International Journal of Geo-Information

MDPI

Article

AHS Model: Efficient Topological Operators for a Sensor Web Publish/Subscribe System

Chih-Yuan Huang [1,*] and Steve H. L. Liang [2]

[1] Center for Space and Remote Sensing research, National Central University, Taoyuan 320, Taiwan
[2] Geomatics Engineering, University of Calgary, Calgary, AB T2N 1N4, Canada; steve.liang@ucalgary.ca
* Correspondence: cyhuang@csrsr.ncu.edu.tw; Tel.: +886-3-4227151 (ext. 57692)

Academic Editors: Jason K. Levy and Wolfgang Kainz
Received: 7 December 2016; Accepted: 13 February 2017; Published: 20 February 2017

Abstract: The Worldwide Sensor Web has been applied for monitoring the physical world with spatial and temporal scales that were impossible in the past. With the development of sensor technologies and interoperable open standards, sensor webs generate tremendous volumes of priceless data, enabling scientists to observe previously unobservable phenomena. With its powerful monitoring capability, the sensor web is able to capture time-critical events and provide up-to-date information to support decision-making. In order to harvest the full potential of the sensor web, efficiently processing sensor web data and providing timely notifications are necessary. Therefore, we aim to design a software component applying the publish/subscribe communication model for the sensor web. However, as sensor web data are geospatial in nature, existing topological operators are inefficient when processing a large number of geometries. This paper presents the Aggregated Hierarchical Spatial Model (AHS model) to efficiently determine topological relationships between sensor data and predefined query objects. By using a predefined hierarchical spatial framework to index geometries, the AHS model can match new sensor data with all subscriptions in a single process to improve the query performance. Based on our evaluation results, the query latency of the AHS model increases 2.5 times more slowly than that of PostGIS. As a result, we believe that the AHS model is able to more efficiently process topological operators in a sensor web publish/subscribe system.

Keywords: worldwide sensor web; publish/subscribe; topological operator

1. Introduction

In recent years, large-scale sensor arrays and the vast datasets produced worldwide have been utilized, shared, and published by a rising number of data providers. With open standards defining data schemas and web service interfaces, such as the Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) standards, sensors, and their data can be integrated in an interoperable manner, which has become the main idea behind the Worldwide Sensor Web [1–3]. This Worldwide Sensor is capable of monitoring the physical world with spatial and temporal scales that were previously impossible.

The powerful monitoring capabilities of the sensor web technologies are increasingly attracting the interest of researchers for a wide range of applications, including large-scale environmental monitoring [4–6], civil structures [7], roadways [8,9], and animal habitats [10,11]. Some of the applications are time-critical and require efficient data processing for timely decision-making and notification, such as emergency response systems [12]. These time-critical applications may be used to support decision-making when handling time-critical events, such as the 2007 Minneapolis Interstate-35W bridge collapse and the 2011 Japan earthquake and tsunami.

As the worldwide sensor web's vision is to connect various types of sensors that are located worldwide and perform high-frequency observations, it could have the ability to capture these time-critical events and provide up-to-date information to support decision-making. We believe that by efficiently converting sensor web data streams into information and providing timely notifications, we can lower or even prevent the damage caused by time-critical disasters.

Apart from the disaster management system, many applications would also require efficient sensor data stream processing and timely notifications, such as house and human security, illegal activity detection, early warning system, location-based services, health monitoring, etc. For example, potential queries are (1) "send me a notification if the temperature in my kitchen is higher than 50 degree Celsius"; (2) "send me a notification if my child's GPS location leaves the school area"; (3) "send me a notification if any water level sensor within 1 km radius of my house reports a reading higher than 1 meter" and (4) "send customers notifications about discounts when they are within 1 km radius of stores". These queries all require efficient processing to notify users for taking action accordingly and in time.

However, the traditional request/response communication model is not suitable for processing continuous data streams as it is based on point-to-point pulling interaction and not designed for rapid and continuous data streams [13]. In a system following the request/response model (i.e., pull-based system), the response is evaluated with a snapshot of the system. However, no predictions of when an event will happen can be made (e.g., starting of a fire, collapse of a bridge, or simply any observation). A communication cannot be scheduled ahead of time. By the time a user sends requests, events may already be out of date.

An alternative communication model is known as the publish/subscribe model (or continuous query processing model). The publish/subscribe model allows users to first register queries in a system (i.e., predefined queries). Whenever the system receives new data, it executes the predefined queries and sends the results to users as notifications. In this way, the publish/subscribe model can provide notifications more promptly than the request/response model.

In a system following the publish/subscribe model (i.e., push-based systems), a typical approach to performing a continuous query is to first create a query execution plan [13], which consists of operators and queues. All new data will traverse through the query plan. While some approaches have been proposed to optimize the overall structure of query plans [14,15], there was not much discussion of how to improve the efficiency of computational-intensive geospatial operators in a push-based system.

While many general-purpose operators are simple and efficient enough to be used directly in query execution plans (e.g., elementary arithmetic, average, count, sum), some geospatial operators are complex and time-consuming. As sensor web data are geospatial in nature, geospatial operators are common in many monitoring applications, such as the aforementioned example queries. For this research, we focus on the topological operators, which are used to determine the geospatial relationships defined in the OGC Simple Feature Access specification [16].

Although the answers to the aforementioned example queries could be obtained by database topological queries, topological operators are usually time-consuming and could create a performance bottleneck when processing a large number of geometries [17]. In the context of a sensor web publish/subscribe system, we face the major sensor web data challenges and opportunities [3]: the large volume and large generation velocity of sensor data make it difficult to manage, but the sensor web's powerful monitoring capability can be extremely valuable in various applications. Therefore, we expect that processing topological queries with big sensor web data and a large number of subscriptions will create a performance bottleneck.

Hence, the aim of this paper is to propose a new approach to efficiently process topological operators in a sensor web publish/subscribe system. It presents the Aggregated Hierarchical Spatial Model (AHS model) for processing topological queries based on the nature of continuous query processing. The AHS model encompasses two key ideas. Firstly, as queries are predefined and continuous in a publish/subscribe system, the AHS model pre-generates the necessary indices for geometries of subscriptions and reuses them whenever new data arrive. Secondly, by indexing

geometries of subscriptions with the same indexing structure, we can aggregate the subscription indices. In this case, we not only can reduce the space for storing the pre-generated indices, but can also intersect new data with all subscriptions in a single process.

In addition, we are also proposing a distributed computing version of the AHS model (i.e., a distributed AHS model). With the advancements in distributed computing [18] and cloud computing techniques (e.g., Amazon Elastic Compute Cloud), we designed the distributed AHS model to scale horizontally for more computation and storage resources.

In summary, this paper presents the following:

1. The AHS model—a model that can efficiently determine the topological relationship between the geometries in a sensor web publish/subscribe system. As a sensor web publish/subscribe system aims to detect and manage time-critical events in a timely manner, the AHS model can be a critical component in such a system.
2. The incorporation of a predefined hierarchical spatial framework to index and aggregate subscription geometries, which allows the AHS model to intersect a publication with all subscriptions in a single process. In addition, all the necessary subscription indices are pre-generated and reused to reduce the runtime processing cost.
3. A distributed AHS model that can scale horizontally to store the pre-generated indices and improve query performance. By connecting more machines, the distributed AHS model is able to distribute the index storage and also process queries in parallel.
4. Evaluation of the AHS model in terms of its scalability, indexing performance, matching performance, and end-to-end query latency. The evaluation results show that the AHS model is more scalable than PostGIS, and the distributed AHS model can attain satisfying end-to-end performance with a relatively realistic dataset.

Before presenting the details, we first define the subscriptions and publications in a sensor web publish/subscribe system. In general, subscriptions are continuous queries registered by users, and publications are the sensor data produced by sensors. As sensor data are geospatial in nature, both subscriptions and publications have geospatial components. A subscription (*SUB*) can have different predicates as query criteria/filters, among which the spatial predicate in a subscription (SUB_{SP}) has two parameters: a base geometry (SUB_{SP_GEO}) and a topological operator (SUB_{SP_OPER}). Users set these two parameters to select publications with a geometry (PUB_{GEO}) that matches the topological relationship (i.e., SUB_{SP_OPER}) with SUB_{SP_GEO}. For example, with relation to the sensor web, a PUB_{GEO} could be a sensor's location or the geometry of a feature that the sensor observed (e.g., the coverage of a river or a road intersection).

The relationship between PUB_{GEO}, SUB_{SP_OPER}, and SUB_{SP_GEO} follows the *subject–verb–object* structure, in which PUB_{GEO}, SUB_{SP_OPER}, and SUB_{SP_GEO} are the subject, verb, and object, respectively. For example, if "*point_1*" is PUB_{GEO}, "*WITHIN*" is SUB_{SP_OPER}, and "*polygon_1*" is SUB_{SP_GEO}, the spatial predicate (i.e., SUB_{SP}) evaluates whether the relationship "*point_1 WITHIN polygon_1*" is true or not. More specifically, in the example "send me a notification if any water level sensor within 1 km radius of my house reports a reading higher than 1 meter", the SUB_{SP_GEO} is the "1 km radius of my house", the SUB_{SP_OPER} is the "within", and the PUB_{GEO} is the location of any water level sensors.

This paper is organized as follows: Section 2 reviews the literature related to this research and defines the topological operators. Section 3 and Section 4 introduce the details of the proposed AHS model and distributed AHS model, respectively. Section 5 explains the evaluation results. Lastly, Section 6 presents the conclusions and future work.

2. Related Work

The AHS model is proposed to efficiently process topological operators in a geospatial sensor web publish/subscribe system. There have been different types of systems applying the publish/subscribe model to manage and process continuous data streams, such as publish/subscribe systems [19], simple event processing systems [20], data stream management systems (DSMS) [13,21,22], and complex event processing systems (CEP) [23]. In addition, there are some new open-source

platforms that allow users to construct a publish/subscribe system, such as the Apache Spark and RethinkDB.

However, geospatial publish/subscribe systems are rarely discussed as compared to general-purpose systems. Although there has been some work discussing the topic of supporting geospatial operators in a publish/subscribe system, most of them applied spatial database join operations to prove the concept, such as [12,15,24]. None of these studies discussed improving the efficiency of geospatial algorithms for publish/subscribe systems. Therefore, we are putting forward the argument that geospatial algorithms can be improved based on the nature of the continuous query processing model.

More specifically, one of the important continuous query optimization approaches is sharing operators across similar query plans [14,25–27], which allows every necessary operator to be executed only once and avoids processing redundancy. However, existing topological operators do not take into consideration this sharing execution concept. Even with using the minimum bounding boxes of geometries to limit the number of topological operator executions [17], handling a vast number of geometries would still require a large number of topological operators and become a performance bottleneck. Hence, the objective of this research is to discover the shareable processes in the topological operators and propose a new approach to share them across queries.

As mentioned earlier, this research focuses on the eight topological relationships defined in the OGC Simple Feature Access Specification [16]: *EQUALS*, *DISJOINT*, *INTERSECTS*, *TOUCHES*, *OVERLAPS*, *CROSSES*, *WITHIN*, and *CONTAINS*. This specification has been widely adopted in many spatial databases, such as PostGIS, Oracle, and Microsoft SQL Server. The typical approach to determine topological relationships is the Dimensionally Extend 9 Intersection Model (DE-9IM) [28].

DE-9IM has three main steps. Firstly, DE-9IM generates the interior, boundary, and exterior regions of two geometries. The second step of DE-9IM is to intersect these interior, boundary, and exterior regions of two geometries and construct a three-by-three intersection matrix (Equation (1)). Finally, if the intersection matrix matches the predefined matrices (i.e., Table 1), the topological relationships between the two geometries can be determined accordingly. Table 1 shows the topological relationships, the definition of relationships, and the corresponding intersection matrices, in which the wildcard symbol (∗) means "any value would work" [16].

$$DE-9IM(\mathrm{a},\mathrm{b}) = \begin{bmatrix} dim\big(I(\mathrm{a}) \cap I(\mathrm{b})\big) & dim\big(I(\mathrm{a}) \cap B(\mathrm{b})\big) & dim\big(I(\mathrm{a}) \cap E(\mathrm{b})\big) \\ dim\big(B(\mathrm{a}) \cap I(\mathrm{b})\big) & dim\big(B(\mathrm{a}) \cap B(\mathrm{b})\big) & dim\big(B(\mathrm{a}) \cap E(\mathrm{b})\big) \\ dim\big(E(\mathrm{a}) \cap I(\mathrm{b})\big) & dim\big(E(\mathrm{a}) \cap B(\mathrm{b})\big) & dim\big(E(\mathrm{a}) \cap E(\mathrm{b})\big) \end{bmatrix}, \quad (1)$$

where the dim function returns the maximum dimension (i.e., 0 for points, 1 for lines, and 2 for polygons) of the intersection (∩) of interior (I), boundary (B), and exterior (E) of geometries a and b. The geometry a and b are respectively called the primary and secondary geometry. If an intersection is an empty set (∅), the dim function returns a value of –1. If an intersection is not an empty set, the dim function returns 0, 1, or 2. One way to simplify the matrix is to store only True (if the dim function returns 0, 1, or 2) and False (if the dim function returns–1) in the matrix, which is also represented in Table 1.

Note that for the *CROSSES* relationship, the OGC definition shown in Table 1 may not be the most commonly used definition. Instead, the *CROSSES* relationship is usually defined as " $(dim(I(a) \cap I(b)) < max(dim(I(a)), dim(I(b)))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$ ". Hence, the DE-9IM matrix is defined as $dim(I(a) \cap I(b)) = 0$ for a line/line relationship. In this research, the proposed solution follows the common definition.

However, DE-9IM is a time-consuming process and could create a performance bottleneck when processing a large number of geometries [17]. In order to address this issue, a common solution is to reduce the number of unnecessary DE-9IM processes. For example, a typical approach consists of two stages: filter and refinement [17]. The filter stage finds candidate geometries with approximated rectangles of geometries (e.g., minimum bounding rectangles (MBRs)); then the refinement stage performs the actual DE-9IM process on the candidates found in the filter stage.

While this approach has been widely applied in many DBMS systems (e.g., PostGIS), we argue that it can be further improved for a publish/subscribe system to handle a larger number of geometries.

Although existing research discussed the support of spatial operators in a publish/subscribe system, most of them only applied the same filter-and-refinement spatial join approach to prove the concept. We believe that time-consuming geospatial algorithms can be improved by sharing executions. As a result, this research proposes the AHS model as a new approach for improving the efficiency of topological operators in a sensor web publish/subscribe system.

3. AHS Model

We propose the AHS model to improve the query efficiency and scalability of topological operators in a sensor web publish/subscribe system. The AHS model follows the definition of the eight topological relationships in the OGC Simple Feature Access specification. As such, it follows the same conceptual framework used in traditional approaches such as DE-9IM.

The difference between the AHS model and traditional approaches is that the current AHS model does not take multi-point, multi-line, and multi-polygon into consideration. Extending the algorithm to support these types of geometries would not change the main idea for the AHS model, but is one of the future directions that will be pursued.

Since the targeted objectives of traditional DBMS and publish/subscribe systems are essentially different, algorithms optimized for DBMS may not be suitable for publish/subscribe systems. For example, since the queries in DBMS are atomic and independent, the algorithms are optimized for each individual query. However, since the queries are continuous and pre-defined in publish/subscribe systems, their algorithms should consider the aggregation of multiple queries/subscriptions.

With the nature of continuous queries, we argue that it is acceptable to spend more effort (e.g., create indices) on the start-up preparation stage to execute continuous queries more efficiently. Although this approach may cause some delay in the beginning, it can generate a larger throughput considering the long-running nature of continuous queries. There are two key ideas in AHS model that are based on these concepts. Firstly, the AHS model pre-generates the necessary indices from the geometries of subscriptions and then reuses the indices when needed in the continuous queries. Secondly, by indexing the geometries of subscriptions with the same indexing structure, we can aggregate the indices of all subscriptions to save storage space and intersect PUB_{GEO} with all SUB_{SP_GEO} in a single process.

The AHS .odel consists of three major stages: (1) The preparation stage generates necessary information from the geometries of subscriptions; (2) the intersection stage intersects the geometries of subscriptions with the geometry of publication; and (3) the determination stage determines geospatial relationships. The details of these three stages are introduced in the following sections.

Table 1. The topological relationships and their corresponding intersection matrices.

Relationship	OGC Definition (P: Point, L: Line, A: Polygon)	DE-9IM Intersection Matrix (T: True, F: False)
$a\ EQUALS\ b$	$a \subseteq b \wedge b \subseteq a$	$\begin{bmatrix} T & * & F \\ * & * & F \\ F & F & * \end{bmatrix}$
$a\ DISJOINTS\ b$	$a \cap b = \emptyset$	$\begin{bmatrix} F & F & * \\ F & F & * \\ * & * & * \end{bmatrix}$
$a\ INTERSECTS\ b$	$a \cap b \neq \emptyset$	$\begin{bmatrix} T & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} * & T & * \\ * & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} * & * & * \\ T & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} * & * & * \\ * & T & * \\ * & * & * \end{bmatrix}$
$a\ TOUCHES\ b$	$(I(a) \cap I(b) = \emptyset) \wedge (a \cap b \neq \emptyset)$. Applies to P/L, P/P, L/L, L/A, and A/A situations.	$\begin{bmatrix} F & T & * \\ * & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} F & * & * \\ * & T & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} F & * & * \\ T & * & * \\ * & * & * \end{bmatrix}$
$a\ OVERLAPS\ b$	$(dim(I(a)) = dim(I(b)) = dim(I(a) \cap I(b))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$. Applies to P/P, L/L, and A/A situations.	$\begin{bmatrix} T & * & T \\ * & * & * \\ T & * & * \end{bmatrix}$ or $\begin{bmatrix} 1 & * & T \\ * & * & * \\ T & * & * \end{bmatrix}$ (for line/line relationship)
$a\ CROSSES\ b$	$(I(a) \cap I(b) \neq \emptyset) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$. Applies to P/L, P/A, L/Land L/A situations.	$\begin{bmatrix} T & * & T \\ * & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} 0 & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$ (for line/line relationship)
$a\ WITHIN\ b$	$(a \cap b = a) \wedge (I(a) \cap E(b) = \emptyset)$	$\begin{bmatrix} T & * & F \\ * & * & F \\ * & * & * \end{bmatrix}$
$a\ CONTAINS\ b$	$(a \cap b = b) \wedge (I(b) \cap E(a) = \emptyset)$	$\begin{bmatrix} T & * & * \\ * & * & * \\ F & F & * \end{bmatrix}$

3.1. Preparation Stage: Generate Necessary Information from the Geometries of Subscriptions

Similar to the way DE-9IM works, the AHS model also determines topological relationships by intersecting the interior, boundary, and exterior regions of two geometries. However, instead of the typical two-step (i.e., filter and refinement) approach, the AHS model performs candidate finding and intersections at the same time. This is doable in a sensor web publish/subscribe system as queries are predefined and PUB_{GEO} is usually small (e.g., a sensor's location, a road intersection, or a football field). The indices of the geometries in subscriptions (i.e., SUB_{SP_GEO}) are first created and reused until SUB_{SP_GEO} is changed. By doing so, we can avoid generating redundant indices, which can consequently speed up the query processing.

However, not every indexing structure is suitable for the AHS model. Firstly, the indexing structure should be space-driven, so that the indices of SUB_{SP_GEO} can be aggregated. Secondly, space-driven indexing structure should have multiple levels to reduce the storage. Therefore, the AHS model needs a space driven and multi-level hierarchical indexing structure, similar to that in the idea proposed by [29]. The hierarchical indexing structure allows multiple nodes on a lower level to be aggregated as a node on a higher level. In this research, a quadtree tile system [30] is used as the hierarchical structure. By defining the lowest level of the quadtree as the finest granularity, any geometry can be indexed into (or approximated as) a list of quadtree nodes (i.e., quadkeys).

Examples can be seen in Figure 1, where the maximum quadtree level is 4 and interiors, boundaries, and exteriors are represented in dark gray, light gray, and white, respectively. Since lower-level indices can be aggregated into higher-level indices, the number of indices can be reduced if a geometry covers multiple quadkeys. The upper two polygons in Figure 1 are used as examples. The upper-left polygon is about four times larger than the upper-right polygon. By using a hierarchical structure, we can aggregate the upper-left polygon's interior indices from 72 forth-level indices to six indices (i.e., five third-level and one second-level indices), which is the same number of interior indices as in the upper-right polygon.

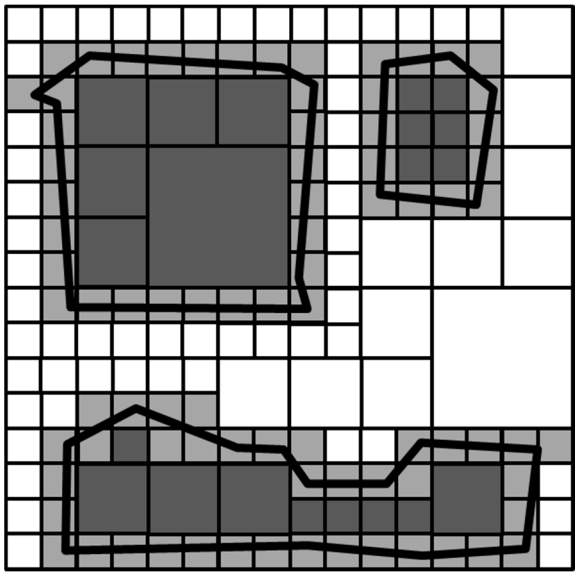


Figure 1. Examples of AHS model indices with the fourth level as the lowest quadtree level (interior: **dark gray**, boundary: **light gray**, and exterior: **white**).

In addition, the determination of a topological relationship does not require all three of the interior, boundary, and exterior of SUB_{SP_GEO}. Instead, the AHS model only needs to generate the necessary information according to the spatial predicate SUB_{SP}. As such, we can further reduce the number of indices and speed up the query processing. Based on the definition of topological relationships in OGC Simple Feature Access Specification (Table 1) and the possible topological relationships between different geometry types, we can analyze and propose the necessary information for determining each geospatial relationship.

However, during a preliminary test of the first version of the AHS model, it was found that the indexing and query performance using exterior indices were poorer than when only using interior and boundary indices. This is because the number of exterior indices is usually larger than the number of interior and boundary indices. As a result, we redesigned the AHS model to avoid exterior indices. Our final analysis is listed below:

- *EQUALS*: Since we can compare the interior and boundary of two geometries to determine the *EQUALS* relationship, only the interior and boundary are needed.
- *DISJOINT*: Since the *DISJOINT* relationship can be seen as "no intersection between the interiors and boundaries", only the interior and boundary are needed to determine *DISJOINT* relationship.
- *INTERSECTS*: Since the *INTERSECTS* relationship means that the two geometries have at least one interior or boundary point in common, only the interior and boundary are needed for this relationship.
- *TOUCHES*: The *TOUCHES* relationship means two geometries are *INTERSECTS* but their interiors do not intersect with each other. Therefore, similar to the *INTERSECTS* relationship, only the interior and boundary are required for the *TOUCHES* relationship.
- *OVERLAPS*: The *OVERLAPS* relationship means that the interior of both geometries intersects the interior and exterior of the other. If both geometries are line geometries, the intersection of interiors needs to be a line for *OVERLAPS* relationship. In order to avoid the processing of exterior indices, we follow the idea that "*if the intersection of interiors and boundaries are not equal to neither geometries, each geometry interests with the exterior of the other*", that is $(a \cap E(b) \neq \emptyset) \wedge (E(a) \cap b \neq \emptyset)\ iif\ (a \cap b \neq a) \wedge (a \cap b \neq b)$. Therefore, only the interior and exterior are required for this relationship. In addition, since multi-point is not in the scope, a single point does not *OVERLAP* with another point.
- *CROSSES:* The *CROSSES* relationship means that the interior of the primary geometry a intersects the interior and exterior of secondary geometry b. Similar to the *OVERLAPS* relationship, we replace the requirement of the exterior with the intersections of interiors and boundaries. In addition, for line geometries, they are *CROSSES* if, and only if, their interiors intersect at a point. Since we only consider single point geometries, no geometry can cross a single point based on the OGC definition. As a result, the determination of the *CROSSES* relationship only needs the interior for line geometries and requires both interior and boundary for polygon geometries.
- *WITHIN*: "a *WITHIN* b" means that the interior and boundary of a fully locate in the interior and boundary of b. That means that only the interior and boundary are required to determine the *WITHIN* relationship.
- *CONTAINS*: The *CONTAINS* relationship is the inverse of the *WITHIN* relationship, which means "a *CONTAINS* b" and "b *WITHIN* a" are equal. Therefore, the interior and boundary are also necessary information for *CONTAINS*.

After generating the necessary information from SUB_{SP}, the AHS model aggregates the necessary indices from all subscriptions into a single data structure. In the data structure, each quadkey (which is shared by at least one subscription) maintains a list of subscription identifiers (SUB_{ID}) and the corresponding region type ($TYPE_{SUB}$) (i.e., interior or boundary). In this case, the AHS model can directly match quadkeys of new data with quadkeys in the data structure to intersect new data with all subscriptions in a single process. The worst-case scenario is that no quadkey is shared by any other subscriptions (i.e., each quadkey only attaches to one SUB_{ID}), which means that the aggregation benefits neither the storage nor the query processing. On the other hand, as long as there is more than one subscription sharing the same quadkey, the aggregation can reduce both the storage size and the query latency. For the remainder of this paper, this aggregated quadtree structure is referred to as AHS_{SUB} for clarity purposes.

A more critical contribution is that by aggregating quadkeys from all SUB_{SP_GEO} into AHS_{SUB}, the AHS model can simply match the PUB_{GEO} with the quadkeys in the AHS_{SUB} and link to a set of SUB_{ID}. This means that the AHS model decouples quadkeys and SUB_{ID} and allows itself to be more scalable in terms of the number of subscriptions.

3.2. Intersection Stage: Intersect with the Geometry of Publication

There are three steps for intersecting the geometry of publication (i.e., PUB_{GEO}) with AHS_{SUB}: (1) index; (2) match; and (3) create matrices. This workflow is shown in Figure 2. In order to efficiently intersect the PUB_{GEO} with the AHS_{SUB}, the AHS model first indexes the interior and boundary of PUB_{GEO} with the same hierarchical structure (i.e., quadtree tile system). The outcome of this indexing is named as AHS_{PUB} for clarity purposes.

As covered in the previous section, the AHS model modifies the determination algorithm in order to avoid processing exterior indices. Therefore, only the interior and boundary of the publication geometry need to be generated at this stage.

This matching step benefits from using the same indexing structure. Since both AHS_{PUB} and AHS_{SUB} are indexed by the same quadtree tile system, we can match the prefixes of quadkeys to find the intersection. More specifically, every quadkey q has a corresponding geospatial bounding box $bbox$. Given two quadkeys q_A and q_B, $bbox_A \subseteq bbox_B$ if, and only if, q_A starts with q_B. The bounding box of quadkey '0' contains all the bounding boxes of quadkeys whose first digit is '0'. If a quadkey in AHS_{PUB} intersects with a quadkey in AHS_{SUB}, this intersection is referred to as a *match*.

Each match contains the following five attributes: (1) the intersected quadkey from AHS_{PUB}; (2) the intersected quadkey from AHS_{SUB}; (3) the subscription identifier (SUB_{ID}); (4) the region type of AHS_{PUB} ($TYPE_{PUB}$; i.e., interior or boundary); and (5) the region type of AHS_{SUB} ($TYPE_{SUB}$; i.e., interior or boundary). After finding all the matches, they are grouped by SUB_{ID} and a matrix created for each group. This matrix is referred to as the area matrix because it records the size of the intersected area. In a similarly way to the three-by-three intersection matrices of the DE-9IM, the area matrices are for determining the topological relationship between geometries. The area matrix is a two-by-two matrix, and its form is shown in Equation (2):

$$\text{Area matrix}(matches) = \begin{bmatrix} Area(II(matches)) & Area(IB(matches)) \\ Area(BI(matches)) & Area(BB(matches)) \end{bmatrix}, \quad (2)$$

where the $Area$ function returns the sum of the number of area unit (here we define that every lowest level quadkey has one area unit and the quadkey on level n has $4^{(\text{lowest level-}n)}$ area units), $II(matches)$ returns intersected quadkeys whose $TYPE_{PUB}$ and $TYPE_{SUB}$ are both interior, $IB(matches)$ returns intersected quadkeys whose $TYPE_{PUB}$ is interior and $TYPE_{SUB}$ is boundary, $BI(matches)$ returns intersected quadkeys whose $TYPE_{PUB}$ is boundary and $TYPE_{SUB}$ is interior, and $BB(matches)$ returns intersected quadkeys whose $TYPE_{PUB}$ and $TYPE_{SUB}$ are both boundary.

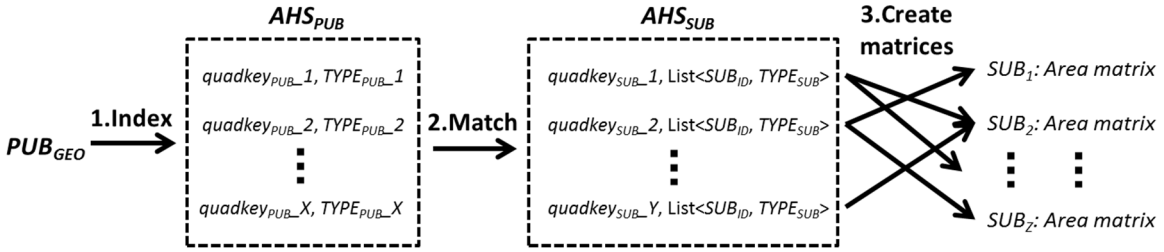


Figure 2. The workflow for intersecting the geometry of publication.

3.3. Determination Stage: Determining the Topological Relationship

With the area matrices generated in the previous stage, the topological relationships between PUB_{GEO} and SUB_{SP_GEO} can be determined. As with the DE-9IM, each relationship has a specific matrix pattern. Table 2 lists the topological relationships and their corresponding area matrices ($AMatrix$), in which the $Area$ function returns the number for the area unit whilst I and B functions return the interior and boundary of SUB_{SP_GEO} or PUB_{GEO}, respectively. In addition, the $AMatrix_{II}$, $AMatrix_{IB}$, $AMatrix_{BI}$, and $AMatrix_{BB}$ represent the cell (0, 0), (0, 1), (1, 0), and (1, 1) in an area matrix, respectively. Finally, $Sum(AMatrix)$ represents $AMatrix_{II} + AMatrix_{IB} + AMatrix_{BI} + AMatrix_{BB}$.

Note that most of the information used during the determination process can be calculated in advance. For instance, $Area(SUB_{SP\_GEO})$, $Area(I(SUB_{SP\_GEO}))$, $Area(B(SUB_{SP\_GEO}))$, $Area(PUB_{GEO})$, $Area(I(PUB_{GEO}))$, and $Area(B(PUB_{GEO}))$ can be generated at the time when subscriptions and publications are entered into the system.

An explanation of the concept of each determination:

- *EQUALS*: If PUB_{GEO} *EQUALS* SUB_{SP_GEO}, their interiors and boundaries should be the same, which means that their interiors completely intersect with each other and so do their boundaries. Consequently, $AMatrix_{II}$ equals the area of both interiors, and $AMatrix_{BB}$ equals the area of both boundaries.
- *DISJOINTS*: If PUB_{GEO} *DISJOINTS* SUB_{SP_GEO}, both the interior and boundary of PUB_{GEO} are completely located on the exterior of SUB_{SP_GEO}. There is no intersection between the interiors and boundaries of PUB_{GEO} and SUB_{SP_GEO}. Therefore, $AMatrix_{II}$, $AMatrix_{IB}$, $AMatrix_{BI}$, and $AMatrix_{BB}$ are all zero.
- *INTERSECTS*: PUB_{GEO} *INTERSECTS* SUB_{SP_GEO} if any interiors or boundaries intersect, which means that one of the cells in *AMatrix* is not zero.
- *TOUCHES*: If PUB_{GEO} *INTERSECTS* SUB_{SP_GEO} and their interiors do not intersect (i.e., $AMatrix_{II}$ equals to zero), PUB_{GEO} *TOUCHES* SUB_{SP_GEO}.
- *OVERLAPS*: As mentioned in Section 4.1, the processing of exterior indices is replaced with the intersection of interiors and boundaries as "*if the intersection of interiors and boundaries are not equal to both geometries, each geometry intersects with the exterior of the other*". Therefore, for non-line/line relationships, the determination algorithm becomes "if $AMatrix_{II}$ is not zero and *Sum*(*AMatrix*) equals to neither *Area*(PUB_{GEO}) nor *Area*(SUB_{SP_GEO}), PUB_{GEO} *OVERLAPS* SUB_{SP_GEO}".
- In addition, as the OGC specification defines, the *OVERLAPS* relationship requires the dimension of the intersection region to be equal to the dimensions of both geometries. The intersection of a "line *OVERLAPS* line" relationship should be a line instead of a point. Therefore, for a line/line relationship, the AHS model also checks if $AMatrix_{II}$ is larger than *one* area unit in order to make sure that the PUB_{GEO} does not intersect with SUB_{SP_GEO} at a point.
- *CROSSES*: For non-line/line relationships, the determination algorithm for PUB_{GEO} *CROSSES* SUB_{SP_GEO} relationship is the same as that for the *OVERLAPS* relationship. For line/line relationships, as the AHS model follows the common definition mentioned in Section 2, PUB_{GEO} *CROSSES* SUB_{SP_GEO} if the interiors intersect at a point (i.e., *one* area unit).
- *WITHIN*: The relationship of PUB_{GEO} *WITHIN* SUB_{SP_GEO} means that $AMatrix_{II}$, $AMatrix_{IB}$, $AMatrix_{BI}$, and $AMatrix_{BB}$ contain the entire area of PUB_{GEO} (i.e., the interior and boundary of PUB_{GEO}).
- *CONTAINS*: PUB_{GEO} *CONTAINS* SUB_{SP_GEO} if $AMatrix_{II}$, $AMatrix_{IB}$, $AMatrix_{BI}$, and $AMatrix_{BB}$ contain the entire area of SUB_{SP_GEO} (i.e., the interior and boundary of SUB_{SP_GEO}).

Table 2. The topological relationships and their corresponding area matrices.

Topological Relationship	Area Matrix	
$PUB_{GEO}\ EQUALS\ SUB_{SP\_GEO}$	$\begin{bmatrix} Area\left(I\left(SUB_{SP\_GEO}\right)\right) & 0 \\ 0 & Area\left(B\left(SUB_{SP\_GEO}\right)\right) \end{bmatrix}$ and $\begin{bmatrix} Area\left(I\left(PUB_{GEO}\right)\right) & 0 \\ 0 & Area\left(B\left(PUB_{GEO}\right)\right) \end{bmatrix}$	
$PUB_{GEO}\ DISJOINTS\ SUB_{SP\_GEO}$	$AMatrix_{II} = AMatrix_{IB} = AMatrix_{BI} = AMatrix_{BB} = 0$	
$PUB_{GEO}\ INTERSECTS\ SUB_{SP\_GEO}$	$\begin{bmatrix} \neq 0 & * \\ * & * \end{bmatrix}$ or $\begin{bmatrix} * & \neq 0 \\ * & * \end{bmatrix}$ or $\begin{bmatrix} * & * \\ \neq 0 & * \end{bmatrix}$ or $\begin{bmatrix} * & * \\ * & \neq 0 \end{bmatrix}$	
$PUB_{GEO}\ TOUCHES\ SUB_{SP\_GEO}$	$\begin{bmatrix} = 0 & \neq 0 \\ * & * \end{bmatrix}$ or $\begin{bmatrix} = 0 & * \\ * & \neq 0 \end{bmatrix}$ or $\begin{bmatrix} = 0 & * \\ \neq 0 & * \end{bmatrix}$	
$PUB_{GEO}\ OVERLAPS\ SUB_{SP\_GEO}$	$AMatrix_{II} \neq 0$ and $Sum(AMatrix) \neq Area(PUB_{GEO})$ and $Sum(AMatrix) \neq Area(SUB_{SP\_GEO})$	For line/line relationship, $AMatrix_{II} > 1$ and $Sum(AMatrix) \neq Area(PUB_{GEO})$ and $Sum(AMatrix) \neq Area(SUB_{SP\_GEO})$
$PUB_{GEO}\ CROSSES\ SUB_{SP\_GEO}$	$AMatrix_{II} \neq 0$ and $Sum(AMatrix) \neq Area(PUB_{GEO})$ and $Sum(AMatrix) \neq Area(SUB_{SP\_GEO})$	For line/line relationship, $AMatrix_{II} = 1$
$PUB_{GEO}\ WITHIN\ SUB_{SP\_GEO}$	$Sum(AMatrix) = Area(PUB_{GEO})$	
$PUB_{GEO}\ CONTAINS\ SUB_{SP\_GEO}$	$Sum(AMatrix) = Area(SUB_{SP\_GEO})$	

4. Distributed AHS Model

As discussed earlier, distributed computing and cloud computing techniques are applied to the AHS model to provide more storage and computation resources. For example, we can split the AHS_{SUB} into pieces and store them in different machines to effectively address the potential storage issue. We can also use distributing computing techniques to improve query processing performance, especially for the indexing and matching stages. To be more specific, we follow the MapReduce framework [18] and propose a distributed AHS model processing flow.

Figure 3 shows the high-level architecture and workflow of the distributed AHS model. There are four stages in the distributed AHS model, namely (1) index; (2) match; (3) aggregate area matrices; and (4) determine relationships. As the processes in the (1) index; (2) match; and (4) determine relationships stages are the same as for the standalone AHS model, the (3) aggregate area matrices stage mainly groups and integrates area matrices based on SUB_{ID}.

As shown in Figure 3, the distributed AHS model has a master node and a set of worker nodes. The master node is responsible for receiving subscriptions and publications as well as forwarding subscriptions and publications to appropriate workers. Each worker is in charge of creating and matching indices according to the set of quadkeys assigned to it. That means that if a worker is in charge of quadkey q_A, all the indices that have q_A as a prefix are created and maintained by this worker. The master node has a lookup table storing the set of quadkeys that each worker is responsible for. For the remainder of this paper, this set of quadkeys will be referred to as *WorkerQ* and the lookup table as *LUT* for clarity purposes. Hence, a worker $Worker_i$ would have $WorkerQ_i$ as the set of quadkeys it is responsible for.

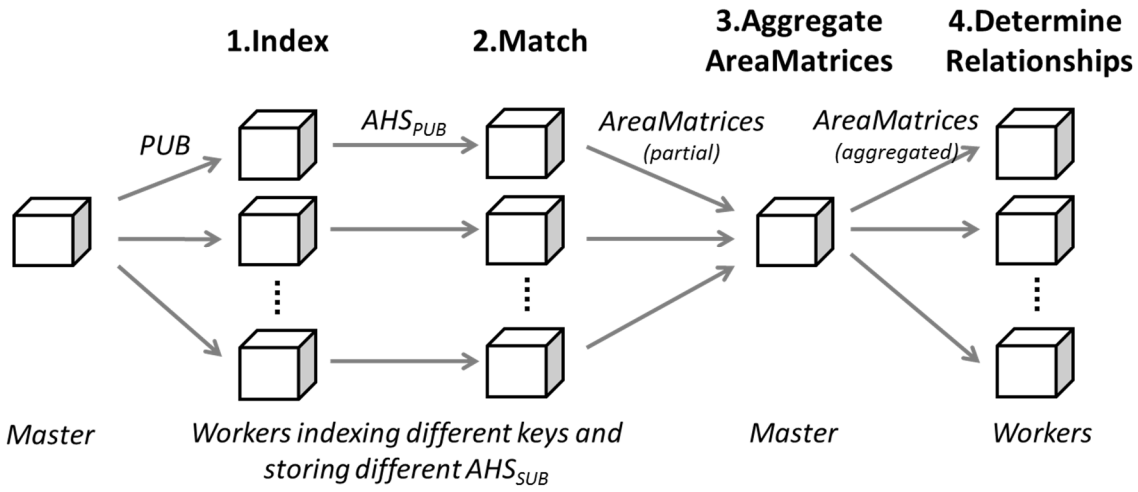


Figure 3. High-level architecture and workflow of the distributed AHS model.

Algorithm 1 shows the worker selection algorithm. When a master node receives a subscription *SUB* or a publication *PUB*, it uses the *LUT* and SUB_{SP_GEO} or PUB_{GEO} to determine the workers that are responsible for processing the *SUB* or *PUB*. Workers are returned if their *WorkerQ* overlaps with SUB_{SP_GEO} or PUB_{GEO}. In order to reduce the computation load on the master node, we used a coarse estimation on SUB_{SP_GEO} and PUB_{GEO}. That is, we first find the lowest level of quadkeys (*i.e.*, *LowestLevel*) in the *LUT* with the *GetLowestQuadkeyLevel* function (line 2 of Algorithm 1), and generate quadkeys of SUB_{SP_GEO} or PUB_{GEO} (i.e., *qs*) on the *LowestLevel* (line 3 of Algorithm 1). Then the containing relationship is determined by the prefix matching of quadkeys from *qs* and $WorkerQ_i$ (line 6 of Algorithm 1). Finally, if quadkeys from *qs* and $WorkerQ_i$ overlap with each other, the algorithm returns the $Worker_i$.

Algorithm 1. The worker selection algorithm.

```
Function SelectWorkers(LUT, Geo): Workers
Workers ← {}
LowestLevel ← GetLowestQuadkeyLevel(LUT)
qs ← GetQuadkeysByLevel(Geo, LowestLevel)
FOREACH WorkerQi ∈ LUT
    FOREACH q ∈ qs
        IF q is contained by WorkerQi OR q contains WorkerQi THEN
            IF Workers does not contain Workeri
                Workers ← Workers + Workeri
            BREAK
        END IF
    END FOREACH
END FOREACH
RETURN Workers
```

To match a publication, when the master node receives a publication *PUB*, it first uses Algorithm 1 to select one or more workers, and then forwards *PUB* to the selected workers. When a worker receives *PUB*, it creates indices with the PUB_{GEO} based on the quadkeys it is in charge of, matches AHS_{PUB} with the local AHS_{SUB}, and creates area matrices based on the local matches, before finally returning the created area matrices to the master. Since each worker only has a portion of AHS_{SUB} (based on the quadkeys it is in charge of), the created area matrices only represent a portion of the complete area matrices. Therefore, after the workers send the partial area matrices to the master, the master groups and aggregates them into complete area matrices based on the SUB_{ID}. Finally, the master node equally distributes the aggregated area matrices to workers to determine topological relationships in parallel.

Regarding the load balancing in the distributed AHS model, the number of quadkeys in AHS_{SUB} indicates the processing performance. This is because the number of quadkeys in AHS_{SUB} determines the required storage and CPU resources in a worker node. Therefore, a simple load-balancing approach is assigning a threshold θ on the number of quadkeys each worker handles. After every process of registering a subscription, if the number of quadkeys in a worker's AHS_{SUB} is larger than the threshold θ, the worker splits the original AHS_{SUB} into multiple AHS_{SUB}s based on the quadkeys it is in charge of. These split AHS_{SUB}s are then assigned to other existing or newly created workers. Finally, the master updates its lookup table accordingly.

To sum up, the AHS model is further improved by using distributed computing concepts. The proposed distributed AHS model is able to harness the storage and CPU resources from multiple machines to address potential storage issues and improve indexing and matching performance. The distributed AHS model assigns quadkeys to workers (i.e., *WorkerQ*) to distribute the processing load. This approach allows the distributed AHS model to retain the ability to match a publication with all subscriptions in a single process. This is because the quadkeys shared by multiple subscriptions remain aggregated in the same AHS_{SUB}.

5. Evaluation Results

We evaluate the proposed AHS model from four perspectives. Firstly, since the major objective of the AHS model is to efficiently process topological operators when handling a large number of geometries, we analyze the scalability of the AHS model in terms of the number of queries/subscriptions by comparing it with PostGIS, which is used to represent traditional topological operators.

The second evaluation is for measuring the indexing latency. Since the AHS model approximates geometries with a quadtree tile system, the indexing for large geometries may be time-consuming. Although we argue that the subscriptions and data in the context of sensor web would not have large geographical coverage, we evaluate the indexing latency of the AHS model by simulating geometries in various sizes for comprehensiveness.

The third evaluation analyzes the matching latency of the AHS model. More specifically, this evaluation measures the latency of matching AHS_{PUB} and AHS_{SUB}. Like the second evaluation, we examine geometries simulated in various sizes to provide a comprehensive evaluation.

Finally, the fourth evaluation is the end-to-end performance analysis for the distributed AHS model. We measure the overhead latencies and carry out each of the following steps: (1) index publication; (2) match AHS_{PUB} with AHS_{SUB}; and (3) determine the relationship. This evaluation examines all possible relationships between two geometries. The testing data for this evaluation are manually simulated city-level subscriptions and publications that we think are relatively realistic and that provide the expected AHS model performance in a real-world application.

5.1. AHS Model Scalability Evaluation

One of the most important functions of the AHS model is to perform topological operators in an aggregated manner. By aggregating the indices from all subscriptions in a single structure (i.e., AHS_{SUB}) and decoupling the indices and subscriptions, the AHS model can match new data with all subscriptions in a single process. In doing so, subscriptions that have quadkeys in common can benefit from this design.

In order to demonstrate this contribution, we evaluate the scalability of the AHS model in this section. The query performance is measured whilst registering different numbers of subscriptions into the AHS model. The point-in-polygon query is chosen as our testing case, as it is one of the most common queries. A subscription with the coverage of a city (i.e., a polygon) was simulated and assigned *WITHIN* as the topological operator in the spatial predicate. This was followed by the simulation of a publication with a point geometry located in the city. With the simulated subscription and publication, different numbers of subscriptions were registered into the AHS model (with different subscription identifier SUB_{ID}) and the query latency measured every 500 additional subscriptions by sending the publication to the AHS model. The same test was performed on an off-the-shelf PostGIS database for comparison.

PostGIS mainly use the GEOS library, which ports Java Topology Suite (JTS) to C++. JTS has been used in many products, including GDAL/OGR, QGIS, MapServer, and GeoTools. For the geometries stored in PostGIS, their minimum bounding rectangles (MBRs) are indexed. In the filter step, the MBRs are intersected with the input geometries, and the intersected geometries are found. Then in the refinement step, PostGIS uses each pair of database geometry and input geometry to construct an intersection matrix, which will be used to compare with the predefined DE-9IM patterns to decide the topological relationships [31].

Since this evaluation was mainly about the scalability in terms of the number of subscriptions, this evaluation was performed on a single machine with a standalone AHS model to avoid communication overhead and machine heterogeneity. This evaluation was performed on a desktop machine, which runs on an Intel® Core™ i5-650 @ 3.20GHz, 6GB RAM, and Western Digital WD10EARS-22Y5B1.

The query latencies on different numbers of subscriptions are shown in Figure 4. Based on these experimental results, we observed that the query latency increases with the number of subscriptions for both the PostGIS database and the AHS model. However, as the query latency of the AHS model increases 2.5 times slower than that of PostGIS, it shows that the AHS model is more scalable than the traditional solution in terms of the number of subscriptions.

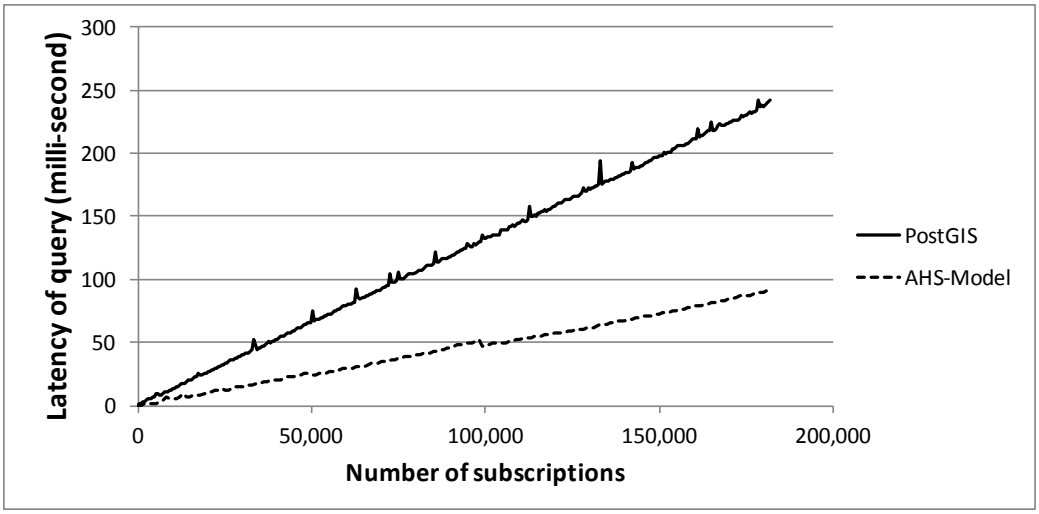


Figure 4. The AHS model query latency on different number of subscriptions.

5.2. Evaluation of AHS Model Indexing Performance

This section measures the latency of generating necessary quadkeys from the geometry of subscriptions. Since the time cost for indexing may differ based on the size of geometry, we randomly generated geometries in different sizes and measured the latencies for indexing them. In addition, as mentioned previously, the lowest level of quadtree tile is needed as the granularity on geometry approximation. The quadtree tile system used in this evaluation had 14 levels.

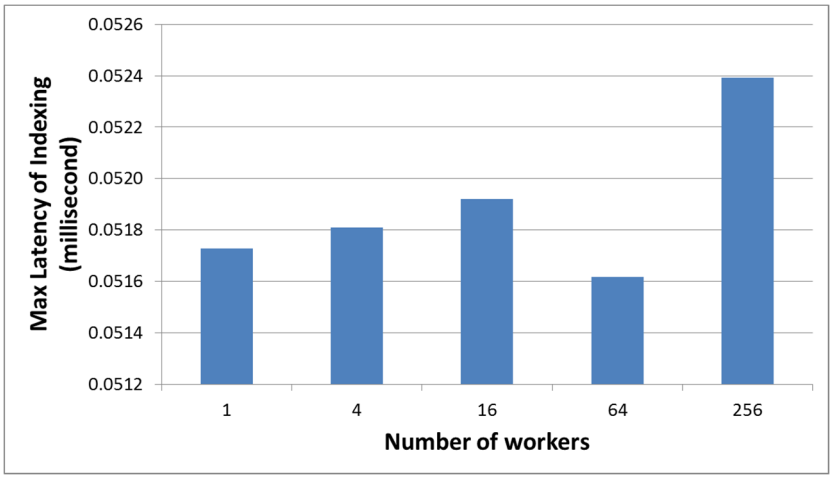
Furthermore, as the distributed AHS model can execute indexing tasks in parallel, we also measure the indexing latency when using different numbers of workers. However, in order to avoid machine heterogeneity, we used a single machine to simulate each worker in the distributed AHS model by executing workers' tasks independently. We simulated scenarios of distributed AHS models with one, four, 16, 64, and 256 workers. While the one-worker scenario is basically the standalone AHS model, each worker in the four-, 16-, 64-, and 256-workers scenarios handles a quadkey in the first, second, third, and fourth levels of the quadtree, respectively. For example, for the four-workers scenario, we simulated four workers handling quadkeys '0', '1', '2', and '3'. With distributed computing processes, the entire process is considered as finished at the time that the last worker finishes its task. Here we present the maximum (instead of average) indexing latency from workers in each scenario.

The indexing latency of point, line, and polygon geometries are shown in Figure 5. Since the sizes of point geometries are the same (i.e., one area unit = one quadkey), we take the average for each scenario. From Figure 5, we can observe that the indexing latency for point geometry is much smaller than that for other types of geometries, and the indexing latency for line geometry is smaller than that for polygon geometry. We believe this is because of the different number of quadkeys being indexed, which is related to the size and location of geometries.

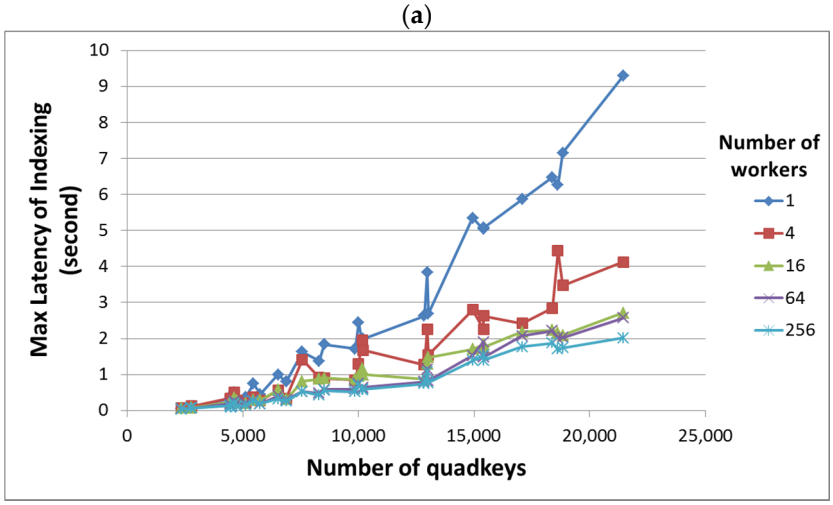
By comparing the indexing latencies based on different numbers of workers, we can observe that performance can be significantly improved by using more workers in the distributed AHS model. In addition, sometimes using four workers results in the same performance as using one worker does. The reason is that workers in the distributed AHS model handle different geospatial regions. If the randomly simulated test data are located only in one region, only one worker will conduct the whole process. Hence, with more workers handling different regions, we will have more opportunities of distributing the processing load. Our evaluation results show that the indexing process of using 256 workers can be 5 to 10 times faster than the standalone indexing process.

Finally, while this evaluation examines simulated geometries in various sizes for comprehensiveness, some of these geometries are too large in the context of the sensor web. Among these simulated geometries, the longest line geometry generated was 7112 kilometers long; the largest polygon geometry covered about 37% of Earth. In reality, a major city highway is usually about 100 kilometers; and a city usually covers less than 1% of Earth.

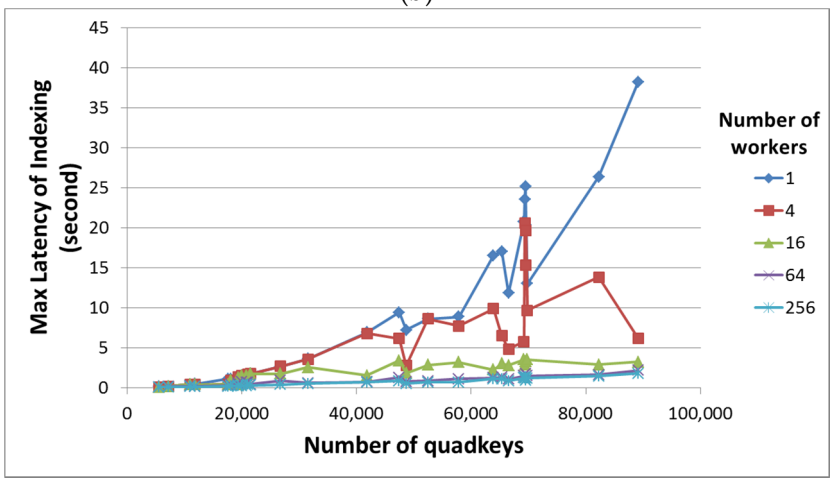
To sum up, while some of the simulated geometries are not realistic, the AHS model is able to finish the indexing step in a timely manner with the help of distributed processing. For indexing subscriptions, considering the long-running nature of the continuous query, we argue that the measured indexing overheads are acceptable. In addition, as real-world sensor web data usually have much smaller geospatial coverage than the simulated geometries, we believe the indexing for publications would be much faster than that for subscriptions.



(**a**)



(**b**)



(**c**)

Figure 5. The indexing latency for (**a**) point; (**b**) line; and (**c**) polygon geometry.

5.3. Evaluation of AHS Model Matching Performance

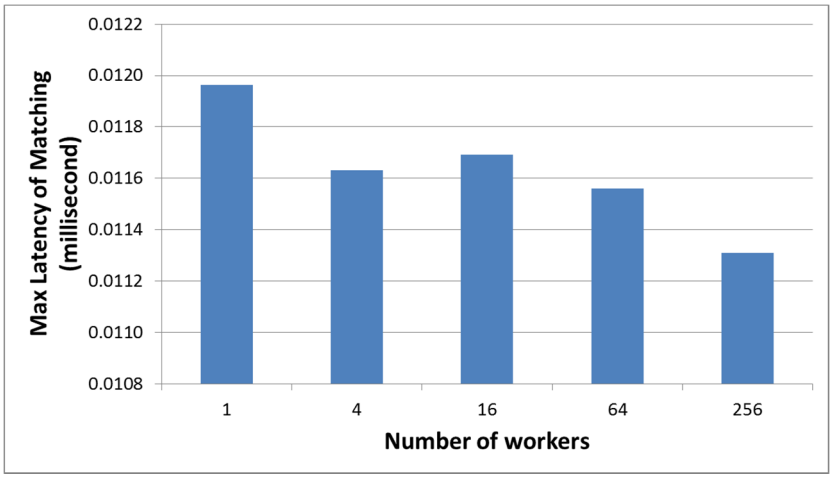
This section measures the latency of matching AHS_{PUB} with AHS_{SUB}. Since the time cost for matching may differ based on the number of quadkeys, we randomly generate geometries in various sizes to provide a comprehensive evaluation. In order to make sure that the quadkeys of these geometries will be processed, we first applied the same geometry to both publication and subscription, and then assigned *EQUALS* as the topological operator. In this evaluation, the quadtree tile system had 14 levels.

In addition, like the previous evaluation, we also measured the matching latency when applying different numbers of workers. In this evaluation, we used the same machine to simulate each worker handling different quadkeys in the distributed AHS model. Scenarios of distributed AHS models were simulated with one, four, 16, 64, and 256 workers. The maximum (instead of average) matching latency from workers in each scenario was also presented.

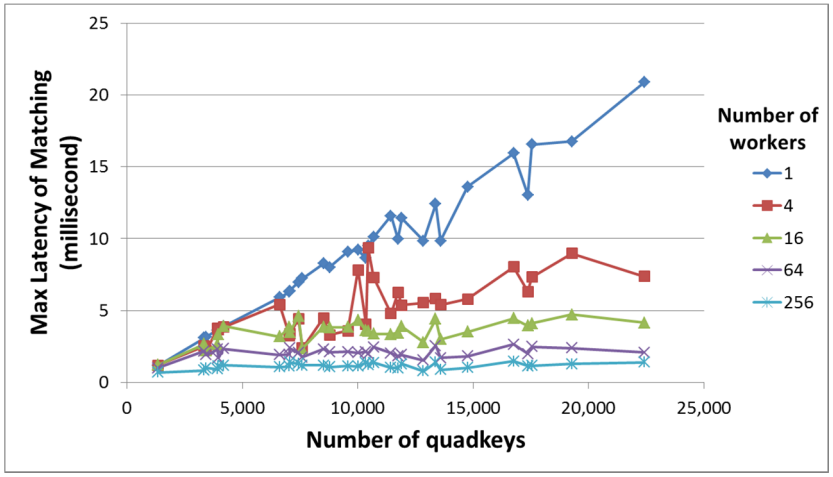
The matching latency of point, line, and polygon geometries are shown in Figure 6. Since the sizes of point geometries are the same (i.e., one area unit), we took the average latency for each scenario. From Figure 6, we can observe that the matching latency for point geometry is much smaller than that for other types of geometries, and the matching latency for line geometry is smaller than that for polygon geometry. As with the indexing performance, we believe this is because of the different number of quadkeys being processed.

By comparing the indexing latencies based on different numbers of workers, we can observe that the performance can be significantly improved by using more workers in the distributed AHS model. In addition, sometimes using four workers results in the same performance as using one worker does. The reason is that workers in the distributed AHS model handle different geospatial regions. If the randomly simulated test data are located only in one region, only one worker will conduct the entire process. Hence, with more workers handling different regions, we will have greater opportunities for distributing the processing load. Our evaluation results show that the matching process of using 256 workers can be 20 to 300 times faster than the standalone matching process.

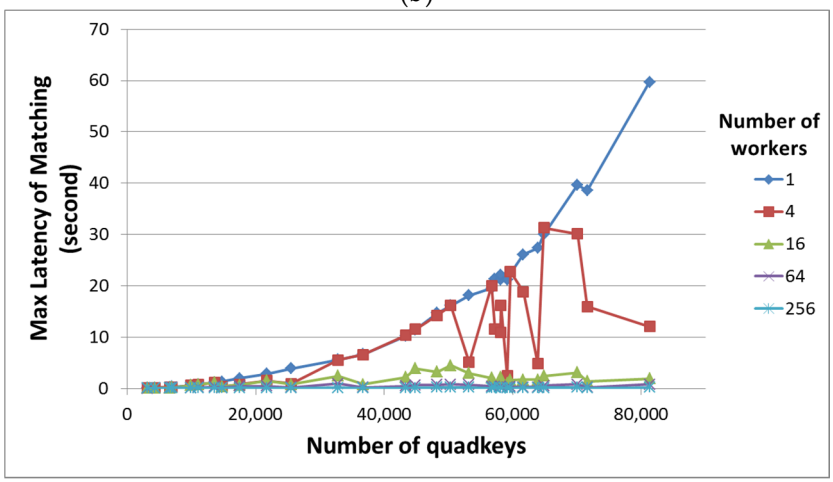
Finally, like the previous evaluation, this evaluation examines simulated geometries in various sizes for comprehensiveness. However, some of these geometries may be too large in the context of the sensor web. For example, among these simulated geometries, the longest line geometry we generated was 7778 kilometers long; the largest polygon geometry covered about 20% of Earth. However, while some of the simulated geometries are not realistic, the AHS model is able to match AHS_{PUB} and AHS_{SUB} in a timely manner with the help of distributed processing.



(**a**)



(**b**)



(**c**)

Figure 6. The matching latency for (**a**) point; (**b**) line; and (**c**) polygon geometry.

5.4. Evaluation of the End-to-End Query Performance of the AHS Model

In this section, we evaluate the end-to-end query performance of the AHS model. We measure the latency of indexing publication, matching AHS_{PUB} and AHS_{SUB} and determining relationships as well as overheads. In addition, this evaluation is performed on all possible relationships. We simulated one subscription/publication pair for each possible relationship and tried to create geometries with common sizes for city-level applications. For example, we created the point, line, and polygon of subscription based on the ideas of a point in a city (e.g., a city landmark), a road crossing a city (e.g., a major highway), and the city coverage, respectively. Publications that match subscriptions for each possible topological relationship were also manually created (e.g., a sensor located at a road intersection).

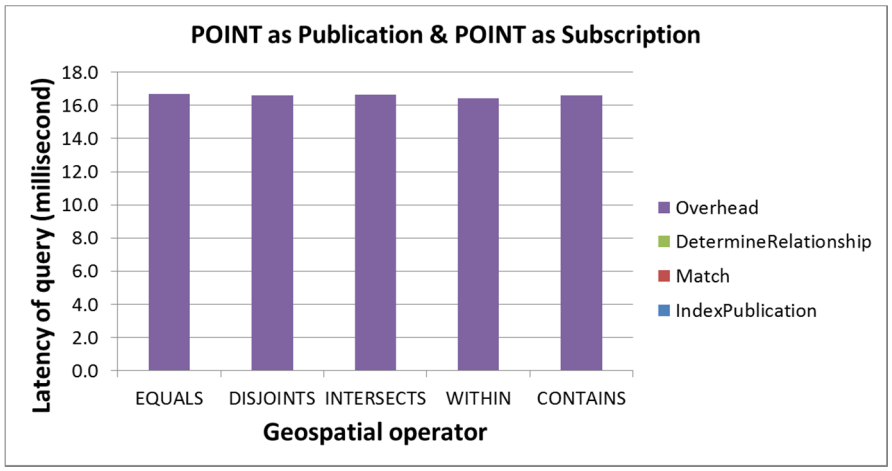
In order to test the overheads of distributed computing, this evaluation used two machines located on the same local network. The sets of quadkeys each worker is in charge of are manually configured so that workers handle similar amounts of work. Both machines are desktop machines. One of them runs on an Intel® Core™ i5-650 @ 3.20GHz, and 6GB RAM and the other on Intel® Core™ i7-3770 @ 3.40GHz, and 10GB RAM. Considering the machine's heterogeneity and the possibly unequal amount of work assigned, instead of presenting the maximum latencies, this evaluation calculates the average latencies to provide an expected AHS model performance in a real-world application. Each scenario is tested 10 times and the average for each scenario taken.

The end-to-end query performances of using point, line, and polygon geometry as subscriptions are shown in Figures 7–9, respectively. Based on these evaluation results, we found that it is difficult to simulate datasets that are equitable enough to be used in the comparison of different topological operators. We argue that the performance differences between topological

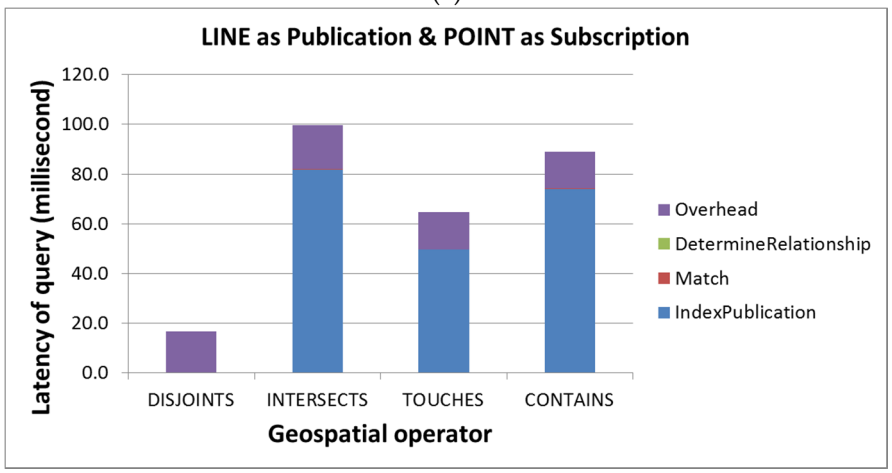
operators do not have much meaning as those differences may come from the machine's heterogeneity or the simulated dataset, such as the size of the geometry. However, this evaluation is still valuable as it measures the overheads of distributed computing, presents the latency of each step, and shows the potential AHS model performance in a real-world sensor web application.

Therefore, based on the experimental results, our first observation is that the indexing and overheads take up more than 99% of the end-to-end latency. While the overheads of applying distributed computing process are relatively stable (between 10 to 30 milliseconds), the indexing latency varies widely depending on the size of the publication geometry. Furthermore, in the context of a sensor web publish/subscribe system, since the geometries of the publication are usually small (e.g., sensor locations or observed features), we believe that the indexing latency will remain small as well.

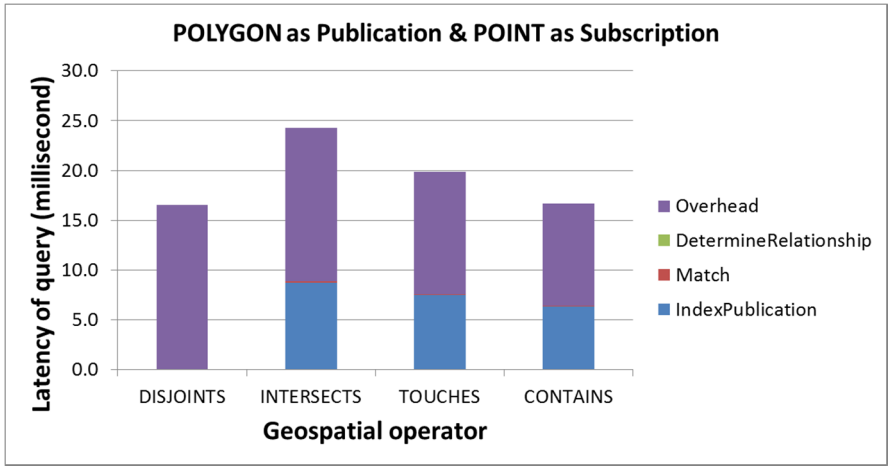
Our second observation is that the latencies for determining relationships are very small as each determination only handles a two-by-two matrix. Lastly, as this evaluation is based on a more realistic dataset, the measured performance could provide us with a possible AHS model performance in a real-world application. As we can see from the evaluation results, most of the tests can be finished in 100 milliseconds, while more than 70% of them can be completed in 50 milliseconds. Therefore, we believe that the AHS model can efficiently process any possible topological operators for sensor web data, which is critical for time-sensitive applications.



(**a**)

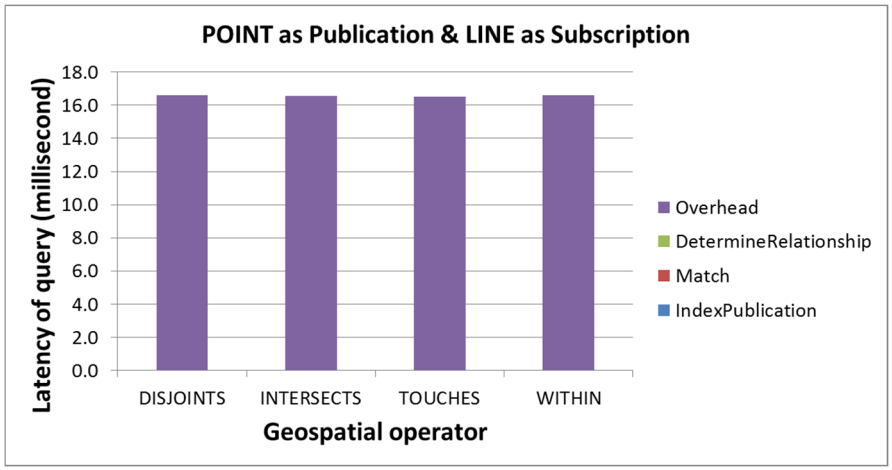


(**b**)

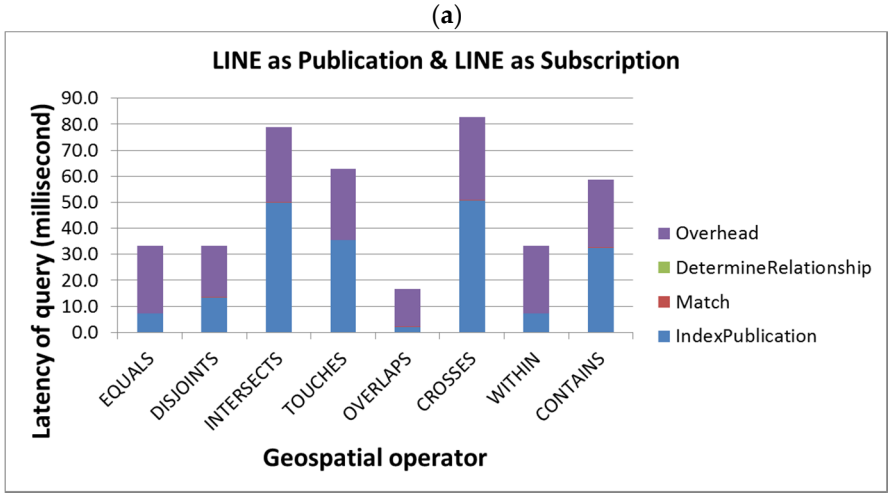


(**c**)

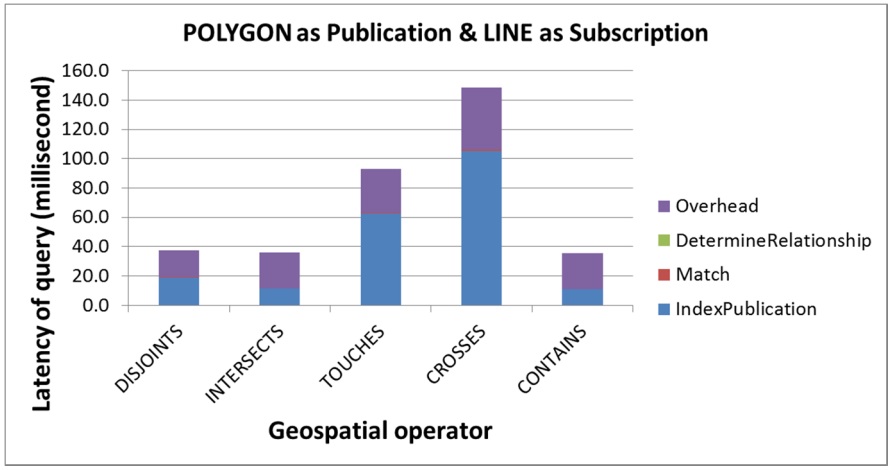
Figure 7. The end-to-end query performance for point as subscription and (**a**) point; (**b**) line; and (**c**) polygon geometry as publication.



(**a**)

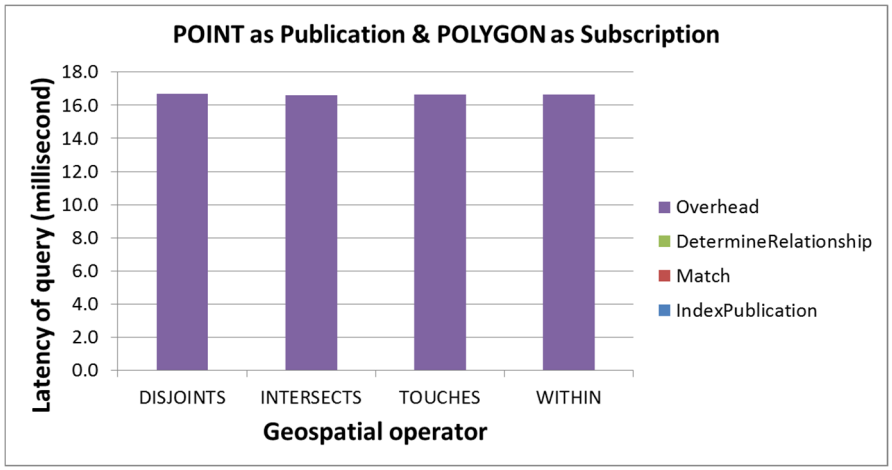


(**b**)

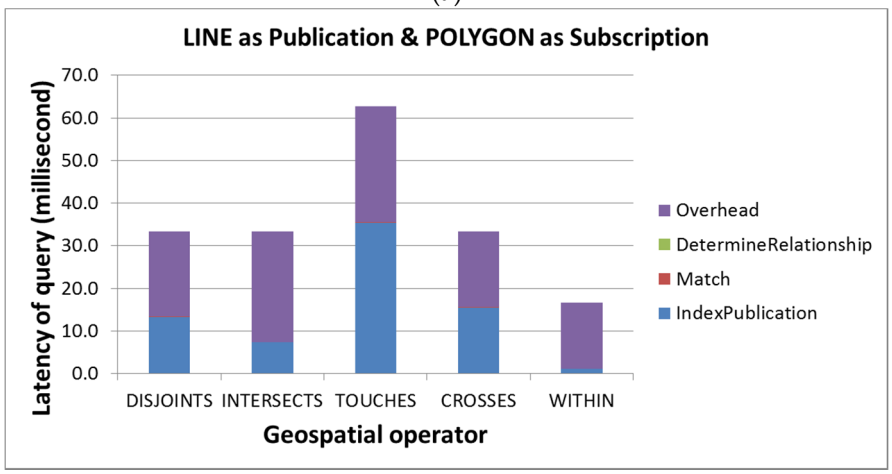


(**c**)

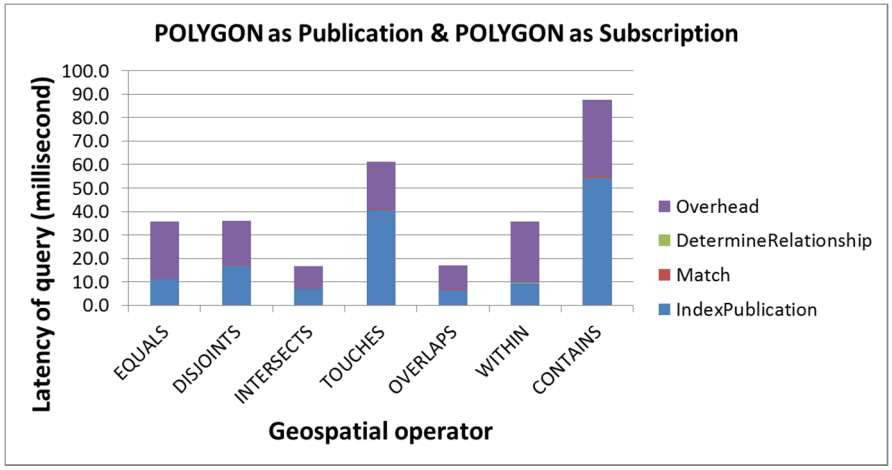
Figure 8. The end-to-end query performance for line as subscription and (**a**) point; (**b**) line; and (**c**) polygon geometry as publication.



(**a**)



(**b**)



(**c**)

Figure 9. The end-to-end query performance for polygon as subscription and (**a**) point; (**b**) line; and (**c**) polygon geometry as publication.

6. Conclusions and Future Work

We have presented the AHS model, which can efficiently determine the topological relationships between geometries in a sensor web publish/subscribe system. Due to the potentially large amount of sensor web data, the continuous query processing model is increasingly attracting interest for many time-critical applications. However, time-consuming geospatial operators are not suitable for applications that require timely processing and notification. The AHS model is an example that shows that traditional geospatial operators can be redesigned as efficient continuous query operators in the context of publish/subscribe systems.

Our evaluation results show that the proposed AHS model is 2.5 times faster than PostGIS when processing a large number of geometries, which indicates that the proposed solution is more scalable than the traditional solution. We also evaluated the indexing and matching performances of the distributed AHS model. The evaluation shows that, with the help of distributed processing, the distributed AHS model can significantly improve indexing and matching performances and finish tasks in a timely manner, even for geometries of large sizes.

Finally, we also evaluated the end-to-end query latency with relatively realistic datasets. We observed that indexing and overheads account for more than 99% of the end-to-end latency. While the overhead of applying the distributed computing process is relatively stable (between 10 to 30 milliseconds), the indexing latency varies widely depending on the size of the geometry of publications. As demonstrated earlier, the AHS model can finish most simulated queries in 100 milliseconds, thus we believe that the AHS model is able to efficiently process topological operators in a geospatial sensor web publish/subscribe system.

With regards to future directions, the current load balance approach is a simple and naïve solution. There are other factors that could be considered in the future. For example, in order to improve service availability and avoid the issue of potential machine failure, the distributed AHS model can assign multiple workers to handle the same quadkeys (i.e., replicas). In addition, as the current load balancing approach only considers the geospatial distribution of subscriptions, monitoring the geometries of publications may allow for the adaptive distribution of the processing loads. Lastly, we will also try to improve the AHS model in order to support multi-point, multi-line, and multi-polygon geometries.

Acknowledgments: The authors would like to thank CANARIE, Cybera, Alberta Innovates Technology Futures, and Microsoft Research for their support of this research.

Author Contributions: Chih-Yuan Huang designed and developed the proposed solution; Steve H. L. Liang provided the research direction; Chih-Yuan Huang and Steve H. L. Liang analyzed the results and wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Botts, M.; Percivall, G.; Reed, C.; Davidson, J. OGC® Sensor Web Enablement: Overview and High Level Architecture (OGC 07-165). Open Geospatial Consortium White Paper, 28 December 2007. Available online: http://docs.opengeospatial.org/wp/07-165r1/ (accessed on 10 February 2017).
2. Bröring, A.; Echterhoff, J.; Jirka, S.; Simonis, I.; Everding, T.; Stasch, C.; Liang, S.; Lemmens, R. New generation sensor web enablement. *Sensors* **2011**, *11*, 2652–2699.
3. Liang, S.H.L.; Huang, C.Y.; GeoCENS: A geospatial cyberinfrastructure for the world-wide sensor web. *Sensors* **2013**, *13*, 13402–13424.
4. Hart, J.K.; Martinez, K. Environmental sensor networks: A revolution in the earth system science? *Earth Sci. Rev.***2006**, *78*, 177–191.
5. Stasch, C.; Foerster, T.; Autermann, C.; Pebesma, E. Spatio-temporal aggregation of European air quality observations in the sensor web. *Towards Geoproc. Web* **2012**, *47*, 111–118.
6. Auynirundronkool, K.; Chen, N.; Peng, C.; Yang, C.; Gong, J.; Silapathong, C. Flood detection and mapping of the Thailand central plain using radarsat and MODIS under a sensor web environment. *Int. J. Appl. Earth Observ. Geoinf.* **2012**, *14*, 245–255.
7. Xu, N. A Survey of sensor network applications. *IEEE Commun. Mag.* **2002**, *40*, 102–144.
8. Hsieh, T.T. Using sensor networks for highway and traffic applications. *IEEE Potentials* **2004**, *23*, 13–16.
9. Bakillah, M.; Liang, S.H.L.; Zipf, A. Toward coupling sensor data and Volunteered Geographic Information (VGI) with agent-based transport simulation in the context of smart cities. In Proceeding of the First ACM SIGSPATIAL Workshop on Sensor Web Enablement, Redondo Beach, CA, USA, 6–9 November 2012.
10. Mainwaring, A.; Polastre, J.; Szewczyk, R.; Culler, D.; Anderson, J. Wireless sensor networks for habitat monitoring. In Proceedings of the 2002 ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, GA, USA, 28 September 2002.
11. Chen, C.P.; Chuang, C.L.; Jiang, J.A. Ecological monitoring using wireless sensor networks-overview, challenges, and opportunities. *Adv. Sens. Technol. Smart Sens. Meas. Instrum.* **2013**, *1*, 1–21.
12. Kassab, A.; Liang, S.; Gao, Y. Real-time notification and improved situational awareness in fire emergencies using geospatial-based publish/subscribe. *Int. J. Appl. Earth Obs. Geoinf.* **2010**, *12*, 431–438.
13. Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; Widom, J. Models and Issues in Data Stream Systems. Available online: http://infolab.usc.edu/csci599/Fall2002/paper/DML2_streams-issues.pdf (accessed on 10 February 2017).
14. Arasu, A.; Babcock, B.; Babu, S.; Cieslewicz, J.; Datar, M.; Ito, K.; Motwani, R.; Srivastava, U.; Widom, J.; STREAM: The Stanford Data Stream Management System. Available online: http://ilpubs.stanford.edu:8090/641/1/2004-20.pdf (accessed on 10 February 2017).
15. Mokbel, M.F.; Xiong, X.; Aref, W.G. Continuous query processing of spatio-temporal data streams in PLACE. *GeoInformatica* **2005**, *9*, 343–365.
16. Herring, J.R. OpenGIS® Implementation Standard for Geographic information— Simple Feature Access— Part 1: Common architecture (OGC 06-103r4). OpenGIS® Implementation Standard, 28 May 2011. Available online: http://www.opengeospatial.org/standards/sfa (accessed on 10 February 2017).
17. Clementini, E.; Sharma, J.; Egenhofer, M.J. Modeling topological spatial relations: strategies for query processing. *Comput. Graph.* **1994**, *18*, 815–822.
18. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113.
19. Eugster, P.; Felber, P.A.; Guerraoui, R.; Kermarrec, A.M. The many faces of publish/subscribe. *ACM Comput. Surv.* **2003**, *35*, 114–131.
20. Michelson, B.M. Event-Driven Architecture Overview, Patricia Seybold Group. Available online: http://soa.omg.org/Uploaded%20Docs/EDA/bda2-2-06cc.pdf (accessed on 10 February 2017).
21. Golab, L.; Ozsu M.T. Issues in data stream management. In Proceedings of the 2003 ACM Special Interest Group on Management of Data, San Diego, CA, USA, 9–12 June 2003.
22. Cugola, G.; Margara, A. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.* **2010**, doi:10.1145/2187671.2187677.
23. Luckham, D. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*; Addison-Wesley: Boston, MA, USA, 2002.

24. Ali, M.; Chandramouli, B.; Raman, B.S.; Katibah, E. Real-time spatio-temporal analytics using Microsoft streaminsight. In Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '10), San Jose, CA, USA, 2–5 November 2010.
25. Chen, J.; DeWitt, D.J.; Tian, F.; Wang, Y. NiagaraCQ: A scalable continuous query system for internet databases. In Proceedings of the SIGMOD Conference, Dallas, TX, USA, 15–18 May 2000.
26. Madden, S.; Shah, M.A.; Hellerstein, J.M.; Raman, V. Continuously adaptive continuous queries over streams. In Proceedings of the SIGMOD Conference, Dallas, TX, USA, 15–18 May 2000.
27. Munagala, K.; Srivastava, U.; Widom, J. Optimization of continuous queries with shared expensive filters. In Proceedings of the PODS '07, Beijing, China, 11–13 June 2007.
28. Clementini, E.; Di Felice, P.; van Oosterom, P. A small set of formal topological relationships suitable for end-user interaction. In Proceedings of 3rd International Symposium SSD '93, Singapore, 23–25 June 1993.
29. Zimbrao, G.; Souza, J.M. A raster approximation for the processing of spatial joins. In Proceedings of the 24th Very Large Data Base Conference VLDB, San Francisco, CA, USA, 24–27 August 1998.
30. Gaede, V.; Gunther, O. Multidimensional access methods. *J. Comput. Surv.* **1998**, *30*, 170–231.
31. Vivid Solutions, 2003. JTS Topology Suite Version 1.4. Technical Specifications. Available online: http://www.vividsolutions.com/JTS/bin/JTS Technical Specs.pdf (accessed on 10 February 2017).