Big Spatial Data Management for the Internet of Things: A Survey

Big Spatial Data Management for the Internet of Things: A Survey

**Isam Mashhour Al Jawarneh, Paolo Bellavista, Antonio Corradi,
Luca Foschini, and Rebecca Montanari**

Dept. Computer Science and Engineering (DISI), University of Bologna,
Viale del Risorgimento 2, 40136 Bologna, Italy

Abstract. The high abundance of IoT devices have caused an unprecedented accumulation of avalanches of geo-referenced IoT spatial data that if could be analyzed correctly would unleash important information. This can feed decision support systems for better decision making and strategic planning regarding important aspects of our lives that depend heavily on LBSs. Several spatial data management systems for IoT data in Cloud has recently gained momentum. However, the literature is still missing a comprehensive survey that conceptualize a convenient framework that classify those frameworks under appropriate categories. In this survey paper, we focus on the management of big geospatial data that are generated by IoT data sources. We also define a conceptual framework and match the woks of the recent literature with it. We then identify future research frontiers in the field depending on the surveyed works.

Keywords: Spatial Data, Spark, MongoDB, Spatial Partitioning, Internet of Things, Query Optimizers.

1. Introduction

In the last decade or so, the proliferation of ubiquitous positioning devices, and a massive spread of the Internet of Things (IoT) paradigm have caused an accumulation of an unprecedented huge mass of datasets, forming a phenomenon referred to as big data. Today, all kinds of businesses are data-driven, with data being mostly geocoded and real-time [1] , making timely analysis a priority, and thus promoting the emergence of Geographic Information Systems (GISs), with wide spectrum of applications, including participatory healthcare [2] , neurology analytics [3] , medical pathology imaging [4] , and city planning [5]. IoT is loosely defined as a network of interconnected computing devices that may constitute home electronic appliances (e.g., security systems and cameras), connected vehicles, and sensor-enabled positioning devices (and actuators) which communicate endlessly and transfer data in real-time [6]. Devices involved in an IoT network does not normally encapsulate large storage and processing capacities. They otherwise depend on sending data loads through a middleware network layer to backend storage media and capable processing systems.

Storing and processing huge avalanches of spatial data by merely depending on traditional centralized batch-processing systems is inconvenient [7]. Because of inherent difficulties in dealing with such huge data traffic, horizontal scalability is becoming essential, where beefing up (scaling up) single node deemed insufficient. Therefore, various parallel-GIS systems have spawned and gained momentum in recent years. A trend that has been made possible by the emergence and widespread adoption of cluster and Cloud computing [8]. Cloud computing can be loosely defined as a paradigm that offers a rapid access to a pool of interconnected computing and storage resources. In stark contrast with IoT, Cloud and cluster computing environments are prepared with, theoretically, unlimited scalable storage and computation capacities. In today's dynamic and scalable applications scenarios, it is becoming a norm that IoT devices serve avalanches of geo-referenced data loads to Cloud/cluster computing environments, allowing both paradigms to operate synergistically so as to process huge amounts of (near)real-time geo-referenced data streams with QoS guarantees. A typical architecture that promotes the integration of those two magnets (Cloud and IoT) is a simple two layers tiered architecture that is composed by the IoT network layer and a Cloud layer [9] . However, this integration between IoT and Cloud does not come for free and has its limitations and drawbacks. Heterogeneous data loads are aggregated from diverse IoT sources and need to be unified and transferred into a Cloud environment. Despite widely adopted, Cloud computing is confronting critics in highly dynamic application scenarios. Perhaps most importantly because it harmfully violates QoS goals envisaged in an SLA at times [8]. Challenging aspects that potentially deteriorate the benefits that we reap from parallel computing that is offered by Cloud need to be addressed. Of a special interest, a problem that is known as *data partitioning.* In its essence, data partitioning means splitting the huge amounts of arrived data to computing (or storage) nodes in a Cloud. Traditional partitioning

approaches focus on balancing loads by sending roughly same size of data loads to worker (or synonymously storage) nodes of the Cloud. Load balancing alone is not enough for processing spatial data loads. Spatial data normally exhibits geographical spatial pairwise relationships, which is also known as *spatial autocorrelation*, where spatial objects normally collocate in the same real geometry because they are closely in a relationship. Respecting such relationship while splitting spatial data loads to worker nodes of Cloud has proved to be efficient [10]. For example, by sending geometrically-proximate spatial objects to same (or geographically nearby) worker nodes of the Cloud. This, in its turn, normally reduces the data shuffling that may otherwise be enlarged. This is attributed to the fact that interesting analytics seeks to reveal patterns behind this kind of autocorrelation, aiming, for example, at solving complex problems that would otherwise remain elusive. For example, analyzing spatial autocorrelation to help containing a contagious infectious disease before it spreads far away [11]. By placing geographically-nearby objects in same (or close-by) Cloud nodes, we guarantee that answering such complex spatial queries will not result in a lot of data shuffling, thus helps in avoiding a network congestion.

Works of the related literature have mainly focused on internetworking and hardware aspects of the problem [6, 12]. That said, the share of works in spatial data management aspects for IoT remains humble. Few works have also focused on surveying the analytical aspects for IoT [8, 13, 14] . However, those are general and did not discuss the spatial characteristics of the IoT data. We posit that more attention must be given to the big spatial data management aspects for IoT, with intercommunication, process and scalable storage be predominant players while designing spatial data management solutions for the Cloud (or in-house cluster) computing.

To close this void, in this survey paper, we focus on frameworks that are dedicated to the management of big geospatial data that are generated by heterogeneous IoT devices and served to either Cloud computing or in-house private computing clusters for processing and scalable storage. We survey those frameworks in the sense that reveals the QoS aware optimizations they offer for the management of such data in Cloud settings (and cluster in-premises computing). We refer to the Cloud that is leased by a third-party as a *public Cloud* (such as Amazon EMR and Microsoft Azure), whereas the one that is a propriety of an individual organizations is referred to as *private Cloud*. For simplicity, for the remaining parts of this paper, we will refer to them both as '*Cloud'*. By QoS in this context, we are interested in time-based QoS goals such as latency/throughput, accuracy goals (such as estimation quality) in Spatial Approximate Query Processing Systems (SAQP), in addition to resource utilization goals. It worth noticing that partitioning per se is a mean-to-an-end, where the goal is to achieve a set of QoS goals prespecified in a Service Level Agreement (SLA). Consequently, we also survey strategies that improve query optimizers in spatial big data management frameworks, such as indexing and caching. All in all, we aim at surveying QoS aware optimizations that geospatial big data management systems offer in Cloud and cluster computing environments. In addition to those contributions, we propose a unique general framework that hosts under its umbrella the QoS aware spatial data management optimization treated as a first-class citizen that is transparently incorporated atop the layers of the underlying systems. This enables those systems to relief the shoulders of front-end developers from reasoning about the underlying logistics of QoS aware big spatial data management in Cloud. Stated another way, in this survey paper, we capitalize on most important aspects of QoS-aware big geospatial data optimizations from the angle of a Cloud computing infrastructure and its synergy with IoT. Our framework is compatible with a typical general architecture that synergistically integrate IoT with Cloud computing [6, 12] .

To fence ourselves within a reasonable scope, we are not focusing on general big data frameworks that come readily deployed with Cloud infrastructures (in what is commonly known as Software as a Service, SaaS for short) such as Apache Spark [15], Hadoop [16] and MongoDB [17, 18]. We otherwise focus on spatial-aware frameworks that are engineered on top of some of those, aiming at serving QoS-aware optimization patches injected transparently within the layers of the underlying core engines. For example, GeoSpark [19] is engineered atop Spark core engine and introduces few patches for spatial-aware data management in Cloud parallel computing environments for data consumed from IoT devices.

The remaining parts of this survey paper are organized as the following. We start by an overview, providing an essential background on the related aspects of the topic, motivating the need for QoS-awareness in big geospatial data context and drawing a general-purpose architectural view accordingly.

This is followed by a general framework that we propose for big geospatial data management in Cloud-alike parallel computing settings. Thereafter, we present a comprehensive categorization of QoS-aware spatial data management techniques in Cloud. We then elaborate the discussion by cross-matching techniques with most relevant literature studies. In what follows, we define a unique taxonomy that divides QoS optimizations into relevant sections. We summarize by recommending few research frontiers.

2. Background

In this section, we highlight various initiatives that are essential for classifying QoS- and spatial-aware data management optimizations for a two-layer architecture that models the integration between IoT and Cloud. First, we elaborate a toy motivating example that deliberately focus on some scenarios where such optimizations are essential for an efficient system performance. Also, we review a typical IoT-Cloud architecture that is widely accepted in the related literature.

2.1 Highly Dynamic Environments as a Motivating Scenario

We herein discuss a typical application scenario that imposes harsh QoS constraints on an underlying big geospatial data management system that is receiving spatial data from IoT on a regular basis. Consider *smart city* scenario where a decision-making application analyzes community Global Positioning System (GPS) data collected by citizens, vehicles and shared bikes moving around in a city in real-time. Let's consider an example of a citizen who is wearing a sensor-enabled health monitoring device (for example, a smart watch connected to the smartphone) with a chronic disease (e.g., Asthma) which may attack suddenly while moving around in a city, and promptly needs an instant first-aid. The goal here is providing reliable assistance to the patient and keeping the danger as low as possible. A system for managing the process of rescue in a timely fashion is required. The system is expected to achieve a set of requirements. First, it sends patient's locations to nearest emergency service point. Also, analyzing patient's health condition at the time of attack for checking severity degree. Second, system can identify communities in the surroundings of the patient (this needs a community detection algorithm [2]) and selects the best volunteer who is the nearest and capable of providing first-aid assistance. Here election is made based on distance to the patient location, social relativeness between patient and volunteer, the real ability of volunteer to provide first-aid (i.e., being trained enough or not), and all those depend on many factors (among which the degree of severity of the patient health condition has a highest priority).

Also, the system should be able to suggest the best route that the ambulance can take in order to avoid heavy road congestions. Also, sensors send street congestion data to periodic traffic signal actuators, which then decides to turn some traffic lights into green while making others red in a consistent fashion so as to pave the way for the ambulance to pass smoothly en-route to the patient location.

In this simple, but representing scenario, people with their smartphones , patients with a smart watches, street congestion sensors, and vehicles GPS-enabled devices are all considered "Things" that regularly send their geo-referenced datasets through a networking layer to the Cloud (or in-house cluster) computing network that hosts a geospatial analytics system which operates in parallel. In the next subsection, we briefly discuss a reference architecture that has been applied widely for similar scenarios in the relevant literature.

2.2 IoT-Cloud Reference Architecture

In this section, we schematize a typical reference IoT-Cloud architecture that has been widely applied for scenarios similar to our toy example that we have discussed section 2.1. We specifically discuss its limitations in meeting the QoS goals envisaged through an SLA.

Figure 1. shows a reference IoT-Cloud architecture showing a typical interplay and interaction between constituting components, where IoT constitutes of devices (i.e. 'Things') at a bottom layer, which are physical objects that are attached to sensors that, in turn, collects relevant geo-referenced data and serve it through a middle network communication layer all the pyramid up to a top layer that hosts computation and processing systems [20]. In this survey paper, we focus on those ecosystems that are deployed in a Cloud. An inherent problem with this simple architecture, however, is that fact that it fails to convey the shape of

the transferred data to the Cloud. Stated in other terms, since most data that arrives from IoT is geo-referenced, encapsulating a spatial multidimensional structure. This structure is however flattened into tabular formats and the applications of the Cloud then need to reconstruct the multidimensional shape of data to analyze it correctly, thus counteracting the benefits we may reap from the Cloud parallel computing. In the next subsection, we briefly review the state-of-art big data management systems that come readily deployed on public Clouds.

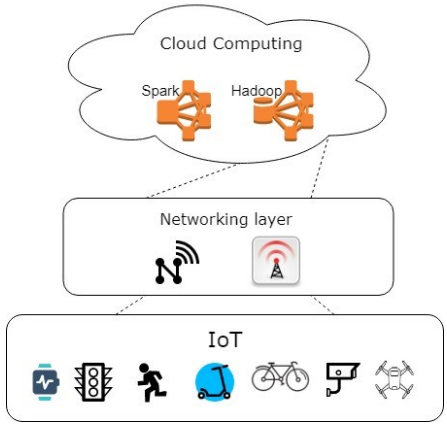

Figure 1. IoT-Cloud Two-layers Architecture

2.3 Traditional Cloud-based Big Data Management Frameworks for IoT

It was toward the end of 1970s that interest has turned from the hierarchical data representation model [21] (which was dominant in DBMSs) to relational models in RDBMSs. However, in the last two decades, companies have shifted their attention toward real-time big data analytics with increasingly complex queries. The performance shortcomings of relational databases in processing big datasets have raised the demand to introduce parallel data management schemes that are deployed on Cloud computing environments, and the transition from monolithic to scalable horizontal architectures has become a necessity. As means of coping with those challenges, database world has witnessed the birth of two indispensable paradigms for big data management (in two veins, storage and processing) in Cloud. I) non-relational storage-oriented DBs (commonly known as NoSQL, a shorthand for 'Not only SQL', we use those henceforth interchangeably), and II) big data processing-oriented engines.

Examples of NoSQL databases include Amazon Dynamo DB [22] , Facebook Cassandra [23] , and Google Bigtable [24] , MongoDB [17] and HBase [25]. The common characteristic of those systems is that notion of referential integrity (consequently schema notion) is absent in those systems, leaving data formatted and organized in a way that makes it self-describing, making it unnecessary to declare the structure to the system a priori. Such flexibility makes non-relational databases easily adaptable to scenarios where data is uneven, or frequently and unpredictably changing its structure or content. [26] Classifies non-relational databases into four categories, document-oriented, graphic, key-value, and wide column databases. Document-oriented databases adopt a document structure (typically a JSON-alike format) that resembles the object model in object-oriented databases, where data for each object are stored in a single document (analogous to records in relational tables) instead of being scattered across multiple tables, thus simplifying data access and reducing the join processing, which otherwise would be necessary as a cause of referential integrity in RDBMs. MongoDB [17] is an example document-oriented NoSQL database. Key-value DBs store keys and related values in hash tables (Dynamo DB [22] is an example). Wide-Column DBs are column-oriented data structures that assign multiple attributes for each key. Examples include Cassandra [23] and Bigtable [24].

With MongoDB and HBase being notably the two predominant frameworks. We restrict ourselves to those wide-columns and document stores. Key/value and graph NoSQL stores are outside the scope of this survey. To the best of our knowledge, the latter are not widely adopted for spatial data management. NoSQL databases are storage-oriented solutions not intrinsically engineered for handling data processing workloads, which, in turns, has led to nearly a simultaneous emergence of processing-oriented parallel-computing solutions for the Cloud, based on the MapReduce paradigm [27] (for example, Hadoop [16]) or its successor variants (for example, Apache Spark [15]).

Those scalable systems are normally deployed on Cloud. They are general-purpose and are unaware of the spatial characteristics of IoT data. In other words, they do not normally embed specialized management strategies for big spatial data loads. Consequently, spatial-aware extensions, mechanisms and strategies are required, which has led to the emergence of spatial-aware glues and patches atop the codebases of those ecosystems.

2.4 IoT Requirements for Big Spatial Data Management Frameworks in the Cloud

IoT impose highly-demanding (sometimes harsh) constraints and requirements on spatial data management frameworks that are deployed in a Cloud, aiming basically at achieving an acceptable degree of user satisfaction. We start by common requirements from the existing literature [8, 13, 28] , then we extend the definition to incorporate strict QoS aware requirements that appear in highly dynamic application scenarios similar to the smart city scenario that we have presented in section 2.1, which necessitate the deployment of spatial-aware big data management frameworks with custom services for IoT in the Cloud. This has led to the emergence of a constellation of frameworks that we term as *big spatial data management for IoT*.

We consider highly dynamic application scenarios that require intermixing loads from heterogeneous IoT sources (mostly in a mashup fashion). From those scenarios, we extract requirements that are common among those kinds of analytics, which should be transparently achieved by the Spatial Data Management System (SDMS) in order to be considered efficient for IoT in the Cloud.

Common requirements that we have identified from most emergent highly dynamic application scenarios (such as smart cities, Industry 4.0 [29] and Industrial IoT), which should be translated afterwards into architectural design goals for a qualified SDMS for IoT in Cloud include the following.

QoS guarantees. Spatial data management systems for IoT in Cloud should ensure that the system runs within a prespecified list of time-based QoS guarantees expressed as latency/throughput and accuracy goals (high-throughput/low-latency). Also, those systems should work on maximizing the resource utilization in Cloud, such as to cut the unnecessarily additional costs from the user. Also, results accuracy should not be affected above allowable error margins.

An efficient SDMS should seeks at transparently incorporating services that achieve a plausible balance between those goals. Those services should be offered within the framework in a way that relieves the shoulders of the programmers from having to reason atomically about them, allowing them thus to focus on the data analytics tasks themselves instead of spending unnecessarily extra time in handling QoS logistics.

We have identified two architectural design perspectives in SDMSs that are normally considered in order to design SDMSs for the Cloud which are able to achieve a plausible balance between the abovementioned QoS goals. Those are, *data partitioning* and *query optimizers*. They are heavily discussed in the next section.

3. A Conceptual Spatial-Aware Big Data Management Framework for IoT in the Cloud

In this section we convey a general conceptual architecture for spatial-aware big data management for IoT. It serves as a springboard where we explain the main components that comprise the underlying systems. We will use those components afterwards (specifically in section 4) to draw a unique taxonomy where we

cross match optimizations from the relevant literature with components of the framework that is schematized in Figure 2.

In addition, we introduce data partitioning, elaborating on its main challenges in the spatial aware context and building taxonomy for traditional data partitioning approaches, thus providing a relevant background in this vein and paving the way for a convenient categorization of spatial-aware portioning methods. Alongside, the same applies for the query optimizers.

In the last few decades, because of an elevated demand for analyzing big geospatial data, and because current processing systems alone are unable to keep pace with those increasing demands, GISs have evolved from centralized single-device systems to parallelized cloud-based systems (examples include Hadoop-GIS [30], SpatialHadoop [31] , SpatialSpark [32] and GeoSpark [19]). Even though those systems are based on a variety of parallel data processing ecosystems (basically Hadoop and Spark), many influences funnel the style of those systems into an isomorphic layered architecture, encompassing three primary layers (Figure 2. tells the story). The bottom layer is either a NoSQL parallel-DBMS, or big data processing-oriented codebase core. The middle layer represents a specialized spatial-awareness extension for various purposes, including spatial data representation or reformatting. The top layer is a service layer, providing services such as custom data partitioning strategies and custom query processing optimizers. However, two main elements are yet to be given more attention in this context. Those are, *data partitioning* and *query optimizers*. Despite architectural generality, systems mentioned above differ on their partitioning and querying strategies as herein follows.

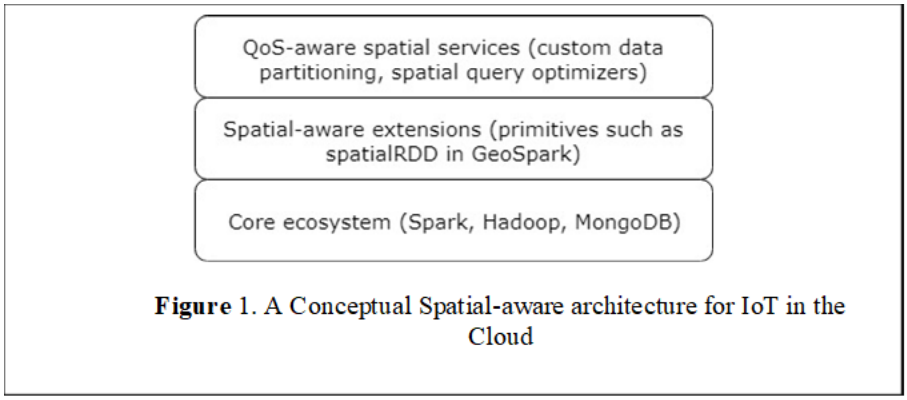

Figure 1. A Conceptual Spatial-aware architecture for IoT in the Cloud

In this survey paper, we focus on the top layer of the architecture in Figure 2. We have selected to scope ourselves to this layer because we believe that custom spatial-aware data partitioning and query optimizers are important factors en-route to achieve the IoT requirements that we have discussed in section 2.4. Data partitioning and query optimizers does not operate in isolation, they instead complement each other synergistically in order to achieve QoS goals prespecified in SLA.

3.1 Spatial Data Partitioning

Data partitioning is loosely defined as a technique for distributing partitions of data over several processing elements (i.e., worker nodes) in a parallel computing environment (i.e., Cloud), where processing is accomplished simultaneously by each processor instance on the corresponding partition. This parallelism is essential for the operation of the two parallel big data management paradigms, processing-oriented and storage-oriented. Comparatively speaking, NoSQL (storage-oriented parallel-DBMSs) partitions data, aiming at a scalable storage and management it in a parallel fashion, whereas processing-oriented systems partition data in order to accelerate query processing. The process of data parallelization is schematized in Figure 3.

Spatial data partitioning comprises strategies that divide the embedding space (the space from which spatial samples are drawn) using hierarchical representations structures (e.g., grid-based and tree-based) or non-

hierarchical representations such as Minimum Bounding Rectangles (MBR), both explained shortly in section 4. Those spatial representations are used to split the IoT spatial data into worker nodes of the Cloud.

In the next subsection, we identify challenges that may hinder the achievement of QoS-aware data partitioning for IoT spatial geo-referenced datasets in the Cloud.

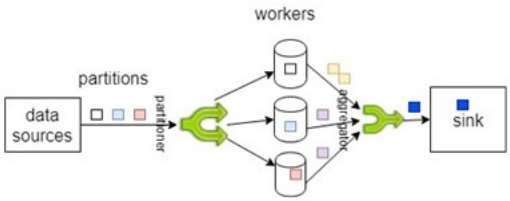

Figure 1. a typical data parallelization for IoT in Cloud

3.1.1 *Spatial data partitioning goals in Cloud*

We have identified three contradicting goals focusing specifically on spatial data partitioning, which determine the QoS of big spatial query processing. i) *Load balancing*, which is the process of de-clustering data loads in a way that guarantees an even distribution among all partitions, thus mitigating data skewness. While this is efficient for general-purpose data loads, it is insufficient for geospatial datasets. Spatial data loads often show co-location continuum relations. We refer to this characteristic as ii) *Spatial Data Locality* (SDL) *preservation*. Preserving this co-location feature is essential for an optimized big geospatial data analytics performance. By achieving SDL preservation while splitting data, the partitioning strategy aims at minimizing cross-partition spatial data access operations. For example, proximity-alike spatial queries normally require accessing spatial tuples (representing objects) that are geometrically-nearby. By being able to preserve such a proximity relation while splitting data, by for example sending geographically-nearby objects to same partitions, the system axiomatically reduces cross-partition access as it only accesses some partitions that host appropriate objects. The partitioning scheme should also, for the same purpose and at the same way, aim at minimizing cross-partition joins. iii) *Boundary Spatial Objects* (BSO) minimization. Imagining the earth flattened out (a.k.a. Euclidean space or flat surface) and split into cells (forming a grid network). We refer to spatial objects residing exactly on borders between cells as *Boundary Spatial Objects* (BSO). Accounting for those in a partitioning scheme is specifically challenging, as it imposes extra processing overhead on the system. Specifically, if BSOs constitute a large portion of the spatial dataset. This can be extremely detrimental to the processing operator in cases such as join processing, especially that most well-performing join algorithms are based on *filter-and-refine* approach, where processing BSOs (a.k.a. edge cases) in the refinement stage requires applying the real geometry processor which is computationally expensive and turns prohibitive in extreme scenarios.

We classify load balancing methods as either being *embedded* or *adaptive*. The former means that the original data partitioning method transparently achieves a plausible degree of load balancing, whereas the latter means that the SDMS needs to perform additional steps after partitioning (which normally requires repartitioning) in order to achieve a good degree of load balancing.

An efficient Spatial Data Management Engine (SDME) targets at allocating roughly equal weights of spatial objects to processing elements, preserving, as much as possible, the SDL by grouping geometrically-nearby objects within same subdomains, and minimizing BSOs. To achieve those, various works of the literature have designed spatial-aware custom partitioning strategies that collectively provide top service layer for solving some of those goals in a way that guarantees an acceptable degree of balance between them as discussed hereafter. We evaluate representatives of those works based on the three goals mentioned above. We first review classical data partitioning methods, as complex spatial-aware methods are based on them. Afterwards, we provide taxonomies for spatial-aware partitioning schemes.

3.1.2 *Spatial query optimizers*

Data partitioning is a mean-to-an-end and the goal would be to optimize spatial queries that are imposed on a SDMS. Query optimizers include spatial indexing and caching strategies. Access structures (i.e., indexes) are normally imposed on the spatial representations in order to speed the access. Query routers normally employ those structures in order to (sometimes aggressively) prune the search space upon receiving a spatial query by forwarding query requests only to partitions (hosted in Cloud worker nodes) that potentially contain part of the result set. This conceptualization reflects the importance of SDL preservation while partitioning spatial IoT data.

Some works have went beyond applying a single access structure by employing instead a multi-layer access structure in a staggered fashion. For example, an imposing a z-order curves ordering structure after a B-tree index in MongoDB, thus comprising a compound index, where the former is applied for pruning the search space by specifying the order of visiting grid cells upon receiving a query (which is then considered as a global index), whereas the latter is applied locally to each grid cell to further prune the search space.

4. Big Spatial Data Management Systems for IoT: a Taxonomy

This section elaborates on taxonomies of spatial-aware optimization strategies for big geospatial data management for IoT in the Cloud. We consider two veins. Those are, storage-oriented and processing-oriented.

4.1 Spatial-aware Partitioning

We first briefly recapitulate the relevant data structures that support the introduction of spatial-aware data partitioning methods for IoT in Cloud. We then taxonomize the rending QoS-aware spatial data partitioning methods in a relevant alignment with the discussed data structures.

4.1.1 *Multidimensional data structures in support for spatial partitioning*

Data structures that support spatial partitioning can be classified under two general categories, either data-dependent or space-dependent (a.k.a. data-independent). Under those broad categories, we classify four prevalent methods for splitting the embedding space from which spatial objects are drawn.

Hierarchical splitting methods fall under data-independent category. Grid [33, 34] and quadtrees [35] are the two most applied methods. Grid-based structures. They work by partitioning the embedding space into a grid-shaped structure (uniform or arbitrary). Quadtree [35] (and its k-d tree [36] variation) also falls under the category of data-dependent space representation structures. They work by recursively dividing a two-dimensional Euclidean space into four square divisions until each division contains no more than one spatial object. The distinction between grid splitting and quadtrees is that the former normally splits the embedding space into uniformly-shaped cells, whereas the latter depends on the data distribution to decide upon the splits. Quadtree approaches are more convenient for highly skewed datasets such as spatial data as they have spatial data statistics in registration, generating more partitions in regions with high data density. In other words, Grid structures are susceptible to load imbalance while they preserve SDL. On the other hand, quadtrees can achieve a plausible balance between SDL preservation and load balancing. It worth noticing that, despite mentioned separately in the literature, quadtrees are formed by a tree indexing structure on the grid so that geometrically-congruent grid cells that are empty are joined into bigger enclosing empty cells.

Other space-partitioning methods include Voronoi diagrams which partitions a Euclidean plane into polygonal regions such that each spatial point within a region is closer to a central point in its region when comparing its distance to other regions central points any other site.

As a way of contrast, data-dependent splitting methods depend on objects hierarchy as they project spatial space from which objects are drawn into a higher level up in the pyramid, thus guaranteeing to have the enclosed objects in registration (i.e., their space identity is preserved). Under this category falls R-tree and R^+-tree [37]. Spatial objects reside in leaf nodes of the R-tree (implemented normally as a B^+-tree because the spatial objects reside in leaf nodes only). R-trees are specifically useful in online (non-stationary) settings as spatial objects can be added to the tree in a hot-swappable fashion instead of waiting all objects

to arrive. The distinction between R-tree and R^+-tree is that the former is based on overlapping bounding rectangles (representing the embedding space), whereas the latter is based on non-overlapping bounding rectangles. The tradeoff in this situation is apparent, as for the non-overlapping representations, a spatial object spans many bounding rectangles with which it intersects, thus expanding the tree height and the storage space, therefore. However, a speed up is easily obtained at query run time. On the contrary, for overlapping representations, an object resides within the boundaries of one bounding rectangle. This reduces the tree size on the cost of an increased search space at query time.

Grid representations (and synonymously quadtree-based representations) can be enriched with space-filling curves [38] **,** which are ordering (a.k.a. Linearization [39]) representation-enriching structures that projects a multidimensional space (representing the embedding space) to a one-dimensional space, thus acting as a dimensionality reduction approach. Ordering structures normally follow other approaches to enrich them. For example, z-order curves can be imposed on grid structure to specify the order at which grid cells will be traversed at query time. Z-order curves loosely preserve the locality of spatial objects.

A special application of Z-order curves is geohash [1], which generates geocodes as strings the larger the shared prefix the more geometrically-proximate spatial objects.

Because every partitioning method has its own drawbacks, most works of the related literature have applied custom spatial partitioning approaches that are based on a mashup of the primitive types that we have discussed in this section. In the next subsection, we recapitulate representative custom spatial partitioning methods for IoT data in Cloud.

4.1.2 *Spatial partitioning methods for IoT data in Cloud*

We first review common custom spatial partitioning approaches and thereafter we build a taxonomy for their application in modern big spatial data management frameworks, together with the primitive types mentioned in section 4.1.1. We also identify their pros and cons of each method in relation to the three spatial partitioning goals that we have discussed in Section 3.1.1, thus providing a guidance to the community interested in big spatial data management for IoT in Cloud. Custom spatial partitioning methods that are most common include the following:

I) Binary Space Partition (BSP) tree [40] . It is a method that is similar to the k-d tree in the sense that it performs halving in every dimension. However, splitting lines are not orthogonal to the axis corresponding to a dimension (as opposed to orthogonal splitting lines in k-d trees), resulting thus in a grid of polygons. The non-regularly shaped polygonal structure helps BSP in achieving a plausible balance between SDL preservation and load balancing.

II) Sort-Tile Recurse (STR) [41] . It first tiles the embedding space into vertical slabs, where each slab contains several tiles. Horizontally-adjacent tiles (belonging to adjacent slabs) need not having straight lines that span neighboring slabs. This process results in a structure that resembles a staggered grid-shaped representation. As tiles have different sizes, STR is able to strike a balance between preserving SDL and load balancing.

III) Methods based on Space-Filling Curves (SFC) [42] , including Hilbert-curves and Z-curves, where in z-curves, data is sorted based on its order along the z-curve, thus splitting the curve into roughly equally loaded splits. SFC is an efficient geometric method for mapping spatial object location from multi-dimensional space to a linear dimension, simulating geometric space as it is flattened out, assigning a key signifying nearest spatial coordinates, thereafter sorted list constitutes the linear ordering, which cut then to equal size divisions to be distributed to working nodes of the cluster.

[1] http://geohash.org/

Other custom partitioning methods. This category includes methods that are aware of the three spatial partitioning goals, and henceforth are applied in scenarios that require accounting for them altogether. For example, some scenarios require applying a density-based clustering algorithm in Cloud (such as DBSCAN-MR), in which case any traditional spatial partitioning results in BSOs. Therefore, a custom BSO-aware partitioning method should be applied to ensure that BSOs are minimized. For example, [43, 44] have developed an adaptive dynamic data-density spatial-aware partitioning method for big geospatial datasets, where it trades-off load balancing and BSO's minimization better than traditional methods. They have applied the method in highly dynamic application scenarios that require applying density-based clustering algorithms on huge amounts of IoT spatial data in Cloud. Also, [45] have designed a query-workload-aware technique for partitioning big spatial data that adaptively renews the partitioning in accordance with a query workload (being adaptive), achieving roughly equal load balances while preserving a good degree of SDL. In the same vein, Cruncher [46] employs a dynamic adaptive method that is aware of query workload. Cost-model-based repartitioning is enabled, where a cost model calculates number of points and queries for each partition and repartitions accordingly.

Table 2 summarizes what have been sketched so far, comparing the performance of the spatial partitioning techniques, and introducing an important dimension that shows capability of every method in handling each of three main partitioning goals (load balancing, SDL preservation and BSO minimization).

Table 2. Taxonomy of capabilities of general spatial-aware partitioning methods in handling spatial partitioning challenges defined in section 3.1.1

method	big spatial data partitioning **goals**		
	load balancing	BSO Minimization	SDL preservation
Grid-based	✓	X	✓
BSP	✓	X	✓
Quadtree	✓	X	✓
STR	✓	X	✓
Grid with space-ordering (z-curves)	✓	X	✓
Custom partitioning method	✓	✓	✓

As it is apparent from the table, only a custom partitioning method will be able to strike a plausible balance between the three conflicting spatial partitioning goals. We now provide a taxonomy for some representative frameworks in both veins (i.e., processing and storage) that we consider baseline representatives. Afterwards, we provide a comprehensive taxonomy that summaries other relevant works that are based on some of those baselines.

In processing-oriented systems, over-the-counter, SpatialHadoop [31] provides few spatial partitioning methods including grid and Z-order curves (and Hilbert synonymously), STR (and STR+ which is a variant of STR), and Quadtree [47] .Hadoop-GIS currently only supports grid partitioning [30] in addition to an adaptive method to repartition overloaded cells into further cells to solve the data skewness that is persistent in spatial data loads (load balancing). They term their adaptive load balancing method as SATO. From the in-memory frameworks, SpatialSpark, currently supports three types including uniform grid, BSP and STR [32] . On the other hand, GeoSpark [48] provides a main support for uniform grid, Hilbert-curves, quadtree, R-tree and Voronoi. In storage-oriented systems, two refence systems that provide spatial partitioning supports are MongoDB in its native form (from the document-oriented NoSQL) and MR-HBase [39] which is a seminal work that provides spatial partitioning capabilities over HBase. However, HBase natively does not offer spatial support. However, it does not support partitioning on geospatial keys, so it intrinsically does not support spatial partitioning. Table 2 sums-up our taxonomy for the spatial partitioning techniques sketched previously. MR-HBase offers a hybrid custom spatial partitioning method

that is based on linearization (specifically, Z-orders). It also supports k-d tree and trie-based quadtree. MR-HBase trades off load balancing with an *embedded* method that checks on insertion time whether a threshold on the size of a physical storage bucket it exceeded, in which case it is split into several buckets. They depend on the fact that the underlying structures for splitting (K-d and Quad trees) intrinsically restrict the number of enclosing objects in each subspace. Table 3 summarizes what has been sketched herein.

Table 3. Taxonomy of spatial-aware partitioning strategies in core SDMEs

Data management paradigm	Type	System	Spatial data partitioning method									
			Space-dependent				Data-dependent			Ordering		custom
			Uniform grid	Quadtree-based	k-d tree	Voronoi	STR	R-tree & R+-tree	BSP	Hilbert-curves	Z-orders	
Storage-oriented	NoSQL-Based	MongoDB	X	✓		X	X	✓	X	X	✓	X
		MD-HBase	✓	✓	✓	X	X	X	X	X	✓	✓
Processing-oriented	Spark-Based	SpatialSpark	✓	X		X	✓	✓	✓	X	X	X
		GeoSpark	✓	✓	✓	✓	X	✓	X	✓	X	X
	Hadoop-Based	SpatialHadoop	✓	✓	✓	X	X	X	X	✓	✓	X
		Hadoop-GIS	✓	X	X	X	X	X	X	X	X	✓

What then remains incumbent is deciding which method to select for a specific IoT application scenario. Since data that is arriving from IoT is georeferenced, all scenarios that are deployed on a Cloud should seek applying methods that at least provide spatial locality preservation and load balancing. IoT scenarios that encompass online interactive processing should avoid data-dependent partitioning such as R-tree and R+-tree. The reason is that, despite simple and appealing approaches, space-depend approaches are expensive and may require reconstructing the grid after the grid cell becomes saturated. As a way of contrast, R-tree

(R+-tree) builds the tree dynamically on-the-fly as new objects arrive. In cases where speed is a priority, R+-tree is preferable over R-tree. However, in cases where two datasets need to be combined (using a spatial join for example), data-dependent methods are undesirable as they do not have maps (i.e., a map of embedding space and a map of spatial objects overlaid) in registration. Hence, spatial queries that require autocorrelation would become costly. In those scenarios, it is preferable to use a space-dependent method as they easily can solve autocorrelation questions by overlaying maps one over the other. Moreover, a profiling can be conducted on part of the IoT data to check the degree of skewness. In scenarios where data skewness is high, a quadtree-based method is to be preferred over a uniform grid counterpart. In scenarios with prevalent proximity-alike queries (such as *k*NN and spatial range search, discussed shortly in section 4.2.1), space-dependent methods based on grid and quadtree are preferred. In scenarios where user is willing to trade tiny error bounded accuracy loss for the benefit of lower latency, space-filling curves are the best.

Spatial data partitioning is a mean-to-an-end, where the goal is optimizing spatial query performance. In the next section, we categorize spatial query optimizers.

4.2 Query Optimizers

Query optimizers encompass methods that tradeoff the QoS goals mentioned in section 2.4 (low-latency/high-throughput, error-bounded accuracy, high resource utilization). Most common optimization methods depend on access structures (i.e., indexing) and caching. We first briefly discuss most common spatial queries.

4.2.1 Spatial data analytics in highly dynamic and scalable IoT scenarios

Dynamic smart city and Industry 4.0 application scenarios that require intermixing loads in an unprecedented mashup fashion are innumerable including, for example, for road traffic control [49], clustering microblogging topics by region [50]. The aspect that is axiomatic in all those scenarios is that they necessitate various spatial analytics.

Common seminal spatial queries include the following

A) **Range spatial query** (a.k.a. proximity queries). Range searches return the set of spatial objects that fall at a maximum specified range (e.g., radius) from a specific spatial object (most often referred to as focal point, query point or test point). An example spatial range search from our scenario is “finding people near an accident location in range that is equal to 1K meters maximum”. We support range spatial queries for the batch processing (explained in chapter 4).

B) **Spatial join**. In its general form, spatial join is a set of all pairs that is formed by pairing two geo-referenced datasets while applying a spatial predicate (e.g., intersection, inclusion, etc.,) [51]. The two participating sets can be representing multidimensional spatial objects. An example spatial join query from our scenario in section 1.1 is “finding boroughs to which each GPS-represented spatial point (volunteer) belongs, a.k.a. geofencing”, which requires joining spatial points with a master table representing boroughs.

C) **Spatial clustering**. Clustering algorithms basically aim at grouping identical spatial objects together into subgroups called clusters. From many types of clustering algorithms, density-based clustering [52] has picked up pace recently and is widely accepted for the overarching traits it provides. It is a class of clustering that basically works by separating spatially dense space regions from outliers, thus dense regions constitute clusters. A well-known method for density-based clustering is DBSCAN [53] . However, tailoring such an algorithm for the parallel computing environments requires attention, as a naïve solution poses heavy network communication overhead. To cope with this challenge, related versions (DBSCAN-MR [54] or MR-DBSCAN [55]) have been tuned for parallel general-purpose big data workloads. Clustering is one of the most important data analytics activities [56] . We support density-based clustering within the layers of SpatialBPE as explained in chapter 4. An example spatial clustering query form our scenario in section 1.1 is “grouping volunteers, in specific proximity to incident location, by the level of training they possess”

D) **K-nearest neighborhoods (kNN).** It is an optimization proximity search problem (i.e., based on range search queries). Formally, given a set A of points in an embedding space S and a query point (a.k.a.

test point) q ∈ S, kNN seeks to find the c ≥ 1 number of points forming a subset B such that all points in B are closest than all other points in the remaining subset (A – B). Stated another way, every point in A but not in B is at least as far away from q as the furthest point in B. More mathematically, given a query point q, a set of c ≥ 1 nearest neighbor to q is B, where B ⊆ A such that ||S|| = c and ∀ point pi ∈ (A – B), EuclideanDistance (q, pi) ≥ $\max_{qp \in B}(q, qp)$. We support kNN for batch mode within the layers of SpatialNoSQL as explained in chapter 4. An example kNN query form our scenario in section 1.1 is "finding the nearest 10 volunteers around an incident location".

4.2.2 *Spatial query optimizers*

Traditional naïve distributed query processors work by simply searching all partitions for results, even though some partitions do not contain relevant data that may contribute to the result. *Query routing* is one of the most widely accepted mechanisms in spatial query optimizers, which then acts as a pruning machine that forces the underlying system to forward spatial query requests to few partitions instead of searching all the partitions in Cloud. Two aspects facilitate query routing in Cloud based spatial data management deployments for IoT. Those are spatial indexing and caching, the topics of the next two subsections.

A. Spatial-aware Indexing

Spatial partitioning methods that we have discussed in section 4.1 are meant to strike a plausible balance between three contradicting spatial partitioning goals that we have recapped in section 3.1.1. However, they do not necessarily guarantee achieving QoS goals envisaged in highly dynamic IoT scenarios. Query routers often comprise spatial structures that are used for expediting the access to spatial partitioned data in Cloud deployments. Access structures (indexes) normally include simple structures such as arrays where each element of the array references a cell in the grid representation, in addition to tree-based structures such as B+-trees and PK-tree [57] , both can be imposed on a quadtree representation. . The way to search for spatial query result in IoT data distributed with a tree structure traversing tree node, which is expensive. This means that query optimizers should seek minimizing, as much as possible, the number of tree nodes visited for answering a spatial request.

As a representative for storage-oriented NoSQL frameworks, several spatial index structures are provided by MongoDB including single, compound (indexes on multiple fields), geospatial Indexes (2d indexes for flat planar geometry, and 2dsphere indexes for spherical geometry). However, As the time of this writing, an indexed key cannot be used for partitioning, a drawback that has been solved in [29] . HBase [25] supports basically single indexing but does not natively support spatial indexing.

Also, we have identified the following spatial-aware indexing techniques in the literature (we use them in our taxonomy hereafter) for spatial query optimization: i) Multi-level index (MLI for short). For example, Two-layer indexing – global and local. SpatialHadoop [31] employs a two-level index structure of global and local indexing. The global index references data across computation nodes while the local index organizes data inside each node. ii) spatial coding index (SCI). In this class, geo-coding keys are indexed. For example, geohash is a class of geo-coding that calculates a special string using the GPS coordinates (specifically longitude and latitude) of a spatial object (normally spatial points). iii) One-Layer Index (OLI) (for example, global indexing (GI)). iv) On-demand spatial indices (also known as on-the-fly spatial indexing [58]) (ODSI)

One reason that promoted the use of spatial-aware indices is borne out by the fact that the cost of creating and storing a spatial-aware index is amortized by the benefits we reap during query time, such as lowering latency.

B. Spatial-aware Caching

Caching is the process of storing latest results in main memory, aiming at speeding up subsequent interrelated queries, and thereby saving re-computation cost and providing system with a performance boost.

As a representative for NoSQL, in addition to its direct support for many spatial processing peculiarities, MongoDB optimizes queries automatically to make evaluation as efficient as possible. The query optimizer selects best index to use by periodically querying and selecting an index with best response time for each query type. The results of these tests are stored as query cache plans and updated periodically [17] .

On the other hand, as representatives for processing-oriented systems. [59] have designed a custom strategy that is chiefly based on caching frequently accessed spatial data objects, and thus preserving spatial-adjacency feature (interchangeably termed as SDL). Most frequently accessed objects are loaded together with their most geographically-adjacent set of related objects to a cache memory pool. We term this caching method as *spatial-locality-aware caching*. Also, LocationSpark [60] applies a dynamic in-memory caching to cache most frequently and recent spatial datasets in-memory. Perhaps more convenient, Cruncher [46] employs an adaptive caching technique for maintaining frequently accessed spatial data in-memory. The mechanism is simply based on maintaining access-pattern statistics within grid cells. This includes a usage counter and the time of last access, which are used for alternating most frequent accessed data in scenarios of main-memory shortage. We term this strategy as *adaptive instantaneous-in-memory caching*. GeoSpark [61] applies custom caching method that is co-location aware in the sense that it automatically caches intermediate results (from previous iterations) in-memory, aiming at facilitating subsequent co-location mining access. This method is a hybrid comprising adaptive and spatial-locality-aware strategies.

Intuitively, spatial-aware caching incurs extra storage cost which however is insignificant when compared to the query performance boost it provides. However, it seems that most relevant works of the literature did not apply caching. If for no other reasons than to avoid taxing precious often-small main memory resources that would be otherwise exploited for processing queued jobs.

5. Solution Comparison

In table 4, we cross-match related studies that encapsulated one of the systems mentioned in table 2, serving a systematic taxonomizing of the literature on spatial data management for IoT. For example, some of the works operates on top of Hadoop-GIS [30] and added their optimizations. So, that Hadoop-GIS does not provide optimizations for spatial locality does not intuitively imply that works operating on top of it behave similarly. For example, while Hadoop-GIS does not provide spatial locality, SATO [62] provides it through the support of Hilbert-curve partitioning, however it utilizes Hadoop-GIS only for query optimization and spatial data representation and preprocessing. We provide a comprehensive analysis of optimizations aspects for QoS-Aware partitioning of big geospatial data for IoT (which have been sketched in all previous sections). Systems surveyed are represented in a chronological order in each orientation and taxonomy section in table 4. We start by processing-oriented systems, where the order proceeds as the following. We first start by Spark-based systems. We have selected this way as Spark has recently stood out as a de facto standard for big data processing workloads. Thereafter, we swiftly shift to Hadoop-based systems, which proceed as the following. We first start by systems that are engineered directly on top of native Hadoop itself, thereafter we move to systems that sit on top of Hadoop representatives (e.g., Hadoop-GIS and SpatialHadoop). We complete our survey by listing most significant related storage-oriented systems.

For each taxonomy section, we first elaborate on representative reference systems, where others are built on top of them. For example, for Hadoop-based systems, we first focus on SpatialHadoop as a reference spatial-aware system built directly on top of Hadoop, thereafter we highlight other systems that are either built on SpatialHadoop or directly on Hadoop (but are not considered reference systems). We follow the same methodology for Spark, as we take GeoSpark and SpatialSpark as spatial-oriented reference systems built on top of Spark. We here cross-match literature works with relevant strategies discussed in section 4, in two veins, partitioning and querying.

5.1 Spatial-aware Partitioning Aspects

Several works of the relevant literature have landed their optimizations either on top of Spark or one of its spatial-aware representatives. One of the main pillars in this domain is GeoSpark [48] that has been

presented in and rapidly evolved to become a spatial-aware adjunct to Spark. It has expanded Spark RDDs with a spatial RDDs (SRDD for short) and designed a custom partitioning method that is aware of load balances, SDL and BSOs, which do not come included out-of-the-box with Spark. It guarantees load balancing by creating a global grid file that resembles the earth flattened out, thereafter disseminates each element from the SRDD into its correspondent location on a cell within the global grid. For loads to be evenly distributed, it is automatic to infer that grid cells have different sizes, which also intuitively guarantees respecting SDL. GeoSpark however is irrelevant as-is for applications that necessitate density-based clustering. To close this void, a recent work in this synergy has been engineered on top of GeoSpark is that of [43, 44] , which has focused on injecting spatial-awareness through a service-oriented layer on top of Spark (and more specifically on top of GeoSpark). This layer consisted of a spatial-density-based and adaptive (repartitioning-enabled) data partitioning method. Their method is adaptive in the sense that for every application session, it self-tunes division factors for the benefit of subsequent sessions, aiming at balancing loads among participating elements while minimizing BSOs in a density-based clustering application. Put simply, the method works by calculating automatically new cutting configurations (analogous to vertical partitioning line in planar geometry) that aim at minimizing BSOs for subsequent application sessions. It takes running time and cutting values (the most recent required by each partition) as an input, performs its computations (mathematical calculations that aim at minimizing BSOs) and returns new optimized splitting configurations. In fact, the calculation method is simply a sliding mechanism that moves the cut towards either east or west, based on a previous knowledge of processing workload of each element. The main query optimizer they provide is a density-based clustering capability that is supported through their custom portioning method.

Along the same lines, another Spark-based framework termed as LocationSpark [60] has incorporated a novel transparent layer (termed as query scheduler) within Spark, aiming at resolving load balancing for highly skewed datasets. Their patch is adaptive in the sense that it collects statistical information from each partition, thereafter, based on a cost model it repartitions data of stragglers (hotspot bottleneck-responsible partitions). In addition, LocationSpark has mitigated the SDL challenge by introducing a learning model that first samples data to learn the distribution in the real geometries, thereafter, indexing with a global index and distribute data accordingly. This global spatial index also guarantees load balancing by shuffling data around until stragglers vanish. Also, LocationSpark has introduced a novel bloom filter (termed as spatial bloom filter, sFilter for short) intended for partially solving the BSOs problem. sFilter automatically recognize whether a spatial point is included within a range, thus avoiding BSOs replication to neighboring partitions or broadcasting query point to overlapping partitions.

Perhaps one of the most significant Spark-based works that aims at mitigating SDL problem is the framework termed as Cruncher [46] , which employs a custom adaptive partitioning strategy that basically aims at optimizing spatial co-locality. It is adaptive in the sense that it maintains statistics for optimizing data partitioning by gradually adapting data partitions in a way that avoid redundant processing based on a cost model. While this method aims at resolving SDL, it also balances loads to some extent. However, it is unable of mitigating BSOs.

Off-the-shelf, SpatialSpark (appears in [32]) supports uniform grid, BSP and STR. Uniform grid intrinsically supports load balancing. BSP and STR are aware of load balancing and SDL preservation.

A recent model called Stark [97] broke into this consortium by incorporating a custom partitioning method, encapsulating the so-called Locality-Manager within Spark layers, aiming at preserving SDL in a best-effort manner. Multiple RDDs are partitioned using the same partitioning strategy, thus avoiding costly shuffling in join and co-group RDD transformations. Partitions that contain data sharing co-locality are aggregated in what they term as *collection partition*, thereafter a single collection partition is disseminated to the same processing element of the processing cluster, thus avoiding unnecessary shuffling afterwards. If a collection partition is very large, deeming it unsuitable for a single executor, it is instead mapped to various executors. They also introduced a consistent hashing scheme for elastically shrinking or expanding partitions without the need for re-partitioning. Stark partially achieves load balancing by employing a mechanism that groups multiple related partitions into the so-called *group partitions*, which can be subdivided or aggregated as needed to achieve load balancing. Also, Stark logs the delay of every transformation in every task, which, in turns, drives subsequent partitioning decisions. Another work that

shares the same name (despite different collaborators) is STARK [63] which has provided the opportunity of choosing between two partitioning schemes, grid and BSP. Grid partitioning supports SDL preservation, it however leaves partitions lopsided.

Shifting our attention now to Hadoop-based systems. A work by [64] introduced AEGIS, which is a Hadoop-based framework that mainly focuses on providing the ability to get along with the partitioning strategy based on a defined set of constraints by the user. For example, clustering images requires a close awareness for a multispectral space, thus localization of geometries within HDFS is essential, therefore the selected partitioning method is required to be able to preserve SDL, while balancing loads.

One of the most interesting Hadoop-based works in this direction is the work of [45] which has introduced AQWA, encompassing a query-workload-aware custom adaptive partitioning method. Its adaptivity flow from the fact that it incrementally repartitions datasets in accordance with query-workloads and data skewness, thus evenly distributing workloads (load balancing). It achieves this by employing a cost model that calculates query cost for each partition, thus assigning the cost with the partition as additional information, thereafter, trying to relieve query execution cost by splitting data of the slowcoaches. Based on query-workloads, this partitioning scheme partitions datasets that are geometrically co-located and queried most frequently into fine-grained partitions, thus supporting SDL preservation.

Also, [59] proposed a custom geography-aware quadripartition method that first partitions a geographic region into four sub-regions, thereafter, applies space filling curves to each sub-region. This guarantees SDL preservation as data residing in each region (geographical regions and grid cells are resembled) are aggregated within the same data block, aiming at distributing them to the same processing element. This method guarantees load balancing by keeping sub-regions with different geometric sizes reside in the same data block, thus allowing the same number of elements for each data block.

In addition, [65] hinted their design of RESQUE, a boundary and density aware spatial data partitioning scheme for pathology image analytical jobs in Hadoop. It preserves locality by being density aware. However, the authors did not discuss the exact working mechanism of their method. A work that goes hand in hand for the same authors is found in [4] , where they have introduced a custom boundary and density-aware spatial data partitioning method for digital pathology imaging analytics (as those resembles spatial data sets) in Hadoop. They handle BSOs by discarding them as they claim that pathology imaging analytics normally employs statistical based methods where tiny fraction of BSOs does not contribute to the overall result. To mitigate data skewness problem, they employ a greedy cost-based partitioning model, thus balancing workloads. The method also accounts for SDL for accelerating proximity-alike queries. However, the authors did not expand their reasoning on that. [66] has designed a Hadoop-based framework that bundles a custom partitioning scheme that is aware of BSOs, SDL and load balancing goals. The method chiefly relies on quadtrees. It embarks on by sampling source data, aiming at distributing equal-sized collections to processing elements of the cluster, thus respecting the load balancing principle. This is possible because they allow jagged shaped partitioning (opposite to rectangular partitioning) that contains many disjoint datasets from the space. Thereafter, a quadtree is built for every element during the *Map* phase (part of MapReduce job), thus generating one partial quadtree for each collection, aiming at emitting partial quadtree indexed collection (instead of a single input record) to the *Reduce* phase (part of MapReduce job). They mitigated the BSOs challenge by referencing BSOs with multiple index entries, thereafter, managed to post-process duplicate spatial index entries during top-level analytics. In addition, they provide a mechanism for maintaining co-location SDL characteristic by incorporating an additional MapReduce job for reorganizing original data source, constructing a spatially-ordered set of data and indexing it.

As a unique work within this consortium, SpatialHadoop [31] , setting at the core of Hadoop, supports a custom locality-load-aware partitioning scheme. It guarantees load balancing by fitting each equal-sized partition within one HDFS Hadoop block. This is also applicable by employing a custom calculation method for computing the number of required partitions beforehand. They also applied a statistical method for calculating co-location-preserving boundaries with different-sized tiles corresponding to space regions, thus preserving SDL. In addition, they provide two approaches for mitigating BSOs problem, where in the first one they assign a record to one partition based on best-matching, while in the other one they replicate

BSOs to bordering partitions, thereafter applies a back-stage processing for refining obtained results through query processor. SpatialHadoop also supports many other specific spatial partitioning methods. Instead of worrying about implementing their half-baked custom partitioning schemes, GISQF [67] take advantage of efforts that have been put in SpatialHadoop. Hence, intrinsically supports that same partitioning scheme.

SHAHED [68] propose a hybrid partitioning method on top of Hadoop that consists of two sub-schemes. The first is a grid partitioning. Even though grid portioning does not guarantee load balancing, in their case and since they presume the uniform distribution of their datasets, they achieve load balancing. The second sub-scheme applies z-orders on each grid cell, which achieves SDL preservation.

CoS-HDFS [69] has been injected within SpatialHadoop layers. It modified Hadoop default partitioning scheme so that it fosters the Minimum Bounding Rectangle (MBR) of blocks in a way that guarantees co-locating bordering or overlapping MBRs, thus preserving SDL. Furthermore, the framework naturally supports load balancing as different-sized MBRs contain roughly same number of spatial objects.

Hadoop-GIS (presented in [30]) supports a custom data skewness aware spatial partitioning model. Their model aims at trading off load balancing and BSOs. For load balancing, their algorithm divides source data into tiles (using Hadoop default grid partitioning scheme), thereafter highly-dense-tiles are further subdivided into granular tiles using a recursive partitioning approach. For resolving BSOs, the algorithm resumes by replicating BSOs to neighboring tiles, thereafter, applying a refinement postprocessing step (as an additional MapReduce job) for remedying duplicated spatial objects before concluding the query result.

In addition to the traditional spatial partitioning methods (uniform grid, BSP, Hilbert-curves and STR), SATO (integrated with Hadoop-GIS) [62] has designed a boundary optimized custom strip partitioning method. SATO partitioning scheme comprises four main steps. Those are *Sample*, *Analyze*, *Tear*, and *Optimize*. In Sample, a subset of the dataset is sampled to identify dense regions. Thereafter, an analyzer is responsible for gleaning data hotspots (high dense regions), thus deriving a suitable global partitioning scheme that minimizes BSOs while accounting for cross-partition load balancing. Then those coarse regions are subjected to tear, thus generating granular data-skewness aware partitions, and thereby enhancing spatial load balancing. Finally, additional partition statistics are produced and employed to optimize query's performance. Example statistics include the number of BSOs, which is useful for minimizing BSOs while repartitioning if necessary. To further improve on BSOs reduction, SATO introduced a boundary optimized strip partitioning (tantamount to strip partitioning) with a slight difference of being able to greedily select best partitioning in both directions (horizontal and vertical) that guarantees to a good extent a minimized BSOs number.

Researchers working on storage-oriented systems contend that optimizations are also possible in this direction. For example, the work by [70] has patterned a novel model called HGrid and injected it within the layers of HBase. The model bundles a custom hybrid spatial data partitioning method that combines quadtree and grid partitioning models into a robust model that aims mainly at achieving good extent of SDL preservation. The shortcomings of the z-ordering linearization in achieving SDL properly motivated their work, as z-curves do not necessarily guarantee that subsequent grid cells are geographically co-located. HGrid works by first splitting the space into equal-sized grid tiles, where each tile corresponds to a single z-ordering value, thereafter all points of each tile are contiguously listed in a regular grid composed of finer-level cells. In this structure, every spatial object is referenced by two values computed by combining quadtree z-values with regular grid indices. The combination of z-curves with regular grid offers a higher-level degree of equilibrium between load balancing and SDL preservation. [71] has incorporated one-dimensional spatial coding-based grid partitioning method known as GeoSOT for supporting spatial partitioning in HBase. GeoSOT is a linear spatial coding method that utilizes quadtrees and comprises of many levels expanding from the macro-scale level (representing the whole earth) to the most granular level (square centimeter for example). It first subdivides into tiles, thereafter for each tile z-ordering is applied. A combination that guarantees at least SDL preservation and load balancing.

Perhaps most importantly in this direction is a MongoDB-based work by [29] , which strikes a plausible balance between load balancing and SDL preservation by applying an adaptive and hybrid spatial custom partitioning method based on geohash encoding.

A seminal work that is based on HBase is MD-HBase [39], which constitutes a multidimensional index overlaid over a key/value store. They utilize a linearization approach based on Z-orders. K-d trees and Quad trees.

Even though many relevant works have proposed hybrid custom partitioning methods that aim at meshing support for the three partitioning goals, they do this attentively, as over-enhancing one of the requirements may negates the benefits of others.

5.2 Spatial Query Optimizers

In this subsection, we discuss the support of big geospatial management systems for IoT analytics by improved query optimizers that exploit some of the spatial partitioning methods mentioned in this survey. We start by some Spark baseline representatives, moving to the retiring Hadoop, and then concluding by storage-oriented systems.

As a reference system in this vein, GeoSpark [29, 48] supports basic querying through SRDD indexing, where spatial indexing (SRDD Indexing) such as quadtree and R-tree are bundled as global indexes. Also, whenever required a local spatial index is created after trading off its creation cost with the gain it provides. In addition, GeoSpark utilizes adaptive and spatial-aware caching for supporting spatial co-location patterns related queries, such as containment and range queries. Moreover, GeoSpark applies an adaptive caching method that is aware of SDL. Simply put, it caches recent results in-memory, aiming at accelerating subsequent proximity-alike access patterns.

LocationSpark [60] has designed a novel algorithm for handling basic spatial query processing needs. It supports MLI indexing scheme (local and global). For global indexing, it uses grid and region quadtrees, whereas the choice of local index is left for the end user (grid or R-tree), aiming at supporting a variety of application scenarios. This elasticity makes it possible to plugin further indexing mechanisms within LocationSpark tiered architecture. To avoid communication overheads produced by replication or duplication in conventional spatial range-based query handling methods, authors of LocationSpark have introduced a spatial bloom filter (encapsulated with global index), which effortlessly discover whether a spatial point is contained within a spatial range or not. As a complementary enhancement, LocationSpark contributes a dynamic in-memory caching that keeps recurrently accessed datasets in-memory, while spilling less frequently accessed data to disk.

SpatialSpark [32] injects R-tree indices for containment spatial queries (i.e., range). In addition, this is significantly important for supporting spatial joins.

Cruncher [46] employs an OLI indexing scheme that creates ODSI indexes for each partition depending on its statistical query workload awareness. OLI indexes include global index (k-d tree) and an ODSI that is built on-the-fly based on need. For example, if a specific partition is hit too hard by recurrent range query requests, it is then indexed on-demand for speeding-up queries, thereafter those indexes are discarded in case that the partition is not receiving more recurrent range queries. In addition, Cruncher leverages an adaptive real-time caching model that keeps frequently-accessed spatial data in-memory, aiming at minimizing the number of RDD copies in memory. It works by keeping statistical information regarding access patterns within granular grid cells. Those statistical data are used for alternating datasets between memory and disk.

Stark [72] employs a static range partitioning method with co-locality enabled, thus boosting range queries performances. However, it does not provide a special indexing or caching mechanisms for this purpose. STARK [63] supports the creation of in-memory R-tree partition-level indexing, aiming at optimizing basic spatial querying such as early pruning of partition-level spatial object not contributing toward the result. Range spatial filters have been further supported by implementing them as a Spark transformation, eliminating the need for resource-intensive shuffling.

Hadoop-based systems also provide efficient optimizations for basic spatial query processing. In [64] , multiple basic querying capabilities are supported through the elasticity that the framework provides regarding the usage of indexing schemes. For example, depending on the case in hand, user can choose to apply R-tree, k-d tree or quadtree among many other techniques. Also, a spatial aware global indexing (GI) can be constructed on demand and shared among the participating processing nodes.

In [45], AQWA maintains a set of spatial-aware main-memory data structures, upholding statistics regarding data and queries distribution, aiming at providing speedup for basic spatial queries (range queries for example). They use that structure for caching also. However, the authors did not intensively explain about the nature of the structures they provide.

Authors in [59] has proposed a multi-layer spatial indexing technique (MLSI) that basically consists of global index (GI) and local index (LI). The GI is based on quadtree, which is the first step necessary for determining locations of the containing data blocks, which is instantiated during the recursive geometrical quadripartition process, whereas local index is constructed based on space filling curves and is utilized for specifying locations of spatial objects within a specific block. Both indexes aim at improving spatial data retrieval for Hadoop. This indexing scheme strongly supports simple spatial querying (selection queries, including range query for example). In addition, authors have designed a custom spatial-locality aware caching scheme that speeds up scanning stragglers (i.e., hotspots). In simple ways, it works by first applying a mathematical model to calculate most frequently accessed spatial objects. For those hotspot objects, the model caches in-memory sets of nearest objects, this significantly accelerates range-based queries.

RESQUE [65] has introduced on-the-fly spatial indexing method (or ODSI) that is utilized to assist spatial calculations, which significantly proves to enhance query response while introducing only tiny neglect-able overhead. Same authors in their similar work appeared in [4] support basic querying by employing the same ODSI scheme.

SpatialHadoop [31] supports basic spatial querying (such as range queries) by employing general-purpose standard spatial index structures such as grid, R- trees and R+ trees. They also design a MLI indexing scheme (local and global). In addition, they have implemented two modes for supporting range queries. The first mode is a no-replication mode (applies in case of R-tree) that encompasses two steps, step one applies a global filter (based on global index) for selecting blocks overlapping with query regions, where fully-satisfying blocks are copied as-is, while partially-satisfying blocks are sent to the second step for further processing. In the second step, a local filter (based on local index) operates on block granularity-level to filter tuples that satisfy query predicates. Thereafter results of the first and second step are combined in a global result set. The second mode (applies in cases of R+-tree and grid) is employed in case that some tuples are replicated throughout partitions. The main difference from the first mode is that, in the first step, blocks that fully satisfy range-query predicates need to be further processed for dealing with replicated records, while the second step includes a local filter that employs an additional replicate avoidance step to ensure that the global result is free of duplications. GISQF [67] are SpatialHadoop-based patches, thus features a similar level of query-ability as SpatialHadoop.

SHAHED [68] applies a MLI technique that constitutes three-layer indexes, grid, z-curves and quadtree. Grid and z-curves resembles a local index that accelerate query processing on a microscale (grid granularity-level), whereas quadtree is utilized to provide a performance boost on the macro-scale (inter-grid) which provides a global overview that accelerates processing of basic spatial queries such as selection queries (for example, range queries). Also, the query processor runs in three stages for better supporting range queries, temporal filter, spatial filter, and spatial refine. The process commences by applying a temporal filter that discards irrelevant partitions by only considering those that contain values within the specified temporal range, thereafter for each temporal partition, a spatial filter is applied to select tiles that fall within the required spatial range, where some fully satisfy the predicate (are completely copied with no further processing) while others partially satisfy, thus transferred to a spatial refinement step that is responsible for filtering spatial objects that overlaps the spatial range specified. In addition to those, SHAHED utilizes mechanisms already announced in SpatialHadoop for basic spatial query processing.

CoS-HDFS [69] utilizes the same indexing scheme that has been proposed by SpatialHadoop, aiming at providing a flat ground for processing basic spatial queries such as range and aggregation queries.

Hadoop-GIS [30] utilize a MLI indexing scheme comprising a global inter-partition-level index, and an intra-partition-level local ODSI index for efficient spatial basic querying operations. For example, in spatial containment queries, global index is utilized for pruning the search space by discarding irrelevant tiles that do not contribute to the query result. This is advantageous, since the size of global index is small, thus easily circulated across cluster processing elements with tiny communication overhead.

SATO [62] has designed a MLI region-based scheme that constitutes a local index for each tile and a global index for each partition aiming at enhancing MapReduce basic spatial queries. However, authors did not elaborate on the mechanism.

Some storage-oriented systems provided spatial indexing mechanisms for efficient basic spatial querying. For example, [73] applies a simple mechanism for supporting range-alike queries. It consists of providing a unique key for each range set, indexing the key which can then be used as a predicate in the range query. Also, it supports indexing geohashes (SCI indexing), which provides additional support for range scans. Aligned with this, HGrid [70] has incorporated a MLI indexing scheme (comprising of global index and fine-grained secondary index). This supports range-based queries as follows, first, a MBR is computed based on the range query predicates, thereafter z-ordering values of the relevant tiles are identified, which constitutes the primary indexing for the equivalent HBase rows, then secondary indices (corresponding to equivalent columns in HBase) are calculated based on grid cells (regular grid) overlapping with MBR covering range query space, concluding by merging subquery results in a global result set. The support for range-alike queries in [71] stems from the SCI indexing for a one-dimensional spatial coding (GeoSOT) they provide.

The discussion sketched so far has focused on basic spatial query support such as range queries. we now shift to more advanced querying. To keep things in the loop, we start by Spark-based, moving to Hadoop-based, and summing up by storage-oriented frameworks.

On top of Spark, SpatialSpark [32] has injected concise, yet effective, patches into Spark cores, aiming at supporting indexed spatial joins. One relation is indexed using R-trees and broadcasted to all processing elements of the cluster, thus providing a local view at each partition, thereafter local joins are performed on each partition by probing elements of the local relation with global relation, then the combination of local results constitute the global spatial join result, thus resembling a divide-and-conquer approach.

GeoSpark [48] introduces a spatial query processing layer incorporating a support for advanced spatial queries, such as spatial join and *K*NN. It was possible to optimize spatial join queries using local spatial on demand indexing techniques that are created for the SRDD involved in a spatial join query.

LocationSpark [60] supports complex queries including spatial join and *k*NN by a robust MLI indexing mechanism that incorporates a novel spatial index (sFilter). It plugged a mathematical model that trades off many alternatives, thereby selects best-matching partition-level execution plan. However, the authors did not elaborate on those mechanisms.

Cruncher [46] introduced inter-query optimizations for kNN queries that is based on OLI and ODSI indexes. Authors did not discuss how this facilitates *k*NN.

Stark [72] employs a module that supports complex transformations that span multiple datasets (spatial join and co-group), which is mainly supported by the guarantee of co-locality preservation, thus minimizing shuffling between participating nodes.

STARK [63] propose a novel algorithm for optimizing spatial *k*NN processing on top of Spark. The algorithm first performs conventional *k*NN locally for each partition of the processing cluster, thereafter in-ascending orders locals contributing in accordance with their distance from the focal point (i.e., query point). Afterwards, local results are joined, sorted in-ascending and a global set of *k*NN elements is selected accordingly. Also, they support spatial joins as follows: a Cartesian product is performed on two sides of

the join query (two RDDs), verifying if they match the join predicate, thus pruning them in case of no-match. On the other side, partitions that cross-match the query predicate is referenced, so that one partition is indexed using R-trees and broadcasted to other partitions for cross-matching against spatial join predicate, thereafter performing traditional local join, and concluding by combining local results into a global result set. As a more sophisticated spatial querying, STARK flattened the ground for the passage of clustering algorithms. They basically adapt [55] to efficiently work under Spark by applying their cost-aware custom partitioning method, that chiefly focuses on balancing loads, thus avoiding a congestion that may occur in straggler nodes of the processing cluster, which are a set of nodes that are overburdened with heavy workloads.

In [45] , AQWA propose an effective method for processing *K*NN. This method requires only one MapReduce job for a specific KNN query. They achieve this by employing some metrics, thus limiting the scanning process only to specific partitions that contain answers to the *K*NN query.

The work by [59] supports the processing of spatial join and *K*NN in a two-phased process (filter-refinement), where in the filter phase applies aggressive pruning to exclude objects that do not contribute toward query results, thereafter passes those to a refinement step to extract only those objects that satisfy query predicates. This is possible because of the support for special spatial indexing mechanisms within this framework.

Under the hood, RESQUE [65] supports spatial join as follows: it first employs R-trees for constructing indexes on all tiles of the grid, thereafter a spatial join component applies a MBR spatial join on two R-trees, afterwards applying a further refinement step to conclude objects that contribute to the results calculation. RESQUE also supports *k*NN queries on medical imaging. For example, finding *k*NN human cells in proximity with specific blood vessels. It also supports density-based clustering for answering queries like “finding human cells regions with density higher than or lower than a specific threshold”. However, the authors did not provide a discussion on how those supports are implemented apart from the support of an on-the-fly spatial indexing that may benefit in those cases.

SpatialHadoop [31] supports *k*NN by designing a custom algorithm consisting of three steps: the first step is an initialization step that computes k-nearest element locally (within the same partition as the focal point) by employing the conventional *k*NN algorithm locally. The second step is a correctness check in the sense that it builds a circle centered at the focal point (*k*NN query point). If the set of elements found in the first step fall only within the circle, they are considered as a result, otherwise a third step is needed, encompassing a refinement procedure that executes a range query for obtaining all points within a MBR bounding the circle, thereafter range query results are scanned to obtain final results contributing to *k*NN answer.

SpatialHadoop also provides a novel algorithm for processing spatial joins. It mainly comprises three steps; the first step is a global join, which constitutes calculating a full list of possible join results by specifying all overlapping MBRs, leveraging the global indexing scheme for each pair, then applying a conventional spatial join algorithm, subsequently a model is employed to combine each pair in a single split that is sent to the second step. In the second step, a local join is applied for which elements of the two blocks of a combined split are joined locally, producing a list of join records. This is basically engineered by exploiting the local indexing scheme while applying the Map function for locally joining records. This step introduces duplicate records which triggers the beginning of a third step, in which a reference-point duplicate avoidance technique is exploited for removing duplicate records before concluding the result. Being directly engineered on top of SpatialHadoop, SHAHED [68] , GISQF [67] utilizes SpatialHadoop mechanisms for supporting complex queries like spatial join and *k*NN.

CoS-HDFS [69] supports join because of its ability of preserving the co-location characteristic, thus data is indexed in a way that facilitates application of join predicates. Further, CoS-HDFS changed the partitioning strategy of Hadoop so that blocks that are potential to be joined are located on the same processing node. It also supports *k*NN by utilizing the capability from SpatialHadoop.

In Hadoop-GIS [30] , a MapReduce job is employed to perform spatial join, where in the Map phase, spatial objects of the same partition are paired, thereafter those are sent to a Reduce stage for performing local MBR based spatial joins on two datasets residing in the same partition utilizing an on-the-fly index.

SATO [62] applies a multi-level sampling approach for an improved spatial join experience. The approach is mainly ruled out by a changeable control sampling ratio, which can be tweaked depending on the query workload, so that spatial objects that are more frequently accessed by spatial joins are sampled with an elevated ratio.

From the storage-oriented frameworks, perhaps most significantly is that MD-HBase [39] supports *k*NN Query by an algorithm that proceeds as follows. The algorithm incrementally expands search region and subsequently sort the enclosed subspaces in an ascending order relative to the query point. Thereafter, the algorithm scans the closest subspace that has not been visited before and accordingly sorts spatial objects in relation to their distance to the focal point. If the algorithm finds *k* points such that the distance to the last point is less than the nearest waiting subspace, the query returns the result.

It is becoming apparent that most methods depend on the filter-refinement (patterned after true-hit filtering approach [74]) approach for spatial join processing. This is attributed to the fact that performing Cartesian product spatial join is specifically resource-intensive. It is also clear from the taxonomy in table 4 that storage-oriented systems do not focus on providing optimizations for complex spatial query processing. One reason for this consensus is that processing disk-resident datasets entail a high tax on the processing system because of the I/O communication overhead. It is also clear that even though Spark has mostly replaced Hadoop, some works still focus on the retiring system harnessing its powers in the geospatial multidimensional analytics for IoT.

Table 4. QoS-aware optimizations aspects analysis for big geospatial data management

Paradigm	Authors, year	Spatial-aware partitioning aspects[1]			Basic spatial query optimizations					Spatial queries		
		LB	SDL	BSO	Spatial-aware caching	Spatial-aware indexing				DBC	SJ	KNN
						MLI	SCI	OLI	ODSI			
Spark - based	SpatialSpark [32] , 2015	✓	✓	X	X	X	✓	X	X	X	✓	X
	Magellan Spark [75] , 2015	✓	✓	X	✓	X	✓	X	✓	X	✓	✓
	LocationSpark [60] , 2016	✓	✓	X	✓	✓	X	X	X	✓	✓	✓
	Cruncher [46] , 2016	✓	✓	X	✓	X	X	✓	✓	X	✓	✓
	Geospark [48] , 2016	✓	X	✓	✓	✓	✓	✓	X	X	✓	✓
	Stark [72] , 2017	✓	✓	X	X	X	X	X	X	X	✓	X
	STARK [63] ,	✓	✓	X	X	X	X	✓	✓	✓	✓	✓

[1] ✓: satisfied, X: not satisfied.

	2017											
	SparkGIS [76] , 2017	✓	✓	X	✓	✓	X	✓	✓	X	✓	✓
	Simba [77] , 2016	✓	✓	X	✓	✓	X	✓	X	X	✓	✓
	[43, 44] , 2017, 2018	✓	✓	✓	✓	X	X	✓	X	✓	X	✓
Hadoop-based	[59] , 2012	✓	✓	X	✓	X	X	✓	X	X	✓	✓
	RESQUE [65] , 2012	X	✓	✓	X	X	X	X	✓	✓	✓	✓
	[4] , 2012	✓	✓	✓	X	X	X	X	✓	✓	✓	✓
	AQWA [45] ,2015	✓	✓	X	✓	X	X	X	X	X	X	✓
	AEGIS [66] , 2015	✓	✓	X	X	X	✓	X	X	X	X	X
SpatialHadoop-based	GISQF [67] , 2014	✓	X	X	X	✓	X	X	X	X	✓	✓
	SpatialHadoop [31] , 2015	✓	✓	✓	X	✓	X	X	X	X	✓	✓
	SHAHED[68] , 2015	✓	✓	X	X	✓	X	X	X	X	✓	✓
	CoS-HDFS [69] , 2016	✓	✓	X	X	✓	X	X	X	X	✓	✓
Hadoop-GIS-based	SATO [62] , 2014	✓	X	✓	X	✓	X	X	X	X	✓	X
	Hadoop-GIS [30] , 2015	✓	X	✓	X	✓	X	X	✓	X	✓	✓
storage-oriented	HGrid [70] , 2013	✓	✓	X	X	✓	X	X	X	X	X	✓
	[71] , 2015	✓	✓	X	X	X	✓	X	X	X	X	X
	[73] , 2017	✓	✓	X	X	X	✓	✓	X	X	X	X
	MD-HBase [39] , 2011	✓	✓	X	X	✓	✓	X	X	X	X	✓
	MongoDB-based [29] , 2018	✓	✓	X	✓	✓	✓	✓	✓	✓	✓	✓

5.3 Discussions and Performance Evaluations

Relevant works of the literature have mainly focused on providing optimizations for IoT data in Cloud in two veins: spatial partitioning and query optimizers. The main aim is minimizing the network shuffling overheads in Cloud deployments, by adopting spatial-aware data management frameworks that encompass methods which collaborate synergistically in achieving that broad goal. Also implies within the same vein a goal of maximizing Cloud resource utilization and cutting the costs on the user. Studies have shown that the average utilization in cloud deployments is under 40 percent of the overall reserved resources [78, 79]. This is possibly due to the fact that users lack the relevant understanding on how to configure the auto-scaling parameters (which requires technical knowledge for most SPEs) that, in its turn, behooves them to select lenient configurations that allow, most often, the overprovisioning in order to handle peak loads , leading then to a low resource utilization. The indirect effect that spatial data management plays on the Cloud networks and IoT data is prevalent.

An important performance note is that partitioning is not about creating so many chunks as this is detrimental to query performance. To this end, efficient systems have focused on building smart partitioning methods that account for this. Despite their disparities, most works have focused on providing an acceptable degree of balance among partitioning goals, including BSO minimization, SDL preservation and load balancing. However, those requirements are greatly contradicting, therefore the domain-specific fixes and patches provided by current frameworks are not general-purpose in the sense that their exposure to diverse application scenarios may reveal their weaknesses and deteriorate their goals. In addition, most of the works have concentrated on Hadoop, neglecting the fact that Hadoop ecosystem has been engineered to specifically process immutable files, deeming it unsuitable for read-intensive applications. Therefore, systems embarking on Spark for managing big spatial workloads significantly outperform their Hadoop-based counterparts, and more works are encouraged on top of Spark, harnessing its robustness for spatial-oriented diverse-domain workloads.

Regarding the patches provided for an optimized query performance, studies mostly geared their attention toward spatial indexing, mostly neglecting spatial-aware caching. Perhaps motivated by the fact that in most cases of spatial indexing, the cost of creating the index carries only tiny overhead fraction, while significantly enhances query overall performance. However, one should consider that spatial indexing is a mean-to-an-end, aiming at accelerating spatial queries. The interest in spatial-oriented join is absent in this consortium, if for no other reason than the advisability that caching is resource-intensive, tending to wreak havoc on any bare metal commodity resources, especially for compute-intensive repetitive structures (kNN join for example). While this applies for centralized systems, it need not be the case in the existence of cloud-based systems, which offer cost-effective elastic solutions where all processing elements of the computing cluster collaborate in every single point of optimization, including the build of distributed parallel caching, where main memories of all participating machines are contributing in the caching process, making it inexpensive and yet efficient.

We notice that the interest in supporting spatial querying capabilities have gained an increased momentum of interest in the last decade or so. Perhaps geared by the fact that most interesting real-life query scenarios in today's businesses encapsulates intrinsically complex querying (for example, spatial join and *k*NN) for discovering interesting patterns that facilitates decision making. However, current systems need to give more awareness to spatial partitioning challenges, including BSOs, SDL and load balancing. One of those challenges significantly challenges systems resources and diminishes the benefits of parallelization, therefore all traditional working algorithms are to consider those challenges while adapting themselves to work in parallel computing environments and their accountabilities are to aim for a weighted balance for those.

6. Open Issues and Recommended Future Research Frontiers

We here recommend some of the QoS- and spatial-aware optimization directions for big geospatial datasets. During our exhaustive review of the literature, we have noticed that those issues have been mainly left unanswered. However, we believe that tackling those issues potentially boosts SDMEs performances for most common spatially-loaded scenarios.

6.1 Weighted Balance to Satisfy Partitioning Requirements

Despite the huge efforts in advancing emerging techniques for QoS-aware big geospatial data management for the benefit of better utilizing Cloud resources in managing georeferenced IoT data, various issues remained unanswered. Perhaps most significantly is the question of how-to better tradeoff the three main requirements for a spatial-aware partitioning in a way that guarantees an acceptable error-bounded loss of accuracy in exchange with a reduced latency. Most methods of the literature have focused on some of the partitioning goals, while trading-off the three together has been left often untouched. We encourage a future work that communicates a weighted balance between spatial partitioning challenges in a way that is general enough to be applied in various highly dynamic application scenarios with huge amounts of IoT data.

6.2 Online Locality Preservation and BSOs

The fast speed of spatial data arrival rate easily drains a spatial processing engine (SPE) resources and challenges its capacity as it quickly becomes laborious. Most works of the literature have focused on providing optimizations for the offline QoS-aware big spatial data management mode and have proved order of magnitude improvements compared to baseline frameworks. However, today many interesting applications, depending on avalanches of continuously arriving IoT data, seek to process an online deluge of big geospatial data streams that normally arrive very fast and continue to grow in an unbounded fashion. In addition, most queries seek proximity-related answers (*k*NN and spatial join). Preserving a pairwise relationship seems interesting for a SPE to withstand during aggressively burst workloads. However, this kind of computations while disseminating fast arriving spatial datasets is specifically not a simple matter. To the best of our knowledge, algorithmic complexity-wise, there are only few algorithms for preserving spatial locality online (such as the work by [1]). This is attributed to the costly calculations by checking spatial co-location. In this context, the linear-time-complexity naïve algorithms are inacceptable. Hence, a layer that bundles such capability is desired in this domain, abstracting the underlying complexities of such handling and offering a flattened ground for front-end developers so that they focus on analyzing data rather than handling congestions. To such an end, among initial considerations to achieve this is finding a compute-tractable approximate sub-linear or even constant-complexity method that preserves SDL online. However, the cost of preserving SDL should be amortized by an improved querying experience on the long run.

6.3 Full-fledged QoS-aware Library for Big Geospatial Data Management for IoT

Accountabilities for QoS-awareness in big geospatial management systems for IoT in Cloud are often patch efforts and domain specific fixes, leaving the handling of many logistics to the front-end developers, which encompasses much of a burden and counteracts the benefits of underlying systems. It is therefore necessary to develop a full-fledged library that helps ease the developers burden by offloading those logistics to lower tiers and incorporate them transparently so that the programmer does not have to reason about them atomically, and therefore give them exposure to a diversity of application domains and poise them to take a central position in modern spatial database management systems.

7. Conclusive Remarks

This survey contemplates taxonomy of big spatial data management strategies for IoT in the Cloud. Novel strategies in both veins (storage and processing-oriented) have been contemplated in two directions (partitioning and query optimizers). Their support for a set of aggressive spatial-aware partitioning requirements is elaborated. Traditional common spatial partitioning methods behave favorably for uniform spatial workloads. However, for highly skewed spatial data loads coming from heterogeneous IoT sources, their performance degrades significantly. This motivated a constellation of researchers to design novel spatial-aware big data management optimizations for IoT in the Cloud, focusing mainly on two aspects, partitioning and query optimization. For example, some partitioning methods aim at preserving SDL, while maintaining a fair load balance throughout data partitions. Other methods focused on minimizing BSOs, providing an opportunity to boost the performance of some data mining algorithms (for example, density-based clustering). To this end, most relevant works of the literature have been reviewed and a cross-matching have been conducted to map those works to their relevant classes of the taxonomy. We aimed at providing a comprehensive comparative taxonomy that enables researchers to advance the domain. An in-line discussion has been drawn throughout sparse parts of the survey, discussing the capability of each optimization in relation to some of the challenges (for example, partitioning challenges). We have also highlighted other avenues for investigation in this road.

What's more, most of the works of the relevant literature have focused only on some classes of optimization, neglecting the others. Seemingly, reasons are testified by the fact that it is challenging in big spatial context to consider contradicting requirements, where optimizing one diminishes the benefits of others. For example, considering SDL preservation often leaves data partitions lopsided, causing load imbalance. We believe that a future work is to consider a weighted balance of all trade-offs involved in big geospatial data management. Also, most works have focused on processing-oriented as most operations in

parallel geospatial data management are predominantly read operations, characterizing those systems by being read-intensive applications.

To sum up, a wealth of spatial-aware frameworks is built on top of Hadoop. However, Hadoop has not been designed specifically for read-intensive applications, which is a fundamental flaw that makes Hadoop-based systems less interesting than their Spark-based counterparts. Spark excels in the field and continues as such for the foreseeable future. However, despite that their ability to outperform Hadoop-based systems by factors of magnitude, Spark-based systems require considerable efforts before reaching their tipping points.

Most works in the relevant literature have focused on networking/hardware aspects for IoT in Cloud, Edge and Fog. Some works have focused on general-purpose big data management for IoT. To the best of our knowledge, this survey paper constitutes a unique effort that focuses on spatial data management aspects for IoT in Cloud.

Acknowledgments. This research was supported by the SACHER (Smart Architecture for Cultural Heritage in Emilia Romagna) project funded by the POR-FESR 2014-20 (no. J32I16000120009) through CIRI.

References

[1] I. M. Al Jawarneh, P. Bellavista, L. Foschini and R. Montanari, "Spatial-aware approximate big data stream processing," in *2019 IEEE Global Communications Conference (GLOBECOM),* 2019, pp. 1-6.

[2] I. M. Aljawarneh, P. Bellavista, C. R. De Rolt and L. Foschini, "Dynamic identification of participatory mobile health communities," in *Cloud Infrastructures, Services, and IoT Systems for Smart Cities*Anonymous Springer, 2017, pp. 208-217.

[3] S. S. Sahoo, A. Wei, C. Tatsuoka, K. Ghosh and S. D. Lhatoo, "Processing neurology clinical data for knowledge discovery: Scalable data flows using distributed computing," in *Machine Learning for Health Informatics*Anonymous Springer, 2016, pp. 303-318.

[4] A. Aji, F. Wang and J. H. Saltz, "Towards building a high performance spatial query system for large scale medical imaging data," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems,* 2012, pp. 309-318.

[5] E. Gomes, M. A. Dantas, D. D. de Macedo, C. De Rolt, M. L. Brocardo and L. Foschini, "Towards an infrastructure to support big data for a smart city project," in *2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE),* 2016, pp. 107-112.

[6] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, I. M. Al Jawarneh and A. Zanni, "How Fog Computing Can Support Latency/Reliability-Sensitive IoT Applications: An overview and a Taxonomy of State-Of-The-Art Solutions," 2019.

[7] R. R. Vatsavai, A. Ganguly, V. Chandola, A. Stefanidis, S. Klasky and S. Shekhar, "Spatiotemporal data mining in the era of big spatial data: Algorithms and applications," in *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data,* 2012, pp. 1-10.

[8] A. Botta, W. De Donato, V. Persico and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future Generation Comput. Syst.,* vol. 56, pp. 684-700, 2016.

[9] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini and A. Zanni, "A survey on fog computing for the Internet of Things," *Pervasive and Mobile Computing,* vol. 52, pp. 71-99, 2019.

[10] I. M. Al Jawarneh, P. Bellavista, L. Foschini and R. Montanari, "Spatial-aware approximate big data stream processing," in *IEEE Global Communications Conference, GLOBECOM,* 2020, .

[11] K. E. Jones, N. G. Patel, M. A. Levy, A. Storeygard, D. Balk, J. L. Gittleman and P. Daszak, "Global trends in emerging infectious diseases," *Nature,* vol. 451, *(7181),* pp. 990-993, 2008.

[12] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini and A. Zanni, "A survey on fog computing for the Internet of Things," *Pervasive and Mobile Computing,* 2018.

[13] M. Ge, H. Bangui and B. Buhnova, "Big data for internet of things: a survey," *Future Generation Comput. Syst.,* vol. 87, pp. 601-614, 2018.

[14] E. Siow, T. Tiropanis and W. Hall, "Analytics for the internet of things: A survey," *ACM Computing Surveys (CSUR),* vol. 51, *(4),* pp. 1-36, 2018.

[15] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud,* vol. 10, *(10-10),* pp. 95, 2010.

[16] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The hadoop distributed file system." in *Msst,* 2010, pp. 1-10.

[17] S. Bradshaw and K. Chodorow, *Mongodb: The Definitive Guide: Powerful and Scalable Data Storage, 3rd Edn.* O'Reilly Media Inc, USA, 2018.

[18] K. Banker, *MongoDB in Action.* Manning Publications Co., 2011.

[19] J. Yu, Z. Zhang and M. Sarwat, "Spatial data management in apache spark: The geospark perspective and beyond," *GeoInformatica,* vol. 23, *(1),* pp. 37-78, 2019.

[20] R. Khan, S. U. Khan, R. Zaheer and S. Khan, "Future internet: The internet of things architecture, possible applications and key challenges," in *2012 10th International Conference on Frontiers of Information Technology,* 2012, pp. 257-260.

[21] D. C. Tsichritzis and F. H. Lochovsky, "Hierarchical data-base management: A survey," *ACM Computing Surveys (CSUR),* vol. 8, *(1),* pp. 105-123, 1976.

[22] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS Operating Systems Review,* vol. 41, *(6),* pp. 205-220, 2007.

[23] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review,* vol. 44, *(2),* pp. 35-40, 2010.

[24] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS),* vol. 26, *(2),* pp. 1-26, 2008.

[25] A. H. Team, "Apache hbase reference guide," *Apache, Version,* vol. 2, *(0),* 2016.

[26] K. Grolinger, W. A. Higashino, A. Tiwari and M. A. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications,* vol. 2, *(1),* pp. 22, 2013.

[27] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun ACM,* vol. 51, *(1),* pp. 107-113, 2008.

[28] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management,* vol. 23, *(3),* pp. 567-619, 2015.

[29] I. M. Al Jawarneh, P. Bellavista, F. Casimiro, A. Corradi and L. Foschini, "Cost-effective strategies for provisioning NoSQL storage services in support for industry 4.0," in *2018 IEEE Symposium on Computers and Communications (ISCC),* 2018, pp. 1227.

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

[31] A. Eldawy and M. F. Mokbel, "Spatialhadoop: A mapreduce framework for spatial data," in *2015 IEEE 31st International Conference on Data Engineering,* 2015, pp. 1352-1363.

[32] S. You, J. Zhang and L. Gruenwald, "Large-scale spatial join query processing in cloud," in *2015 31st IEEE International Conference on Data Engineering Workshops,* 2015, pp. 34-41.

[33] J. L. Bentley and J. H. Friedman, "Data structures for range searching," *ACM Computing Surveys (CSUR),* vol. 11, *(4),* pp. 397-409, 1979.

[34] D. E. Knuth, "The art of computer programming: Sorting and Searching, 2nd edn., vol. 3," 1998.

[35] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica,* vol. 4, *(1),* pp. 1-9, 1974.

[36] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun ACM,* vol. 18, *(9),* pp. 509-517, 1975.

[37] T. K. Sellis, N. Roussopoulos and C. Faloutsos, "The R -tree: A dynamic index for multi-dimensional objects," in *Proceedings of the 13th International Conference on very Large Data Bases,* 1987, pp. 507-518.

[38] H. Sagan, "Space-Filling Curves. Springer-Verlag, 1994." 1994.

[39] S. Nishimura, S. Das, D. Agrawal and A. El Abbadi, "Md-hbase: A scalable multi-dimensional data infrastructure for location aware services," in *2011 IEEE 12th International Conference on Mobile Data Management,* 2011, pp. 7-16.

[40] H. Fuchs, Z. M. Kedem and B. F. Naylor, "On visible surface generation by a priori tree structures," in *ACM Siggraph Computer Graphics,* 1980, pp. 124-133.

[41] S. T. Leutenegger, M. A. Lopez and J. Edgington, "STR: A simple and efficient algorithm for R-tree packing," in *Proceedings 13th International Conference on Data Engineering,* 1997, pp. 497-506.

[42] T. Asano, D. Ranjan, T. Roos, E. Welzl and P. Widmayer, "Space-filling curves and their use in the design of geometric data structures," *Theor. Comput. Sci.,* vol. 181, *(1),* pp. 3-15, 1997.

[43] I. M. Aljawarneh, P. Bellavista, A. Corradi, R. Montanari, L. Foschini and A. Zanotti, "Efficient spark-based framework for big geospatial data query processing and analysis," in *2017 IEEE Symposium on Computers and Communications (ISCC),* 2017, pp. 851-856.

[44] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, R. Montanari and A. Zanotti, "In-memory spatial-aware framework for processing proximity-alike queries in big spatial data," in *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD),* 2018, pp. 1-6.

[45] A. M. Aly, A. R. Mahmood, M. S. Hassan, W. G. Aref, M. Ouzzani, H. Elmeleegy and T. Qadah, "AQWA: adaptive query workload aware partitioning of big spatial data," *Proceedings of the VLDB Endowment,* vol. 8, *(13),* pp. 2062-2073, 2015.

[46] A. S. Abdelhamid, M. Tang, A. M. Aly, A. R. Mahmood, T. Qadah, W. G. Aref and S. Basalamah, "Cruncher: Distributed in-memory processing for location-based services," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE),* 2016, pp. 1406-1409.

[47] A. Eldawy, L. Alarabi and M. F. Mokbel, "Spatial partitioning techniques in SpatialHadoop," *Proceedings of the VLDB Endowment,* vol. 8, *(12),* pp. 1602-1605, 2015.

[48] J. Yu, J. Wu and M. Sarwat, "Geospark: A cluster computing framework for processing large-scale spatial data," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems,* 2015, pp. 70.

[49] S. Amini, I. Gerostathopoulos and C. Prehofer, "Big data analytics architecture for real-time traffic control," in *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS),* 2017, pp. 710-715.

[50] H. Abdelhaq and M. Gertz, "On the locality of keywords in twitter streams," in *Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming,* 2014, pp. 12-20.

[51] E. H. Jacox and H. Samet, "Spatial join techniques," *ACM Transactions on Database Systems (TODS),* vol. 32, *(1),* pp. 7, 2007.

[52] H. Kriegel, P. Kröger, J. Sander and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 1, *(3),* pp. 231-240, 2011.

[53] M. Ester, H. Kriegel, J. Sander and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd,* 1996, pp. 226-231.

[54] B. Dai and I. Lin, "Efficient map/reduce-based dbscan algorithm with optimized data partition," in *2012 IEEE Fifth International Conference on Cloud Computing,* 2012, pp. 59-66.

[55] Y. He, H. Tan, W. Luo, S. Feng and J. Fan, "MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data," *Frontiers of Computer Science,* vol. 8, *(1),* pp. 83-99, 2014.

[56] R. Xu and D. Wunsch, *Clustering.* John Wiley & Sons, 2008*10*.

[57] W. Wang, J. Yang and R. Muntz, "PK-tree: A spatial index structure for high dimensional point data," in *Information Organization and Databases*Anonymous Springer, 2000, pp. 281-293.

[58] A. Aji and F. Wang, "High performance spatial query processing for large scale scientific data," in *Proceedings of the on SIGMOD/PODS 2012 PhD Symposium,* 2012, pp. 9-14.

[59] Y. Zhong, X. Zhu and J. Fang, "Elastic and effective spatio-temporal query processing scheme on hadoop," in *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data,* 2012, pp. 33-42.

[60] M. Tang, Y. Yu, W. G. Aref, A. R. Mahmood, Q. M. Malluhi and M. Ouzzani, "Locationspark: In-memory distributed spatial query processing and optimization," *CoRR,* pp. 1-15, 2019.

[61] J. Yu, Z. Zhang and M. Sarwat, "Spatial data management in apache spark: The geospark perspective and beyond," *GeoInformatica,* vol. 23, *(1),* pp. 37-78, 2019.

[62] H. Vo, A. Aji and F. Wang, "SATO: A spatial data partitioning framework for scalable query processing," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems,* 2014, pp. 545-548.

[63] S. Hagedorn, P. Gotze and K. Sattler, "The STARK framework for spatio-temporal data analytics on spark," *Datenbanksysteme Für Business, Technologie Und Web (BTW 2017),* 2017.

[64] R. Giachetta, "A framework for processing large scale geospatial and remote sensing data in MapReduce environment," *Comput. Graph.,* vol. 49, pp. 37-46, 2015.

[65] A. Aji and F. Wang, "High performance spatial query processing for large scale scientific data," in *Proceedings of the on SIGMOD/PODS 2012 PhD Symposium,* 2012, pp. 9-14.

[66] R. T. Whitman, M. B. Park, S. M. Ambrose and E. G. Hoel, "Spatial indexing and analytics on hadoop," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems,* 2014, pp. 73-82.

[67] K. M. Al Naami, S. Seker and L. Khan, "GISQF: An efficient spatial query processing system," in *2014 IEEE 7th International Conference on Cloud Computing,* 2014, pp. 681-688.

[68] A. Eldawy, M. F. Mokbel, S. Alharthi, A. Alzaidy, K. Tarek and S. Ghani, "Shahed: A mapreduce-based system for querying and visualizing spatio-temporal satellite data," in *2015 IEEE 31st International Conference on Data Engineering,* 2015, pp. 1585-1596.

[69] M. M. Fahmy, I. Elghandour and M. Nagi, "CoS-HDFS: Co-locating geo-distributed spatial data in hadoop distributed file system," in *2016 IEEE/ACM 3rd International Conference on Big Data Computing Applications and Technologies (BDCAT),* 2016, pp. 123-132.

[70] D. Han and E. Stroulia, "Hgrid: A data model for large geospatial data sets in hbase," in *2013 IEEE Sixth International Conference on Cloud Computing,* 2013, pp. 910-917.

[71] Z. Weixin, Y. Zhe, W. Lin, W. Feilong and C. Chengqi, "The non-sql spatial data management model in big data time," in *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS),* 2015, pp. 4506-4509.

[72] S. Li, M. T. Amin, R. Ganti, M. Srivatsa, S. Hu, Y. Zhao and T. Abdelzaher, "Stark: Optimizing in-memory computing for dynamic dataset collections," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS),* 2017, pp. 103-114.

[73] K. Zheng, D. Gu, F. Fang, M. Zhang, K. Zheng and Q. Li, "Data storage optimization strategy in distributed column-oriented database by considering spatial adjacency," *Cluster Computing,* vol. 20, *(4),* pp. 2833-2844, 2017.

[74] T. Brinkhoff, H. Kriegel, R. Schneider and B. Seeger, *Multi-Step Processing of Spatial Joins.* ACM, 1994*23(2).*

[75] R. Sriharsha, "Magellan: geospatial analytics on spark," *Retrieved May,* vol. 1, pp. 2018, 2015.

[76] F. Baig, H. Vo, T. Kurc, J. Saltz and F. Wang, "Sparkgis: Resource aware efficient in-memory spatial query processing," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems,* 2017, pp. 1-10.

[77] D. Xie, F. Li, B. Yao, G. Li, L. Zhou and M. Guo, "Simba: Efficient in-memory spatial analytics," in *Proceedings of the 2016 International Conference on Management of Data,* 2016, pp. 1071-1085.

[78] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing,* 2012, pp. 7.

[79] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," in *ACM SIGARCH Computer Architecture News,* 2014, pp. 127-144.

Isam Mashhour Al Jawarneh is a research Assistant and Ph.D. student at the Computer Science and Engineering Department (DISI) of the University of Bologna, Italy. His research interests cover many aspects of big data stream processing and active data warehousing for highly dynamic application scenarios. He has authored/co-authored many international journal articles and papers for flagship conferences (such as IEEE GLOBECOM and ICC). He has a research and teaching experience at higher-education level for more than 12 years.

Paolo Bellavista (SM'06) received MSc and PhD degrees in computer science engineering from the University of Bologna, Italy, where he is now a full professor of distributed and mobile systems. His research activities span from pervasive wireless computing to location/context-aware services, from edge cloud computing to middleware for Industry 4.0 applications. He is currently the scientific coordinator of a large H2020 big data innovation action called IoTwins about distributed digital twins for the manufacturing industry. He serves on the Editorial Boards of IEEE Communications Surveys and Tutorials, IEEE T. on Network and Service Management, Elsevier Pervasive Mobile Computing, Elsevier Journal on Network and Computing Applications, and Springer Journal of Network and Systems Management.

Antonio Corradi (SM'19) graduated from University of Bologna, Italy, and received MS in electrical engineering from Cornell University, USA. He is a full professor of computer engineering at the University of Bologna. His research interests include distributed systems, middleware for pervasive and heterogeneous computing, infrastructure for ser-vices and network management.

Luca Foschini (SM'19) graduated from the University of Bologna, Italy, where he received a Ph.D. degree in computer science engineering in 2007. He is now an associate professor of computer engineering at the University of Bologna. His interests span from integrated management of distributed systems and services to wireless pervasive computing and

scalable context data distribution infrastructures and context-aware services. Currently, he is working on mobile crowdsensing and crowdsourcing and management of Cloud systems for Smart City environments.

Rebecca Montanari graduated from the University of Bologna, where she received a Ph.D. degree in computer science engineering in 2001. She is now an associate professor of computer engineering at the University of Bologna. Her research primarily focuses on semantic-based middleware supports for service provisioning, context-aware services, security solutions for pervasive environments, policy-based service management, and adaptive and scalable middleware solutions for system and service management.