial trixels occur between neighbouring regions. While adding intervals in skip list we take into account all edge-cases and presented pseudocode of corresponding algorithm in Figure 2.5.

```
Algorithm2 intervalSkipListAddInterval(regionId, start, end)
Input: Id of the region for corresponding interval as number,
       Lower bound of interval as 64-bit(long) number,
       Upper bound of interval as 64-bit(long) number.
Output: Void, side-effect is that interval skip list is modified.

if start > end then
  swap start and end
end if
Node lowerNode := search(start)
Node upperNode := search(end)
if upper bound of lowerNode >= start and lower bound of lowerNode <= start and
upper bound of upperNode >= end and lower bound of upperNode <= end then
  if lowerNode = upperNode and regionId is not in idSet of lowerNode then
    Node currenNode := lowerNode
    if currentNode→start = start and currentNode→end = end then
      currentNode→idSet.add(regionId)
```

Figure 2.5. Interval skip list algorithm to add new interval.

```
        else if currentNode→end = end then
            delete interval of currentNode
            add intersected interval between start and end with new regionId
            add non-intersected interval between currentNode→start and start - 1
        else if currentNode→start = start then
            delete interval of currentNode
            add intersected interval between start and end with new regionId
            add non-intersected interval between end + 1 and currentNode→end
        else
            delete interval of currentNode
            add intersected interval between start and end with new regionId
            add non-intersected interval between currentNode→start and start - 1
            add non-intersected interval between end + 1 and currentNode→end
        end if
    else
        delete lowerNode and upperNode
        if lowerNode and upperNode idSets contains only regionId then
            add interval between lowerNode→start and upperNode→end
        else if lowerNode idSet contains not only regionId then
            add interval between lowerNode→start and upperNode→start - 1 with regionId
            add interval between upperNode→start and upperNode→end without regionId
        else if upperNode idSet contains not only regionId then
            add interval between lowerNode→end + 1 and upperNode→end with regionId
            add interval between lowerNode→start and lowerNode→end without regionId
        else
            add interval between start and lowerNode→end with regionId and lowerNode
            regionIds
            add interval between lowerNode→end + 1 and upperNode→start - 1 with re-
            gionId
            add interval between upperNode→start and end with regionId and upperNode
            regionIds
        end if
    end if
else if upper bound of lowerNode >= start and lower bound of lowerNode <= start
then
    if lowerNode idSet contains regionId then
        lowerNode→end := end
```

Figure 2.5. Interval skip list algorithm to add new interval.(cont.)

```
    else if lowerNode→start = start then
      add regionId to idSet of lowerNode
      add interval between lowerNode→end + 1 and end with regionId
    else
      lowerNode→end := start - 1
      add interval between start and end with regionId and idSet of lowerNode
    end if
  else if upper bound of upperNode >= start and lower bound of upperNode <= start
  then
    if upperNode idSet contains regionId then
      upperNode→start := start
    else if upperNode→end = end then
      add regionId to idSet of upperNode
      add interval between start and upperNode→start - 1 with regionId
    else
      upperNode→start := end + 1
      add interval between start and end with regionId and idSet of upperNode
    end if
  else
    add interval between start and end with regionId
  end if
```

Figure 2.5. Interval skip list algorithm to add new interval.(cont.)

2.3. Overview of Apache Kafka

Apache Kafka can be defined as a distributed streaming platform that works as a mid-layer to publish and subscribe to streams of records in a way similar to message queue. Besides being a messaging platform, it possesses features such as storing records in a fault-tolerant way and provides user-defined processing capability over records. Kafka is beneficial in two use-cases. First is working as a real-time streaming pipeline between applications and second one is being a layer that enables an application to react of transform real-time streams. Kafka serves our goal in building-up a real-time spatial data processing tool as being a geo-location data pipeline between clients sending spatial queries and backing Spark workers processing these queries.

Kafka platform has a lot of features to gain applications scalability and robustness while reacting real-time events. Platform operates as a cluster either on private infrastructure or on cloud and it can be distributed over multiple data centers as well. Kafka provides APIs to establish communication between clients and cluster easily. As a communication message template records which consist of a value, key and timestamp are used. These records are separately stored and identified with topics. So-called ProducerAPI enables a client to publish records to topics of their selection. ConsumerAPI allows a client to subscribe topics and process the records. Besides ConsumerAPI, Kafka provides StreamsAPI that is specialized for transforming an application into a stream processor which gathers input streams from multiple sources specifically topics and produce an output stream to a single or multiple topics after transforming records. Last core API of Kafka is ConnectorAPI is that the one allows building up reusable components either producing or consuming records by connecting topics to existing systems or applications. In our approach we are using Spark Streaming integration for Kafka 0.10 version which provides parallelism an one-to-one correspondence between Kafka and Spark partitions and access to metadata over ConsumerAPI.

It is mentioned that Kafka provides stream of records with a well-known abstraction called topics. Topic can be explained to be a category in which stream records are published or subscribed. Kafka topics allow single or multiple subscribers to consume written records by design. Each and every topic maintains a partitioned structure called log. Partitions are ordered and immutable sequence of records and each record inside a partition is represented by a sequential unique id number called the offset. Kafka cluster is able to persist published records for a configurable period of retention. By these offsets consumers can trace partition and reprocess past data or skip forward again to consume or process current records. Topics may have multiple partitions span along different cluster machines so that arbitrary amount of data can be assigned to a partition. In our application, each topic has more partition than number of Spark executors so that we can achieve a totally parallelized architecture. Partitions are replicated across cluster servers in a configurable number for fault tolerance. Consumers and Producers are first-class users of Kafka topics. Producers simply publish records to a topic and even partition of their choice. If they do not specify a partition then parti-

tion assignment of records can be done in round-robin for load balancing. Consumers are receivers of published records in a way that one consumer in each subscribing consumer group receives the record. Consumer group label is a configuration to determine delivery strategy so that if all consumer instances belong to same group, records would be load balanced evenly, In this sense, Kafka provides order inside each partition, but inter partition ordering is not guaranteed. We exploit from consumer group labeling in distributing incoming query messages evenly on processors as well.

Kafka has two different streaming models queuing and publish-subscribe. In queuing model, a consumer pool may read from a single queue and records are load balanced among consumers, in contrast publish-subscribe model is based on broadcasting record to all consumers. Kafka generalizes these two models with the consumer group concept. Consumer group allows user to divide up processing with a queue, on the other hand publish-subscribe allows to broadcast messages to multiple consumer groups. Kafka can be used as a message broker for various reasons such as buffering messages or decouple processing from data producers with providing better throughput, partitioning, fault-tolerance and message compression. In case data streaming of an application is at high level, the bottleneck of overall system becomes network bandwidth not disk or CPU. In order to improve throughput of applications with untangling network bottleneck, GZIP [13], snappy and LZ4 compression protocols supported transparently by Kafka are promising. Moreover, Kafka allows end-to-end batch compression which leads to very high compression ratios as compressing batch of messages together which would be decompressed in consumer end consequently.

2.4. Overview of Apache Spark

Apache Spark is an in-memory, scalable and general purpose cluster computing system. It provides an execution engine for processing general execution graphs. Applications can adopt framework and leverage from platform capabilities with high level APIs provided in Scala, Java and Python languages.

Spark programming brings several concepts into stage such as RDDs and Datasets. Main abstraction that is given by Spark API is Resilient Distributed Dataset. It contains collections of objects with partitioning all over the Spark cluster nodes. With the RDD abstraction, parallel processing of huge collections become practical and optimized. RDDs can be created based on a HDFS, HBase file or parallelizing an existing Scala collection in driver. They can be persisted in shared memory which is to be used across parallel operations. RDDs have support on two operations such as transformation and action. Transformation is to create a dataset from an existing dataset. Action is to retrieve values from a computation task on any dataset to driver program. Map can be classified as a transformation which traverses each element of a dataset and produces a RDD keeping result set. On the other hand, reduce can be classified into action type which is aggregation of elements in a RDD using a given function and provides result to driver program. By the transformations and actions possible bottlenecks could occur on driver node arising from streaming large dataset are prevented by map and reduce tasks executed merely on cluster workers. Transformation functions are lazily evaluated during program flow. A transformation such as map is actually computed at the time a terminating action is triggered which returns stream result to driver program. Spark provides persist or cache operations for transformed RDDs to prevent re-computation cost and provide quick access for multiple queries.

Another fundamental concept of Spark programming is shared variables. Shared variables can be used in parallel execution steps. Normally Spark ships a copy for every variable in functions that are executed within scope of tasks. In some cases, variables are essentially shared between tasks and driver program or across some tasks. There exists two shared variables provided such that accumulators like sums and counters and broadcast variables which caches values in memory on every cluster node. In order to run up a Spark program, SparkContext object should be created initially. SparkContext is initiated with a SparkConf object which holds all configuration parameters regarding to application and cluster. With the SparkContext object program can access cluster. We mainly used standalone cluster mode so cluster manager of our program is Spark. In Figure 2.4 the associations between Spark components and where our point classifier and driver reside are clearly shown.

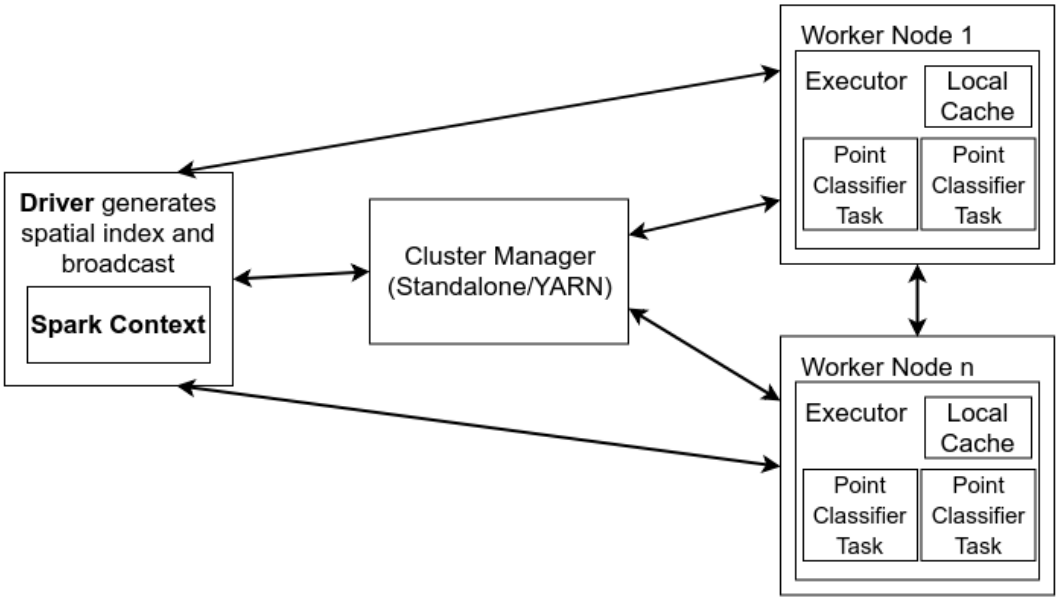

Figure 2.6. Overall Spark architecture of parallel point classifier.

2.5. Overview of Amazon Cloud and EMR

AWS as a vendor offers reliable, scalable and easy to access and use cloud services publicly. The services that we use are Amazon EC2 (Elastic Cloud) and Amazon EMR, additionally we created Amazon CloudFoundry scripts for Kafka cluster setup.

Amazon EC2 instances are used as an infrastructure resource in classification application and exploit from Amazon CloudFoundry scripts in setting up cluster environment. Amazon EMR is a service that provides popular big data and data processing distributed frameworks such as Hadoop, Apache Spark, Flink, HBase and Presto. It allows researchers or commercial users to process large amount of data across dynamically scalable EC2 instances. Also, EMR can interact with data stores provided by Amazon such as S3 and DynamoDB.

3. RELATED WORK

Spatial indexing has been a significant problem of GISs (Geographic Information Systems) and astronomical applications. Various schemes to efficiently index spherical data are proposed such as HTM, HEALPix, R-Tree, R*-Tree, quad-tree and Geohash [14].

Szalay [8] from Microsoft Research Center offers to apply HTM indexing strategy on sphere. The proposed method is to divide the Earth surface into spherical triangles of similarly sized shapes. Basically, HTM is a quad-tree based index which provide results from search operations at different resolutions. Their proposed indexing scheme provides basis for addressing and fast look-ups. HTM provides very efficient geospatial indexing structure appropriate for relational databases in case corresponding data have an inherent location on the Earth surface.HTM algorithms create triangulation over regions that are represented in polygons on the Earth by transforming regions of sphere into unique identifiers. These IDs can be used to address an area or to use for indexing. HTM trixels of a given depth covering an arbitrary region are called that region's HTM cover. We are using these set of IDs to classify points into regions. While doing the classification task, we generate HTM ID mapping of specific coordinates with the help of HTM algorithms. Computing a list of trixels that cover a region is a quite complex task and in classification perspective resulting trixels may have pitfalls. These trixels can be either inside or partially inside of a region so that this yields to a false-positive classification result in case query point falls inside a partial trixel. In our approach, we solve the problems that arise from partial trixels by adopting a supportive indexing strategy.

Budavári and Szalay propose an extension scheme on their HTM indexing approach [10]. Within the scope of their study, they focus on application of HTM on astronomical observations, especially most significant area of it which is named as sky coverage. They present algorithms to manipulate shapes composed of arbitrarily sized and complex polygons on celestial sphere. HTM indexing tools are enhanced in terms

of performance and capabilities like compliance with the mentioned shapes. Toolbox is simplified to easily interact within runtime of SQL server and an internal representation of various regional shapes like convex, polygon and circle are provided. Pixelization schemes benefit from the improved representation. An optimal set of HTM pairs of an administrative region can be computed by their improved library in considerably less amount of time compared to earlier library versions. Moreover, C# language and .NET framework is used in implementation which means that the provided libraries become platform and OS independent. This makes the solution very compact and easy to integrate in applications hence we use this improved version of HTM toolbox while building-up our method architecture.

Kondor et al. [1] propose a method and framework where they adopt HTM indexing to enable HTM indexing to be applicable for spatial filtering and spatial join on Microsoft SQL Server. They collect approximately a billion tweets from Twitter social network data stream, specifically from 51 states of USA which is used for benchmarks and verification. In their paper, they present results of index generation time for 50 continental states of United States using two different depths of SQL Server grid levels like 8×8 and 16×16 and three different resolution of HTM index such as 12, 14 and 16. If Microsoft SQL Server indexing scheme is considered, according to the test results it is evident that increasing the resolution of index makes things worse as there is a strict limit of 8192 cells on number of index rows for SQL server. Therefore, SQL server spatial index does not perform well for complex polygons. It performs well only in case there are large number of relatively small and simple polygons. Benchmarks are collected on a 16-core single instance RDBMS with 96 GB memory. The results of HTM based filtering shows that their approach outperforms Microsoft SQL Server ten times faster and spatial joins against SQL Server performs roughly a factor of hundred times faster. Their study shows that HTM indexing on SQL server utilizes in spatial data operations.

Gorski [4] offers another index structure for pixelization of data on the Earth. The indexing scheme originally designed to address the processing and analyzing huge volume of astronomical data needs. In addition, it is exploited on cosmic microwave

background experiments. In the study, Gorski provides the constraints and requirements of HEALPix hierarchical indexing framework and explain the details of designed hierarchical indexing concept.

In addition to HTM based indexing solutions, there are several research studies that investigate spatial operations leveraging from big data processing capabilities of different scalable platforms. One of these approaches of processing large-scale spatial data on Cloud environment is proposed by You and Zhang [15]. Their primary objective is to compare two widely used platforms namely Apache Spark and Cloudera Impala to process spatial join task on Cloud environment. Recently, general approach while working with spatial data or other large datasets, is using in-memory analytic platforms like Apache Spark on commodity server environments. Underlying reason for research and enterprise community to divert from Hadoop based spatial processing systems such as SpatialHadoop [16], HadoopGIS [17] and ESRI Spatial Framework for Hadoop [18] is performance weaknesses of Hadoop based platforms against Spark. Even if Hadoop based systems provide high scalability, they output intermediate results onto disk which causes excessive amount of disk I/O and performance degradation. On the other hand, Spark is more efficient in cutting unnecessary disk operations so that You preferred to use Spark for their study. They make benchmarks on spatial join queries utilizing R-Tree index on cloud environment composed of 10 g2.2xlarge Amazon EC2 instances. Two different datasets are used in experiments, one of them is New York Taxi pickup locations includes around 170M points and 10M global species occurrences of GPS points. They conclude with higher performance rate against Hadoop based platforms by conducting experiments with Spark and Impala which is similar to our conclusions.

Yu and Wu [19] build a platform which has a basis of Apache Spark as well. They introduce a cluster computing framework for spatial data operations called GeoSpark [20]. GeoSpark has a layered structure and these layers are Apache Spark Layer, Spatial RDD Layer and Spatial Query Processing Layer. Apache Spark Layer is responsible for providing fundamental Spark functionality such as RDD operations and data storing operations on disk. They create an extension of Spark RDD structure which is specialized for spatial and geometrical objects and called Spatial Resilient Distributed

Dataset (SRDD). The last one called Spatial Query Processing Layer processes spatial query and algorithms such as spatial range and join with backing SRDDs. Logic behind leveraging Spatial RDDs is to partition data across machines using a grid structure in which each grid is assigned to a machine. In contrast, in scope of our method we distribute incoming data stream in Kafka level into classifier processes instead of collecting whole input data inside a single partitioned RDD. There is a decomposition point between our approach and Yu's in spatial index structure-wise as they use quad-tree and R-Tree indexes but we use HTM with supporting and optional R-Tree index. Their framework generates R-Tree or quad-tree indexes in grid cell level based on the decision taken in query runtime depending on trade-off between efficiency impact of indexing according to the quantity of objects inside a specific grid and indexing overhead of the mentioned grid. As another optimization, the platform puts effort to minimize data shuffling to achieve good performance. There exists different amount of shuffling overhead in applications given in paper such as spatial join, spatial range, aggregation or co-location and for each case it is optimized or nearly-optimized. The reason behind founding an architecturally complex and layered platform is that in the current state they make GeoSpark evolve to an enterprise platform. The experiments within scope are conducted on Amazon cluster EC2 machines with 16 r3.2xlarge slaves and 1 c4.2xlarge with Apache Spark set-up on these machines. Tests are done with a dataset used in TIGER project [21] which contains all cities, lakes and boundaries of US in rectangular shapes corresponding to approximately 65 GB of data. They make a comparison with SpatialHadoop [16] by applying R-Tree and quad-tree indexes over regions separately. In conclusion, the results show that GeoSpark outperforms its Map-Reduce [22] based counterparts like SpatialHadoop in terms of query response time, memory and CPU usage metrics.

3.1. Summary

In table 3.1 contributions of our thesis and techniques that are used in our study are compared against other studies that are also related with spatial indexing and spatial data processing. The table can be accepted as summary of related work section.

The critical separation points between studies are indexing techniques and platforms used. Szalay proposes to apply HTM indexing on sphere his study. Kondor extends HTM framework with a SQL server compatible version and develop an algorithm which separates partial trixels from regional index to increase false-positive detection rate and improve spatial join performance against SQL server built-in index. The studies related to HTM structure has application on RDBMS, in contrary our study adopts HTM indexing into Spark processing environment which is capable of working under large volume of streaming data. In that sense our method becomes a scalable solution moreover solves partial indexing problem by combining two index types. Also, huge amount of streaming data can be classified faster than RDBMS and in parallel with Spark. You, Zhang and Yu, Wu exploit from high-performance distributed computing platforms such as Spark and Impala and develop methods in spatial data operations, but they take advantage of different indexing strategies such as quad-tree and R-Tree exclusively, from our study.

Table 3.1. Comparison of research studies for spatial indexing and point classification against our dissertation.

	HTM	R-Tree	quad-tree	Streaming	Spark	RDBMS
Our thesis	✓	✓	-	✓	✓	-
Kondor [1]	✓	-	✓	-	-	✓
Szalay [8]	✓	-	-	-	-	✓
You and Zhang [15]	-	✓	-	-	✓	-
Yu and Wu [19]	-	✓	✓	-	✓	-

4. METHODOLOGY OF PARALLEL POINT CLASSIFICATION

4.1. Data Collection and Data Cleaning

Tweeter is one of the most valuable data sources since source data is gathered from society directly. Besides the content of tweets, bunch of meta data such as geographic location and relations within social network are provided as a side set. Tweeter platform broadcasts tweets via a streaming interface to be exploited for research and commercial purposes. Tweeter streaming API delivers data to connected clients over a very long lived HTTP request. HTTP is an application protocol for communication of distributed and hypermedia information systems living in the World Wide Web. Clients that establish a connection to Streaming API should form a HTTP request, consume the resulting stream as long as it does not experience server or network errors and parse the response content incrementally. Tweeter servers hold the Streaming API connections open indefinitely as long as it is cut by a client or not having server malfunctions or failures. Tweeter delivers tweet data in JSON format in a way that each client could easily implement scripts to parse, analyze and modify them. JSON is a lightweight and human-readable data-interchange format. Streaming JSON object accommodates high amount of information. The most important part of data that we concerned is tagged with "geo" keyword and the valuable geographic location information is reside under this tag. In Figure 4.1 an example of an anonymized geo-tagged JSON object picked from Tweeter public stream data, for which redundant fields are filtered out.

In order to establish connections and deliver live stream, libraries in different languages are provided. We use Tweepy [23] library written in Python language. Also, we implement scripts which are used to store data with region and date based archiving format and to post process validity of collected data. Tweeter provides regional filtering option for streaming data and this feature is transparently available on Tweepy library.

```
{
    'id_str':'1000000000000000000',
    'geo':{
        'type':'Point',
        'coordinates':[
            38.4499254,
            27.21017361
        ]
    },
    'user': {...},
    'place': {...},
    'created_at':'Sun Jun 10 12:51:02 +0000 2018',
    'text':"..."
}
```

Figure 4.1. Tweeter Stream response JSON object.

Istanbul, Izmir, Ankara, Kocaeli and Eskisehir are densest cities of Turkey so that for verification purpose data is collected from these areas. Bounding boxes approximately covering these areas are defined for filtering out tweets collected via Tweeter stream. However, Tweeter stream could provide incorrect geo-tags which point to places out of these regions. Therefore, we need to post process collected tweets in order to eliminate data discrepancies. After collecting and storing tweets, another Python script is run to traverse all tweets and check their geocode if it points to correct city by reverse geocoding. Reverse geocoding is the process of back coding of a point location in latitude, longitude to a readable address or region name. We use Nominatim [24] which is the search engine serving OpenStreetMaps data via RESTful API [25] publicly. RESTful implies a web service implemented in accordance with REST service architectural design principles and constraints. In Figure 4.2 the HTTP GET request to reverse geocode a point in Ankara region and incoming response are shown. GET type of HTTP method requests a representation of a resource from server.

Reverse geocoding improves the effectiveness of our test set by purifying Tweeter data. In Table 4.1 ratio of useful data streamed for regions that we are concerned about is presented.

GET REQUEST: https://nominatim.openstreetmap.org/reverse?format=json&lat=38.9566025&lon=35.24322

RESPONSE:


```
{
  "place_id": "109855931",
  "licence": "Data OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright",
  "osm_type": "way",
  "osm_id": "169217006",
  "lat": "38.9867281",
  "lon": "35.2888521",
  "address": {
    "suburb": "Eskiomerler Mahallesi",
    "village": "Eski Omerler",
    "county": "Kocasinan",
    "state": "Central Anatolia Region",
    "postcode": "38090",
    "country": "Turkey",
    "country_code": "tr"
  },
  "display_name": "Eskiomerler Mahallesi, Eski Omerler, Kocasinan, Kayseri, Central Anatolia Region, 38090, Turkey",
  "boundingbox": [
    "38.9504582",
    "38.9868876",
    "35.2464268",
    "35.3431183"
  ]
}
```


Figure 4.2. Reverse geocoding response of Nominatim contains OpenStreetMapData.

According to the results, it is obvious that without data cleaning, accuracy of the system would be effected dramatically because of the false positive tweets that would be included inside the verification set of all regions.

Table 4.1. Ratio of location correctness for Tweeter data collected via Tweeter public stream across regions.

	Istanbul	Ankara	Izmir	Kocaeli	Eskisehir
Data Correctness	%71.6	%65.2	%77.4	%69.88	%70.2

4.2. Mechanism

Parallel point classification method described in this thesis has dependency on Apache Spark and Apache Kafka platforms which are the main components boosting our approach in establishing a scalable and a robust architecture. As a result of dependency on Kafka and Spark, we heavily use their open-source Java and Python libraries. Additionally, essential parts of HTM library are re-implemented in Java to enable point classifier to work on Spark environment. Since our approach is based on a complex mechanism and has dependency on the aforementioned platforms, in architectural perspective we come up with a layered structure. There are various steps in classification flow and these steps are explained in detail throughout upcoming parts.

First step is to obtain necessary GeoJSON files to use for HTM Index generation. These GeoJSON files for administrative regions of Turkey are loaded from OpenStreetMaps [26] which is a source providing free map data under open license. Map data is given in several optional formats such as well-known text (WKT), GeoJSON and poly. We prefer the most flexible JSON based GeoJSON format to achieve ease of conversion into Java objects. The GeoJSON files loaded from OpenStreetMap system are processed with Spherical Converter executable which depends on HTM library toolbox in generating HTM ID pairs of complex polygons. These HTM ID pairs are representing a set of trixels covering that specific polygon in level 20 depth which is the wisely selected depth based on benchmarks performance and optimized for internal floating point calculations in latest (3.1.2) version of HTM library. The

generation of HTM index pairs has two steps, initially GeoJSON files are converted into MultiGeometry objects defined with format stated in GeoJSON specifications [27] and implemented by GeoJSON.Net library [28]. MultiGeometry object is assumed to contain a single MultiPolygon object completely covering administrative region. Then, for each Polygon given under MultiPolygon object, HTM library parser method is triggered to generate HTM ID pairs as a list. To generate a list of pairs, BNF grammar defined by the toolbox is used to represent spherical shapes for HTM index generation. BNF is a notation technique to define context-free grammars. It can be used to describe syntax of languages in computing such as communication protocols, programming languages and document formats. For each Polygon object under MultiPolygon of region, program produces a statement as specified in HTM library with format of BNF grammar given in Figure 4.3. As a result, program generates a generic JSON object including region name, region id, list of HTM ID boundaries for each polygon. Finally it dumps data into a JSON file which is used to generate interval skip list of the processing region in the next step of execution.

Polygon Specification → POLY LATLON <latitude longitude> 3+
or POLY CARTESIAN <x y z> 3+
or POLY J2000 <ra dec> 3+
<...> 3+ implies 3 or more instances of the element specified inside braces
LATLON, CARTESIAN and J2000 implies coordinate system type

Figure 4.3. Polygon specification.

The next step of our approach is based on Apache Kafka cluster running on AWS. Kafka cluster contains a Zookeeper instance and dynamic number of broker instances. Amazon CloudFormation scripts are created to set up Kafka Cluster which contains variable number of broker instances with all necessary networking configuration to establish communication between cluster broker nodes and Zookeeper node. AWS CloudFormation provides a common script language that allows to describe and manage all infrastructure resources for Cloud environment. According to the stream volume expectation, Kafka cluster can have 3, 5 or 10 brokers with selected EC2 instance

type within CloudFormation script. Besides networking and instance configurations, security configurations for cluster and VPC that hosts cluster are specified within these scripts. Kafka cluster stands between client and point classifier executors running on Spark cluster at the lower level. As mentioned earlier, each Kafka topic holds configurable number of partition in accordance with load expectation of client by our design. Spark cluster executors communicate with Kafka brokers via Kafka client. Group id, which is a configuration value assigned by executors that consume stream of records, is identical for each executor which yields executors to open up multiple connections to Kafka brokers and retrieve even load from partitions on Kafka brokers. In Figure 4.4 the visual representation of how architectural layers of system interact with each other is shown.

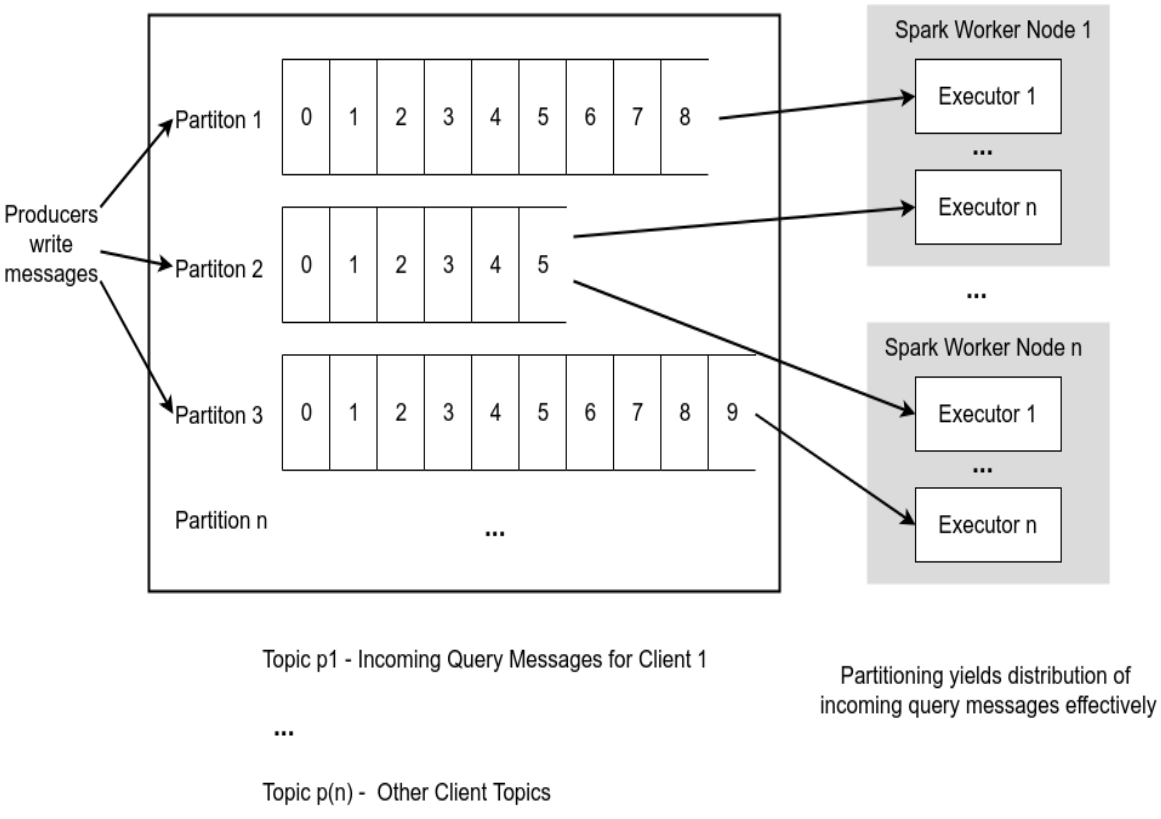

Figure 4.4. Kafka topic partitioning and Spark executors processing data stream.

Third part of our method works on Apache Spark cluster which hosts and run point classification tasks. Spark platform has vast amount of configurable components

and provides a complex yet easy to adopt framework. The program is capable of running on AWS EMR with both Spark Standalone and Apache YARN (Hadoop NextGen) cluster managers. For our mechanism, we select Standalone architecture that works in master/slave mode on AWS cloud or local set-ups. In Spark architecture there are two types of nodes such as master and slave nodes. Master node is the resource manager for the Spark Standalone cluster and allocates CPU, memory and disk resources among Spark applications by evaluating reports received from Worker nodes about resources. The allocated resources are used for running Spark driver and executors running on Workers. Slaves are the nodes hosting driver and executors.

Program executing by Spark driver initializes with generation of interval skip lists for each specific region. There exists a JSON file accessed by point classifier executable JAR which stores all HTM ID pairs of concerned regions. The file is used to generate an interval skip list out of all HTM index pairs. After generation of interval skip list, it is serialized and sent to executors wherein streaming coordinates are classified. Another set of JSON files hold GeoJSON objects that represents regions in polygon. The GeoJSON files are used to generate R-Tree index over each region which is optional feature. In case R-Tree indexes are generated, region ids mapping to R-Tree indexes are kept inside a Java map object in program context. R-Tree indexes take part as a supportive index which resolves classification problem that arises because of partial trixels between neighbouring regions. In case R-Tree index option is disabled, partial trixels cause slight decrease of classification accuracy. Effect of enabling supportive R-Tree index is shown within experiments chapter.

In this chapter, R-Tree indexing strategy is briefly explained without opening a new section since it is supportive index used as a part of our architecture. The main reasons why we chose not to assign a new chapter for deeper discussion of R-Tree are that it is a well-known and broadly used index type and there are enough resources to examine mathematical details and algorithms under the hood for R-Tree itself and for its variations such as Hilbert R-Tree [29] and R*-Tree [30]. R-Tree stems from rectangle tree since the idea behind tree structure is to group nearby objects and represent these regions with a minimum bounding rectangle in the next higher level of

tree. It is obvious that each MBR describes a single object in leaf level which is the lowest level. R-Tree is a balanced tree very similar to B-Tree which is broadly used data structure for indexing and storing data in disk in RDBMS. Like with most data structures searching, more specifically spatial operations such as nearest neighbour and intersection, is efficient. While searching a target point or area inside indexed 2D space, MBRs on nodes are used to decide whether sub-tree contains target object or not. The key difficulty of R-Tree lies under keeping MBRs with less overlapping and empty space so that search efficiency does not decrease. The aforementioned R-Tree variants all aim to improve the way how MBRs and the tree itself are built. We did not implement R-Tree to use in our program, instead we use JTS library implementation which is quite efficient and written in Java.

As the latest step of approach, executors begin streaming batches of records that contain coordinates throughout Kafka direct stream channels after regional indexes generated. In order to distribute incoming stream to the executors evenly, Kafka topics are configured by setting partition count configuration parameter as a value exceeding number of executors times number of consumer direct streams per executor. For the sake of improved utilization and fault-tolerance while consuming stream data, we introduce an extra dimension by batching from multiple partitions in executor level beyond task level parallelization. Multiple Kafka direct stream objects, which are called as DStream (Discretized Stream) of Spark library, are created per executor and data streams from these channels are concatenated by union operation of Spark context before classification operation. DStream is an abstraction of Spark Streaming and delivers a continuous stream of data received from sources like Kafka or TCP sockets. DStream contains a series of RDDs covering a certain time interval that can be changed according to the batch duration configured in streaming context. Any transformation or action applied on DStreams are reflected upon underlying RDDs by Spark engine thus programs operate with higher level API should not concern implementation details. As it is stated before, we are mapping partitions evenly on executors so that DStreams contains records received from specific partitions in executor perspective. Kafka ConsumerAPI configuration parameter called "LocationStrategies" holds the client strategy. We set "LocationStrategies.PreferConsistent" as configured strategy

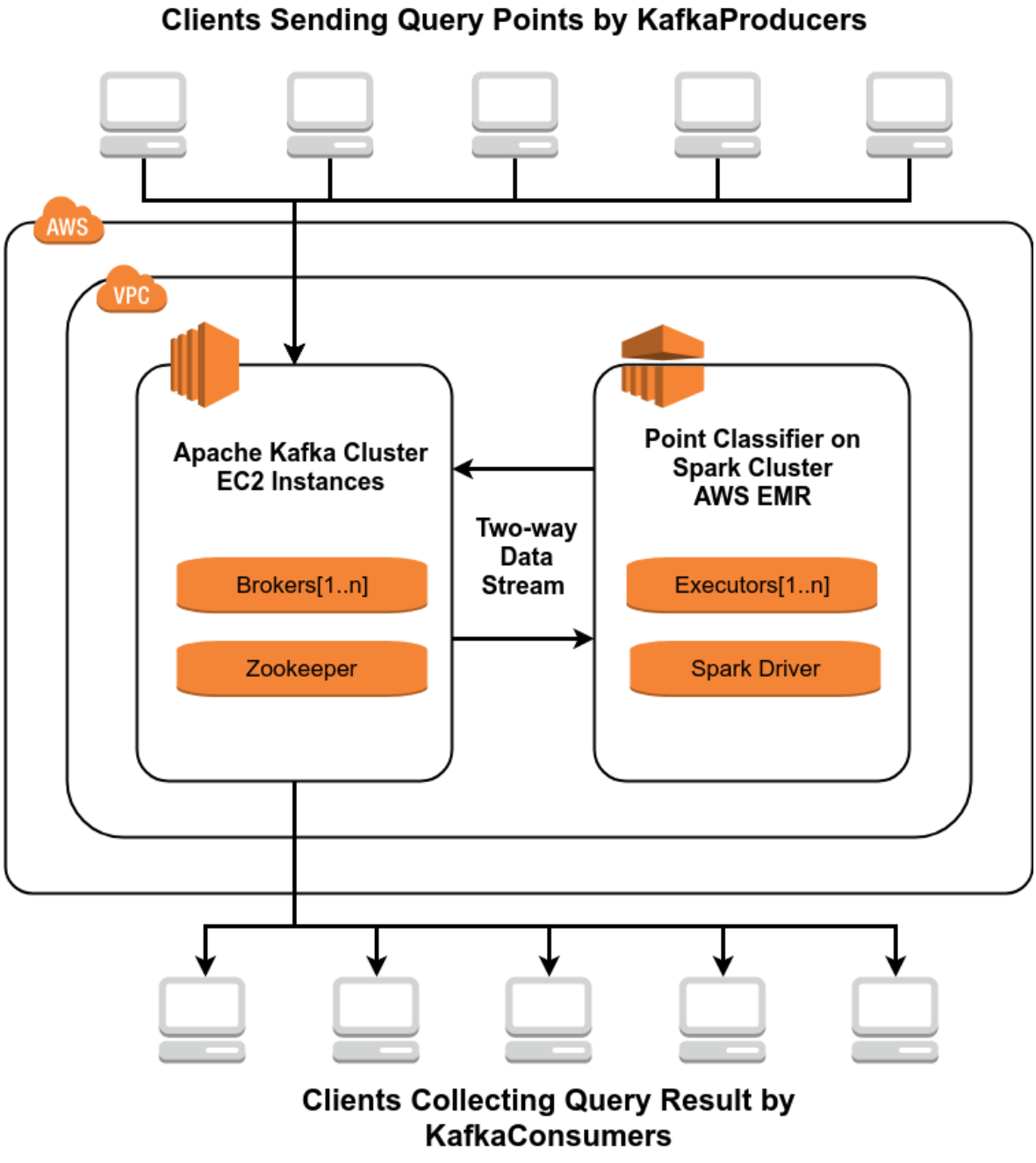

Figure 4.5. End-to-end process scheme of parallel point classifier with all components.

for client to distribute traffic on available executors. Moreover, Kafka consumer clients are created and cached on executors since clients pre-fetch and buffer stream messages so that these mentioned strategies lead to achievement of optimum performance. In Figure 4.5 the overall architecture and process steps are expressed.

```
Algorithm3 classifyPoints(record)
Input: record provided within Spark DStream which is created from live
       data of Apache Kafka stream of an arbitrary topic, namely client channel.
Output: Void, side-effect is response message sent via Kafka which contains
       classification results.
String consumerTopic := record→ _1
Integer queryId := record→ _2
String[] pointArray := record→ _3
for each point in pointArray do
  split point with "," character
  double longitude := splitPoint[0]
  double latitude := splitPoint[1]
  long pointHTMId := compute point HTM id with longitude and latitude
  search regionIdSet inside interval skip list with pointHTMId
  if regionIdSet is not found then
    append empty string to classificationResultString
  else if regionIdSet contains more than 1 element then
    for each region in regionIdSet do
      if point within region by R-Tree index then
        append regionId string to classificationResultString
        break
      end if
    end for
  else
    append regionId string to classificationResultString
  end if
end for
create response to send client with producerTopic queryId and classificationResultString
send response in ProducerRecord template with Kafka client
```

Figure 4.6. Point classification algorithm.

The detailed pseudocode of classification algorithm is given in Figure 4.6. In pseudocode we assumed that a point classifier is expecting messages of query holding multiple coordinates inside.

4.3. Software Package Details

All pieces of program are provided in a bundle within Github which is publicly accessible via the link given in references section [31]. Under the root folder there are several modules so that each one is accountable for execution of different tasks.

"TweeterGeoStream" module is responsible for data collection via Tweeter Streaming API to use for experiments. The module is written in Python by using Tweepy [23] library that allows application to connect Tweeter Streaming API with location filtering option. There are two set of Python scripts inside this module, one of which contains scripts that are responsible for streaming, shaping and storing tweets. Other scripts are responsible for post processing the stored tweet data. While post processing tweets, we take advantage of reverse geocoding to validate if a particular tweet belongs to the region it attached with. For this specific reason, post processing scripts are sending GET requests to Nominatim servers which present OpenStreetMap data. Since Nominatim is an open-source tool, it has usage policy as follows that it limits incoming requests to prevent commercial or individual users abusing their system. That's why our script complies with their policy such a way that putting throttle control while sending HTTP requests.

SphericalConverter module is the one which is used to generate JSON files including HTM index pairs from complex geographical region specifications with GeoJSON format. Later on point classifier process running on Spark cluster would use these output JSON files. Module has a dependency on HTM library created by Szalay and Budavari [10].

We provide two Java projects "htmServer" and "pointStreamClassifier" as separate modules, however they are serving the same goal to be run on Spark within JVM

environment providing necessary data structures and logic to parallel classification of points into regions. A JAR file called "point.stream.classifier-1.0.jar" is provided under target directory and the both projects have maven support so that they can be rebuilt easily after overriding default configuration and adding modifications. Maven is an Apache project and a software project and dependency management tool. The same JAR contains stream generators (parallel and sequential) which can generate coordinates in a specified bounding box.

KafkaCluster module contains CloudFormation scripts which allow users to set up and start Kafka cluster with desired configuration by setting instance types, account specific security configurations and networking configurations. AWS has an informative console screen where users can monitor cluster creation steps and instance status information. In Appendix A, a CloudFormation script to setup and run a small-sized Kafka cluster is provided.

5. EXPERIMENTS AND RESULTS DISCUSSION

Due to the fact that our program is running on Amazon Cloud, it is tightly coupled with the Amazon infrastructure and network status. Therefore, performance of the program can easily be affected by the hosting AWS region status, networking vulnerabilities and other infrastructure issues. In order to minimize the Cloud related problems, the method is tested several times with different configurations and average of benchmark results are presented. In order to show improvements of our approach, the method is compared against results of point classification method of Kondor et al.which is built on Microsoft SQL Server and SQL Server built-in spatial index. At first, administrative regions of Istanbul, Ankara, Izmir, Eskisehir and Kocaeli from GADM (Database of Global Administrative Areas) [32] are loaded into SQL Server. Afterwards spatial index with 16×16 grid cell configuration which corresponds to approximately level 16 HTM index, is created and performance metrics of classification task are collected.

Table 5.1. Classification accuracy of indexing strategies measured for regions over Turkey.

Indexing Strategy	Istanbul	Ankara	Izmir	Kocaeli	Eskisehir
SQL Server geography 16*16	%89	%90	%88	%73	%86
HTM Level 20	%98.2	%98.6	%97.9	% 96.3	%95.2
HTM Level 20 with R-Tree support	%99.1	%99.3	%97.9	%97.4	%96.1

Average latency values of parallel point classifier against changed set of infrastructure level and application level configurations are reported in the next sections. Two infrastructure setups having different horizontal scaling level are prepared. The number of Kafka producer and consumer topic partitions is tuned up for each configuration in accordance with estimation of executor count in Spark environment.

5.1. Results of HTM Indexing on Apache Spark with small infrastructure configuration

First cloud configuration that we run benchmarks on, includes a Kafka cluster with 5 brokers and a Spark cluster with 8 executors each having 6GB RAM and 4 virtual CPU cores running on the AWS cloud. The EC2 instance type that is used for benchmarks on AWS is m4.xlarge which has 4 virtual cores and 16GB memory. The benchmarks of Microsoft SQL server are run on a 16 core machine with 92GB memory. Single instance SQL server (with version 2012) is used on the infrastructure. According to the results, it is obvious that SQL server's built-in index cannot classify large number of records since it cannot complete query for 1 billion records.

Table 5.2. Average classification time of HTM on Apache Spark with snappy compression against SQL server built-in and HTM indexing.

Indexing Strategy	300k[sec]	500k[sec]	1M[sec]	5M[sec]	1G[sec]
SQL Server geography 16 * 16	120	210	520	2110	-
SQL Server HTM Level 16 [1]	6	7	10	40	967
SQL Server HTM Level 14 [1]	6	8	11	49	1099
HTM Level 20 with R-Tree support on Apache Spark	0.89	1.25	2.07	6.976	678
HTM Level 20 on Apache Spark	0.86	1.2	1.99	6.880	670

Apache Kafka provides compression support for communication channel between producer and consumer. In our benchmarks, we provide results of queries that are run with snappy compression support. The outcome of results show that compression helps to increase total processing time in case the query contains large set of records.

Classifying batches with small number of records compression causes increase in query time since there is a trade-off between time loss of CPU time and network I/O time. When the number of batch is large, query time is cut off since smaller sized network packets are conveyed faster and it yields decrease in overall query time.

Table 5.3. Average classification time of HTM on Apache Spark without using Kafka message compression.

Indexing Strategy	300k[sec]	500k[sec]	1M[sec]	5M[sec]	1G[sec]
HTM Level 20 with R-Tree support on Apache Spark	0.664	0.995	1.718	6.509	752
HTM Level 20 on Apache Spark	0.670	0.987	1.7	6.440	745

5.2. Results of HTM Indexing on Apache Spark with large infrastructure configuration

Other cloud configuration which is relatively larger than first one includes Kafka cluster with 5 brokers and Spark cluster with 8 executors each having allocated 12GB RAM and 6 virtual CPU cores running on AWS cloud. The EC2 instance type that is used for benchmarks on AWS is m4.2xlarge which has 8 virtual cores and 32GB memory. Since result comparison with SQL server dependent approaches are reported in the previous section, the same results are not provided in Table 5.4 and Table 5.5. Only results of experiments of our proposed method running on Spark are presented.

Table 5.4. Average classification latency values of HTM on Apache Spark with snappy compression against SQL server built-in and HTM indexing.

Indexing Strategy	300k[sec]	500k[sec]	1M[sec]	5M[sec]	1G[sec]
HTM Level 20 with R-Tree support on Apache Spark	0.78	1.2	1.99	6.02	520
HTM Level 20 on Apache Spark	0.8	1.21	1.8	5.997	518

Table 5.5. Average classification latency values of HTM on Apache Spark without compression.

Indexing Strategy	300k[sec]	500k[sec]	1M[sec]	5M[sec]	1G[sec]
HTM Level 20 with R-Tree support on Apache Spark	0.664	0.995	1.718	6	660
HTM Level 20 on Apache Spark	0.670	0.987	1.7	5.98	655

6. CONCLUSION

Experiment results show that HTM indexing has significant performance improvement over quad-tree based spatial index provided by Microsoft SQL Server with built-in support. Besides, utilizing R-Tree index within JTS library [7] as an inhibitor against partial trixel problem, we improve the accuracy of classification results. Above and beyond, overall performance is much more increased since we scale up parallel classification program on AWS Cloud Environment. As a further improvement to our approach, different kind of spatial index structures like quad-tree provided by JTS library and a practically efficient and improved variant of R-Tree called Priority R-Tree [33] can be ported in our framework. Moreover, current implementation may be improved by extending Spark RDD in a way to create and store HTM index on RDD partitions distributed over Spark cluster machines which would be beneficial while working with larger regional data. Also, with our current method, tests can be repeated with larger set of regions and using new set of tweets collected over those larger region set. Experimenting and adopting the improved methods may lead to increased accuracy in situations like points falling into partial trixel and may also yield better containment testing results. Another future work topic may be the implementation of spatial range based queries such as nearest-neighbour search with HTM indexing as an extension to the current approach which implements spatial join based query operations.

REFERENCES

1. Kondor, D., L. Dobos, I. Csabai, A. Bodor, G. Vattay, T. Budavári and A. S. Szalay, "Efficient classification of billions of points into complex geographic regions using hierarchical triangular mesh", *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, 2014.

2. Kunszt, P., A. S. Szalay, I. Csabai and A. Thakar, "The Indexing of the SDSS Science Archive", *Veillet, D. Crabtree, (ASP Conference series)*, 2000.

3. Guttman, A., "R-Trees: A Dynamic Index Structure for Spatial Searching", *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 1984.

4. Górski, K. M., E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke and M. Bartelmann, "HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere", *Astrophys.J.*, 2005.

5. Zaharia, M., M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, "Spark: Cluster Computing with Working Sets", *HotCloud*, 2010.

6. *Snappy compression codec library*, `http://google.github.io/snappy/`, accessed at May 2018.

7. *GeoTools JTS*, 2018, `http://docs.geotools.org/latest/userguide/library/jts/index.html`, accessed at May 2018.

8. Szalay, A., J. Gray, G. Fekete, P. Kunszt, P. Kukol and A. Thakar, "Indexing the Sphere with the Hierarchical Triangular Mesh", *Microsoft Research Technical Report*, 2005.

9. Gray, J., A. Szalay, G. Fekete, M. O'Mullane, W. Nieto-Santisteban, A. Thakar

and G. Heber, "There Goes the Neighborhood: Relational Algebra for Spatial Data Search", *Microsoft Technical Report*, 2004.

10. Budavari, T., A. S. Szalay and G. Fekete, "Searchable Sky Coverage of Astronomical Observations: Footprints and Exposures", *Publications of the Astronomical Society of the Pacific*, 2010.

11. Pugh, W., "Skip lists: A probabilistic alternative to balanced trees", *Communications of the ACM.*, 1990.

12. Munrot, J. I., T. Papadakis and R. Sedgewick, "Deterministic skip lists.", *Proceeadings of the third annual ACM-SIAM symposium on Discrete algorithms (SODA '92)*, 1992.

13. Deutsch, P., *GZIP file format specification version 4.3*, `https://www.rfc-editor.org/rfc/rfc1952.txt`, accessed at May 2018.

14. *Geohash*, `http://en.wikipedia.org/wiki/Geohash`, accessed at May 2018.

15. Simin, Y., J. Zhang and L. Gruenwald, "Large-Scale Spatial Join Query Processing in Cloud", *Proceedings of the 2015 31st IEEE International Conference on Data Engineering Workshops (ICDEW)*, 2015.

16. Eldawy, A. and M. Mokbel, "A demonstration of Spatialhadoop: an efficient mapreduce framework for spatial data.", *Proc. VLDB*, 2013.

17. A.Aji, "Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce", *Proc. VLDB*, 2013.

18. *ESRI Hadoop*, 2018, `http://esri.github.io/gis-tools-for-hadoop/`, accessed at May 2018.

19. Yu, J., J. Wu and M. Sarwat, "Geospark: A cluster computing framework for processing large-scale spatial data", *Proceedings of the 23rd SIGSPATIAL Inter-*

national Conference on Advances in Geographic Information Systems, 2015.

20. *GeoSpark*, 2015, `https://geospark.co`, accessed at May 2018.

21. Bureau, U. S. C., *TIGER - Topologically integrated geographic encoding and referencing.*, 2018, `https://www.census.gov/geo/maps-data/data/tiger.html`.

22. Dean, J. and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Communications of the ACM*, 2008.

23. *Tweepy Library*, 2018, `http://www.tweepy.org`, accessed at May 2018.

24. *Nominatim OpenStreetMaps search engine documentation*, 2018, `https://nominatim.openstreetmap.org/`, accessed at May 2018.

25. *Nominatim RESTful API URL for reverse geocoding*, 2018, `https://nominatim.openstreetmap.org/reverse`, accessed at June 2018.

26. *Open Source World Wide Map Data*, 2018, `https://www.openstreetmap.org`, accessed at May 2018.

27. Butler, H., M. Daly, A. Doyle, S. Gillies and T. Schaub, *The GeoJSON Format Specification*, https://tools.ietf.org/html/rfc7946, 2016.

28. *.Net library for GeoJSON types & corresponding Json.Net (de)serializers*, `https://github.com/GeoJSON-Net/GeoJSON.Net`, accessed at May 2018.

29. Kamel, I. and C. Faloutsos, "Hilbert R-tree: An improved R-tree using fractals", *Proceedings of VLDB conference*, 1994.

30. Beckmann, N., H. P. Kriegel, R. Schneider and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles", *Proceedings of the 1990 ACM SIGMOD international conference on Management of data - SIGMOD '90*, 1990.

31. *HTM Parallel Point Classifier Source Code*, 2018, `https://github.com/sanvert/HTMPointClassifier`, accessed at June 2018.

32. *GADM Database of Global Administrative Areas*, 2018, `http://gadm.org`, accessed at May 2018.

33. Arge, L., M. Berg, H. Haverkort and K. Yi, "The priority R-tree: A practically efficient and worst-case optimal R-tree", *ACM Transactions on Algorithms (TALG)*, 2008.

APPENDIX A: KAFKA CLOUDFORMATION SCRIPT

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "AWS CloudFormation Template: This template creates 1 Kafka Zookeeper and 3 Kafka
      Broker instances",
  "Parameters": {
    "EBSVolumeSize": {
      "Type": "Number",
      "Default": "10"
    },
    "EBSType": {
      "Type": "String",
      "Default": "gp2",
      "AllowedValues": [
        "standard",
        "gp2",
        "io1"
      ]
    },
    "EBSDeviceName": {
      "Type": "String",
      "Default": "/dev/xvda",
      "AllowedValues": [
        "/dev/xvda",
        "/dev/sda1"
      ]
    },
    "VPCId": {
      "Type": "AWS::EC2::VPC::Id",
      "Description": "VpcId of existing Virtual Private Cloud (VPC)"
    },
    "SubnetID": {
      "Description": "Subnet ID in your Virtual Private Cloud (VPC) for Zookeepers and Kafka Brokers
          ",
      "Type": "AWS::EC2::Subnet::Id"
    },
    "SubnetCIDR": {
      "Description": "CIDR Block for Private Subnet where Zookeepers and Kafka Brokers will be
          deployed. ex:10.0.1.0/24",
      "Type": "String",
      "MinLength": "9",
      "MaxLength": "18",
      "Default": "10.0.1.0/24",
      "AllowedPattern": "[a-zA-Z0-9]+\\..+"
    },
    "RemoteAccessCIDR": {
      "Description": "IP CIDR from which you are likely to SSH into. You can add rules later by
          modifying the created security groups e.g. 54.32.98.160/32.",
      "Type": "String",
      "MinLength": "9",
      "MaxLength": "18",
      "Default": "0.0.0.0/0",
      "AllowedPattern": "(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})",
      "ConstraintDescription": "must be a valid CIDR range of the form x.x.x.x/x."
    },
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.

```
    "KeyName": {
      "Description": "Name of an existing EC2 KeyPair to enable SSH access to the instances",
      "Type": "String",
      "Type": "AWS::EC2::KeyPair::KeyName",
      "MinLength": "1",
      "MaxLength": "64",
      "AllowedPattern": "[-_ a-zA-Z0-9]*",
      "ConstraintDescription": "can contain only alphanumeric characters, spaces, dashes and
          underscores."
    },
    "KafkaDownloadURL": {
      "Description": "URL to download kafka tarball",
      "Type": "String",
      "Default": "http://ftp.nluug.nl/internet/apache/kafka/1.1.0/kafka_2.11-1.1.0.tgz"
    },
    "KafkaVersion": {
      "Description": "Version used",
      "Type": "String",
      "Default": "kafka_2.11-1.1.0"
    },
    "ZkeeperServerInstanceType": {
      "Description": "Zookeeper EC2 instance type",
      "Type": "String",
      "Default": "t2.micro",
      "AllowedValues": [
        "r3.large",
        "t2.nano",
        "t2.small",
        "t2.micro"
      ],
      "ConstraintDescription": "must be a valid EC2 instance type."
    },
    "KafkaServerInstanceType": {
      "Description": "KafkaBroker EC2 instance type",
      "Type": "String",
      "Default": "t2.small",
      "AllowedValues": [
        "r3.xlarge",
        "t2.small"
      ],
      "ConstraintDescription": "must be a valid EC2 instance type."
    }
  },
  "Mappings": {
    "AmiId": {
      "eu-central-1": {
        "AMI": "ami-875042eb"
      },
      "sa-east-1": {
        "AMI": "ami-27b3094b"
      },
      "ap-northeast-1": {
        "AMI": "ami-0dd8f963"
      },
      "eu-west-1": {
        "AMI": "ami-38c09341"
      },
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
    "us-east-1": {
      "AMI": "ami-2051294a"
    },
    "us-west-1": {
      "AMI": "ami-d1315fb1"
    },
    "us-west-2": {
      "AMI": "ami-775e4f16"
    },
    "ap-southeast-2": {
      "AMI": "ami-e0c19f83"
    },
    "ap-southeast-1": {
      "AMI": "ami-3f03c55c"
    }
  }
},
"Resources": {
  "RootRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "ec2.amazonaws.com"
              ]
            },
            "Action": [
              "sts:AssumeRole"
            ]
          }
        ]
      },
      "Path": "/",
      "Policies": [
        {
          "PolicyName": "root",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": "*",
                "Resource": "*"
              }
            ]
          }
        }
      ]
    }
  },
  "IAMProfile": {
    "Type": "AWS::IAM::InstanceProfile",
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
    "Properties": {
      "Path": "/",
      "Roles": [
        {
          "Ref": "RootRole"
        }
      ]
    }
  },
  "KafkaServerSecurityGroup":{
    "Type":"AWS::EC2::SecurityGroup",
    "Properties":{
      "GroupDescription":"KafkaServer SG to allow access to/from Kafka Brokers and Zookeepers.",
      "VpcId":{
        "Ref":"VPCId"
      },
      "SecurityGroupIngress":[
        {
          "IpProtocol":"tcp",
          "FromPort":"0",
          "ToPort":"65535",
          "CidrIp":{
            "Ref":"SubnetCIDR"
          }
        }
      ]
    }
  },
  "SSHSecurityGroup": {
    "Type": "AWS::EC2::SecurityGroup",
    "Properties": {
      "GroupDescription": "Enable SSH access",
      "VpcId": {
        "Ref": "VPCId"
      },
      "SecurityGroupIngress": [{
        "IpProtocol": "tcp",
        "FromPort": "22",
        "ToPort": "22",
        "CidrIp": {
          "Ref": "RemoteAccessCIDR"
        }
      }]
    }
  },
  "KafkaZookeeperInstance": {
    "Type": "AWS::EC2::Instance",
    "Metadata": {
      "region": {
        "Ref": "AWS::Region"
      },
      "stack_name": {
        "Ref": "AWS::StackName"
      },
      "AWS::CloudFormation::Init": {
        "config": {}}
    },
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
"Properties": {
  "ImageId": {
    "Fn::FindInMap": [
      "AmiId",
      {
        "Ref": "AWS::Region"
      },
      "AMI"
    ]
  },
  "InstanceType": {
    "Ref": "ZkeeperServerInstanceType"
  },
  "IamInstanceProfile": {
    "Ref": "IAMProfile"
  },
  "KeyName": {
    "Ref": "KeyName"
  },
  "BlockDeviceMappings": [
    {
      "DeviceName": {
        "Ref": "EBSDeviceName"
      },
      "Ebs": {
        "VolumeSize": {
          "Ref": "EBSVolumeSize"
        },
        "VolumeType": {
          "Ref": "EBSType"
        }
      }
    }
  ],
  "NetworkInterfaces":[
    {
      "GroupSet":[
        {
          "Ref":"KafkaServerSecurityGroup"
        },
        {
          "Ref":"SSHSecurityGroup"
        }
      ],
      "AssociatePublicIpAddress": "true",
      "DeleteOnTermination": "true",
      "DeviceIndex":"0",
      "SubnetId": { "Ref" : "SubnetID" }
    }
  ],
  "Tags":[
    {
      "Key":"Name",
      "Value":"ZooKeeperInPublicSubnets"
    }
  ],
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
    "UserData": {
      "Fn::Base64": {
        "Fn::Join": [
          "",
          [
            "#!/bin/bash -v\n",
            "\nyum -y install unzip java wget\n",
            "\ncurl \"https://s3.amazonaws.com/aws-cli/awscli-bundle.zip\" -o \"awscli-bundle.
                zip\"\n",
            "unzip awscli-bundle.zip\n",
            "./awscli-bundle/install -b ~/bin/aws\n",
            "export PATH=~/bin:$PATH\n",
            "export ZOOKEEPER_ID=1\n",
            "su - root -c 'mkdir -p /app/kafka'\n",
            "wget ",
            {
              "Ref": "KafkaDownloadURL"
            },
            " -P /app\n",
            "export file=`echo ",
            {
              "Ref": "KafkaDownloadURL"
            },
            " | rev | cut -f1 -d '/' | rev`\n",
            "tar -zxvf /app/$file -C /app/kafka\n",
            "su - root -c 'mkdir -p /tmp/zookeeper'\n",
            "echo $ZOOKEEPER_ID > /tmp/zookeeper/myid\n",
            "echo \"initLimit=5\nsyncLimit=2\" >> /app/kafka/kafka_2.11-1.1.0/config/zookeeper.
                properties\n",
            "sed -i 's/Defaults    requiretty/Defaults    !requiretty/g' /etc/sudoers\n",
            "sed -i 's/KAFKA_JMX_OPTS=\"-D/KAFKA_JMX_OPTS=\"-Djava.net.preferIPv4Stack=true -D/g
                ' /app/kafka/kafka_2.11-1.1.0/bin/kafka-run-class.sh\n",
            "su - root -c 'nohup /app/kafka/kafka_2.11-1.1.0/bin/zookeeper-server-start.sh /app/
                kafka/kafka_2.11-1.1.0/config/zookeeper.properties > /dev/null 2>&1 &'\n"
          ]
        ]
      }
    }
  }
},
"KafkaBrokerInstance": {
  "Type": "AWS::EC2::Instance",
  "Metadata": {
    "region": {
      "Ref": "AWS::Region"
    },
    "stack_name": {
      "Ref": "AWS::StackName"
    },
    "AWS::CloudFormation::Init": {
      "config": {
      }
    }
  },
  "DependsOn": "KafkaZookeeperInstance",
  "Properties": {
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
"ImageId" : {
  "Fn::FindInMap" : [
    "AmiId",
    {
      "Ref" : "AWS::Region"
    },
    "AMI"
  ]
},
"InstanceType" : {
  "Ref" : "KafkaServerInstanceType"
},
"IamInstanceProfile" : {
  "Ref" : "IAMProfile"
},
"KeyName" : {
  "Ref" : "KeyName"
},
"BlockDeviceMappings" : [
  {
    "DeviceName" : {
      "Ref" : "EBSDeviceName"
    },
    "Ebs" : {
      "VolumeSize" : {
        "Ref" : "EBSVolumeSize"
      },
      "VolumeType" : {
        "Ref" : "EBSType"
      }
    }
  }
],
"NetworkInterfaces":[
  {
    "GroupSet":[
      {
        "Ref":"KafkaServerSecurityGroup"
      },
      {
        "Ref":"SSHSecurityGroup"
      }
    ],
    "AssociatePublicIpAddress" : "true",
    "DeleteOnTermination" : "true",
    "DeviceIndex":"0",
    "SubnetId" : { "Ref" : "SubnetID" }
  }
],
"Tags":[
  {
    "Key":"Name",
    "Value":"KafkaBrokerInPublicSubnets"
  }
],
"UserData" : {
  "Fn::Base64" : {
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
        "Fn::Join": [
          "",
          [
            "#!/bin/bash -v\n",
            "\nyum -y install unzip java wget\n",
            "\ncurl \"https://s3.amazonaws.com/aws-cli/awscli-bundle.zip\" -o \"awscli-bundle.
                zip\"\n",
            "unzip awscli-bundle.zip\n",
            "./awscli-bundle/install -b ~/bin/aws\n",
            "export PATH=~/bin:$PATH\n",
            "export ZOOKEEPER_ID=1\n",
            "su - root -c 'mkdir -p /app/kafka'\n",
            "wget ",
            {
              "Ref": "KafkaDownloadURL"
            },
            " -P /app\n",
            "export file=`echo ",
            {
              "Ref": "KafkaDownloadURL"
            },
            " | rev | cut -f1 -d '/' | rev`\n",
            "tar -zxvf /app/$file -C /app/kafka\n",
            "su - root -c 'mkdir -p /tmp/kafka-logs'\n",
            "sed -i.bak \"s/zookeeper.connect=.*/zookeeper.connect=",
            {
              "Fn::GetAtt": [
                "KafkaZookeeperInstance",
                "PrivateIp"
              ]
            },
            ":2181/g\" /app/kafka/kafka_2.11-1.1.0/config/server.properties\n",
            "sed -i.bak \"s/#advertised.listeners=.*/advertised.listeners=PLAINTEXT://`curl
                http://169.254.169.254/latest/meta-data/public-hostname`:9092/g\" /app/
                kafka/kafka_2.11-1.1.0/config/server.properties\n",
            "sed -i.bak \"s/broker.id=.*/broker.id=0/g\" /app/kafka/kafka_2.11-1.1.0/config/
                server.properties\n",
            "sed -i 's/Defaults    requiretty/Defaults    !requiretty/g' /etc/sudoers\n",
            "sed -i 's/KAFKA_JMX_OPTS=\"-D/KAFKA_JMX_OPTS=\"-Djava.net.preferIPv4Stack=true -D/g
                ' /app/kafka/kafka_2.11-1.1.0/bin/kafka-run-class.sh\n",
            "su - root -c 'nohup /app/kafka/kafka_2.11-1.1.0/bin/kafka-server-start.sh /app/
                kafka/kafka_2.11-1.1.0/config/server.properties > /dev/null 2>&1 &'\n"
          ]
        ]
      }
    }
  }
},
"KafkaBrokerInstance2": {
  "Type": "AWS::EC2::Instance",
  "Metadata": {
    "region": {
      "Ref": "AWS::Region"
    },
    "stack_name": {
      "Ref": "AWS::StackName"
    },
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
    "AWS::CloudFormation::Init" : {
      "config" : {}}
  },
  "DependsOn" : "KafkaZookeeperInstance",
  "Properties" : {
    "ImageId" : {
      "Fn::FindInMap" : [
        "AmiId",
        {
          "Ref" : "AWS::Region"
        },
        "AMI"
      ]
    },
    "InstanceType" : {
      "Ref" : "KafkaServerInstanceType"
    },
    "IamInstanceProfile" : {
      "Ref" : "IAMProfile"
    },
    "KeyName" : {
      "Ref" : "KeyName"
    },
    "BlockDeviceMappings" : [
      {
        "DeviceName" : {
          "Ref" : "EBSDeviceName"
        },
        "Ebs" : {
          "VolumeSize" : {
            "Ref" : "EBSVolumeSize"
          },
          "VolumeType" : {
            "Ref" : "EBSType"
          }
        }
      }
    ],
    "NetworkInterfaces" : [
      {
        "GroupSet" : [
          {
            "Ref" : "KafkaServerSecurityGroup"
          },
          {
            "Ref" : "SSHSecurityGroup"
          }
        ],
        "AssociatePublicIpAddress" : "true",
        "DeleteOnTermination" : "true",
        "DeviceIndex" : "0",
        "SubnetId" : { "Ref" : "SubnetID" }
      }
    ],
    "Tags" : [
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
        {
          "Key":"Name",
          "Value":"KafkaBroker2InPublicSubnets"
        }
      ],
      "UserData": {
        "Fn::Base64": {
          "Fn::Join": [
            "",
            [
              "#!/bin/bash -v\n",
              "\nyum -y install unzip java wget\n",
              "\ncurl \"https://s3.amazonaws.com/aws-cli/awscli-bundle.zip\" -o \"awscli-bundle.
                  zip\"\n",
              "unzip awscli-bundle.zip\n",
              "./awscli-bundle/install -b ~/bin/aws\n",
              "export PATH=~/bin:$PATH\n",
              "export ZOOKEEPER_ID=1\n",
              "su - root -c 'mkdir -p /app/kafka'\n",
              "wget ",
              {
                "Ref": "KafkaDownloadURL"
              },
              " -P /app\n",
              "export file=`echo ",
              {
                "Ref": "KafkaDownloadURL"
              },
              " | rev | cut -f1 -d '/' | rev`\n",
              "tar -zxvf /app/$file -C /app/kafka\n",
              "su - root -c 'mkdir -p /tmp/kafka-logs'\n",
              "sed -i.bak \"s/zookeeper.connect=.*/zookeeper.connect=",
              {
                "Fn::GetAtt": [
                  "KafkaZookeeperInstance",
                  "PrivateIp"
                ]
              },
              ":2181/g\" /app/kafka/kafka_2.11-1.1.0/config/server.properties\n",
              "sed -i.bak \"s/#advertised.listeners=.*/advertised.listeners=PLAINTEXT://`curl
                  http://169.254.169.254/latest/meta-data/public-hostname`:9092/g\" /app/
                  kafka/kafka_2.11-1.1.0/config/server.properties\n",
              "sed -i.bak \"s/broker.id=.*/broker.id=1/g\" /app/kafka/kafka_2.11-1.1.0/config/
                  server.properties\n",
              "sed -i 's/Defaults    requiretty/Defaults    !requiretty/g' /etc/sudoers\n",
              "sed -i 's/KAFKA_JMX_OPTS=\"-D/KAFKA_JMX_OPTS=\"-Djava.net.preferIPv4Stack=true -D/g
                  ' /app/kafka/kafka_2.11-1.1.0/bin/kafka-run-class.sh\n",
              "su - root -c 'nohup /app/kafka/kafka_2.11-1.1.0/bin/kafka-server-start.sh /app/
                  kafka/kafka_2.11-1.1.0/config/server.properties > /dev/null 2>&1 &'\n"
            ]
          ]
        }
      }
    }
  },
  "KafkaBrokerInstance3": {
    "Type": "AWS::EC2::Instance",
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
"Metadata": {
  "region": {
    "Ref": "AWS::Region"
  },
  "stack_name": {
    "Ref": "AWS::StackName"
  },
  "AWS::CloudFormation::Init": {
    "config": {}
  }
},
"DependsOn": "KafkaZookeeperInstance",
"Properties": {
  "ImageId": {
    "Fn::FindInMap": [
      "AmiId",
      {
        "Ref": "AWS::Region"
      },
      "AMI"
    ]
  },
  "InstanceType": {
    "Ref": "KafkaServerInstanceType"
  },
  "IamInstanceProfile": {
    "Ref": "IAMProfile"
  },
  "KeyName": {
    "Ref": "KeyName"
  },
  "BlockDeviceMappings": [
    {
      "DeviceName": {
        "Ref": "EBSDeviceName"
      },
      "Ebs": {
        "VolumeSize": {
          "Ref": "EBSVolumeSize"
        },
        "VolumeType": {
          "Ref": "EBSType"
        }
      }
    }
  ],
  "NetworkInterfaces":[
    {
      "GroupSet":[
        {
          "Ref":"KafkaServerSecurityGroup"
        },
        {
          "Ref":"SSHSecurityGroup"
        }
      ],
      "AssociatePublicIpAddress": "true",
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
            "DeleteOnTermination": "true",
            "DeviceIndex":"0",
            "SubnetId": { "Ref" : "SubnetID" }
          }
        ],
        "Tags":[
          {
            "Key":"Name",
            "Value":"KafkaBroker3InPublicSubnets"
          }
        ],
        "UserData": {
          "Fn::Base64": {
            "Fn::Join": [
              "",
              [
                "#!/bin/bash -v\n",
                "\nyum -y install unzip java wget\n",
                "\ncurl \"https://s3.amazonaws.com/aws-cli/awscli-bundle.zip\" -o \"awscli-bundle.
                    zip\"\n",
                "unzip awscli-bundle.zip\n",
                "./awscli-bundle/install -b ~/bin/aws\n",
                "export PATH=~/bin:$PATH\n",
                "export ZOOKEEPER_ID=1\n",
                "su - root -c 'mkdir -p /app/kafka'\n",
                "wget ",
                {
                  "Ref": "KafkaDownloadURL"
                },
                " -P /app\n",
                "export file=`echo ",
                {
                  "Ref": "KafkaDownloadURL"
                },
                " | rev | cut -f1 -d '/' | rev`\n",
                "tar -zxvf /app/$file -C /app/kafka\n",
                "su - root -c 'mkdir -p /tmp/kafka-logs'\n",
                "sed -i.bak \"s/zookeeper.connect=.*/zookeeper.connect=",
                {
                  "Fn::GetAtt": [
                    "KafkaZookeeperInstance",
                    "PrivateIp"
                  ]
                },
                ":2181/g\" /app/kafka/kafka_2.11-1.1.0/config/server.properties\n",
                "sed -i.bak \"s/#advertised.listeners=.*/advertised.listeners=PLAINTEXT://`curl
                    http://169.254.169.254/latest/meta-data/public-hostname`:9092/g\" /app/
                    kafka/kafka_2.11-1.1.0/config/server.properties\n",
                "sed -i.bak \"s/broker.id=.*/broker.id=2/g\" /app/kafka/kafka_2.11-1.1.0/config/
                    server.properties\n",
                "sed -i 's/Defaults    requiretty/Defaults    !requiretty/g' /etc/sudoers\n",
                "sed -i 's/KAFKA_JMX_OPTS=\"-D/KAFKA_JMX_OPTS=\"-Djava.net.preferIPv4Stack=true -D/g
                    ' /app/kafka/kafka_2.11-1.1.0/bin/kafka-run-class.sh\n",
                "su - root -c 'nohup /app/kafka/kafka_2.11-1.1.0/bin/kafka-server-start.sh /app/
                    kafka/kafka_2.11-1.1.0/config/server.properties > /dev/null 2>&1 &'\n"
  ]]}}}}
},
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)

```
  "Outputs":{
    "KafkaBrokerInfo":{
      "Value" : {
        "Fn::Join" : [ ":", [ { "Fn::GetAtt" : [ "KafkaBrokerInstance", "PublicDnsName"]}, "9092" ]]
      },
      "Description":"PublicDnsName of KafkaBroker.\n"
    },
    "KafkaBrokerInfo2":{
      "Value" : {
        "Fn::Join" : [ ":", [ { "Fn::GetAtt" : [ "KafkaBrokerInstance2", "PublicDnsName"]}, "9092" ]
              ]

      },
      "Description":"PublicDnsName of KafkaBroker2.\n"
    },
    "KafkaBrokerInfo3":{
      "Value" : {
        "Fn::Join" : [ ":", [ { "Fn::GetAtt" : [ "KafkaBrokerInstance3", "PublicDnsName"]}, "9092" ]
              ]

      },
      "Description":"PublicDnsName of KafkaBroker3.\n"
    },
    "ZookeeperInfo":{
      "Value" : {
        "Fn::Join" : [ ":", [ { "Fn::GetAtt" : [ "KafkaZookeeperInstance", "PublicDnsName"]}, "2181"
              ] ]
      },
      "Description":"PublicDnsName of Zookeeper.\n"
    }
  }
}
```

Figure A.1. Amazon Cloudformation script to setup Kafka cluster.(cont.)