senseable city lab:.::

SSTDE: An Open Source Semantic Spatiotemporal Data Engine for Sensor Web

Liang Yu
SENSEable City Lab, Singapore-MIT Alliance for Research and Technology
1 CREATE Way, #09-01/02,
CREATE Tower, 138602, Singapore
yuliang@smart.mit.edu

Yong Liu
Microsoft Corporation
One Microsoft Way, Redmond, WA
98052-6399, USA
yong.liu@microsoft.com

Jong Lee
National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign Urbana, IL, 61801, USA
jonglee@ncsa.illinois.edu

ABSTRACT

Recently, many tools have emerged to manage sensor web data using Semantic Web technologies for effective heterogeneous data integration. However, a remaining challenge is how to manage the massive volumes of sensor data in their semantic form, i.e., Resource Description Framework (RDF) triples. Our survey revealed that most semantic tools either do not have geospatial support, or have severe limitations on providing full GeoSPARQL support and good performance for complex queries. In this paper, we present an open source Semantic Spatiotemporal Data Engine (SSTDE), which incorporates both semantic tools and Geographic Information System (GIS) systems under a hybrid architecture. Our main contribution includes 1) introducing the sub-graph index to substitute the single node index, which results in significant performance gain for a spatiotemporal query; 2) developing a query optimization algorithm based on graph matching; 3) proposing a benchmark test for spatiotemporal query over triple stores. The spatiotemporal SPARQL query is intelligently decomposed and executed on different systems, which significantly improves the query performance by more than a hundred times comparing to other solutions.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software – *distributed systems, performance evaluation.*

General Terms

Algorithms, Management, Performance.

Keywords

Spatiotemporal Query; GeoSPARQL; Sensor Web; Graph Index; Hybrid Architecture; Semantic Triple Store.

1. INTRODUCTION

The past few years have seen a paradigm shift of geospatial data production driven by the advances of sensor technologies. Heterogeneous sensor systems have been developed which make it difficult to describe them in a uniform way. Many standards and ontologies are proposed to formalize the semantics for all sensor systems and the observations made by them, such as those published by Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE)[1]. A recent release of the Semantic Sensor Network Ontology (SSNO) further aligns them to the semantic web [1]. Success stories can be found in multiple domains such as resource management, meteorology, environment, etc., where data discovery and integration have been shown to be much easier by following those standards and ontologies. In the recently released SWE 2.0 specification, the data components are able to reference a concept by simple links which provide their semantic definition [2].

In the meantime, ubiquitous sensor networks lead to a new challenge, which is often referred to as "Big Data" in both scientific domains and business enterprises. Such data grow quickly in both size and complexity (e.g., various data structures, models, ontologies, etc.), while the solutions to them are usually contradictory, i.e., system performance is always improved by reducing the complexity and vice visa. To manage big data, one interesting technology is the NoSQL databases [3], which address the scalability from multiple perspectives using different data models, e.g., key-value, column family, graph model, etc.

To address the challenge of data complexity, interest of managing and publishing large scale sensor data through semantic web is increasing. Sheth et al. propose a semantic sensor web where the sensor data are annotated with spatial, temporal and thematic semantic metadata [5]. Linked Open Data (LOD) [6] is the most famous initiative that has been followed by many practitioners. People are encouraged to share their data using the semantic triples and also link to each other. It becomes the driving force and an ideal environment for the sensor web community to follow the ontologies and publish their data in triples [7]. Government agencies such as the United States Geological Survey (USGS) are also joining the trend by republishing their archived datasets using triples [8]. Very recently, the supercomputing company Cray funded YarcData[2] starts to push "Big Data" graph analytics using RDF and Simple Protocol and RDF Query Language (SPARQL)[3] as the industry standard. Since RDF model is one kind of graph model, thus semantic triple stores can be seen as graph databases of NoSQL series.

In particular, sensor data are always associated with spatiotemporal contexts, i.e., they are produced in specific locations and ordered by time. To process big sensor streams within a spatiotemporal context, traditional GIS functions are

[1] http://www.opengeospatial.org/ogc/markets-technologies/swe

[2] http://www.yarcdata.com/press-release-6-26-12.html

[3] http://www.w3.org/TR/rdf-sparql-query/

coupled with new technologies such as stream processing and supercomputing to meet the performance requirement [4]. Our research interest is to manage big senor data in semantic triples to achieve an optimal balance between the data size and complexity, and provide full-fledged GIS functions with them. Several known challenges include: 1) selecting ontologies for spatiotemporal semantics; 2) providing GIS functions on top of triple stores; 3) improving performance for big spatiotemporal data. Existing tools often have very limited capabilities to manage the massive geospatial data streams generated by sensor systems and to perform efficient spatiotemporal queries over them. In this paper, we present an open source middleware which combines existing triple stores and GIS tools to collectively support semantic spatiotemporal data management. It provides a common SPARQL interface to users while synchronizing data writing across multiple systems and optimizing spatiotemporal SPARQL queries at run-time. It extends our previous research on implementing a GeoSPARQL[4] interface using the Tupelo[5] triple store [9]. Our goal is to provide an independent solution for most existing semantic systems. Our contributions are as follows:

1) We propose to use a sub-graph index instead of a node index for triple stores. A spatiotemporal query is known to be very slow on existing triple stores because each single query can only employ one index. Thus, the evaluation of a complex query does not benefit much from such index strategies. Our solution is to build the spatial and temporal index as a whole, which significantly improves the performance.

2) We develop a middleware to maintain a hybrid system and to optimize the SPARQL query on-the-fly. The spatiotemporal index is created and maintained externally and updated accordingly whenever the primary database is changed. Also, we optimize the SPARQL query using graph-matching algorithms and score-based evaluation to decide how to make use of an index.

3) We propose a benchmark for the spatiotemporal SPARQL queries. Five benchmark queries are proposed to measure how a system performs as a sensor web database. To the best of our knowledge, there is no other benchmark evaluation conducted in a similar way.

2. BACKGROUND AND RELATED WORK

In the sensor web research community, people are actively using sensor ontologies because of the importance of data integration, which is highlighted by the heterogeneous sensor networks. For sensor web practitioners, there are two major concerns for utilizing the ontologies to manage data, i.e., the performance and geospatial support. Both the commercial industry and open source communities have made steady progresses on improving the performance of their products, which are reported by many published benchmark evaluations [19–24]. Also, many extensions to triple stores have been developed to provide geospatial support as we will review in this section.

2.1 Sensor Ontologies

Many ontologies have been developed in the past decade, such as Sensor Data Ontology [10], OntoSensor [11], Coastal Environmental Sensor Networks (CESN) ontology [12], etc., to name a few. Each of them is developed within different contexts and targets at different sensor systems and functions, e.g., sensor centered or observation centered. The aforementioned SSNO, launched in 2005 by W3C incubator, has been continuously evolving and the latest version was published on 28th Jun, 2011. Shortly after its release, SSNO has been used in quite a few projects in the sensor web community, such as those described in [13], [14]. Sometimes sensor ontologies are not able to provide all the semantics needed by a scientific system and additional ontologies are often required. For example, the W3C Geo Ontology[6] defines simple predicates for the WGS84 coordinates; W3C Time ontology[7] is widely used when temporal contexts are needed; the recently discussed GeoSPARQL vocabulary is a milestone for standardizing the geospatial terms using ontology; Semantic Web for Earth and Environmental Terminology (SWEET) [15] ontology developed at NASA is very popular in earth science related research, which provides many domain specific concepts that sensor ontologies do not have.

Even though those ontologies can conceptually formalize the semantics for sensor web, people have found difficulties when practically using them to develop applications. Managing semantic triples is very expensive and inefficient comparing to other traditional means [26]. The text based encoding and unrestricted structure of RDF data trade off the performance with flexibility. However, this problem is being alleviated by the continuous advances of triple store products.

2.2 Triple Stores and Geospatial Support

In the past few years, various semantic triple stores have been developed to store those ontologies as well as the data in RDF triples. Jena[8] and Sesame[9] are two leading open source semantic web projects, both of which provide high level APIs as well as backend data repositories. These two API families become predominant in this area. Even though there are other new repositories that are proved to perform much better than these two built-in repositories, these APIs still get widely supported due to their generality and flexibility.

Many examples can be found on using those triple stores to manage sensor ontologies as well as triple data. Barnaghi et al. develop their sensor data model based on the SensorML, OntoSensor as well as SWEET ontology, which is managed in Jena and Sesame repositories [17]. Huang et al. proposes a Semantic Web Architecture for Sensor Networks (SWASN) and manages the data with Jena [27]. Patni el al. use Virtuoso to manage the RDF triples for both sensor and observation data, and then publish them as linked data. Also, the provenance information is captured using their Sensor Provenance ontology [18]. Battle et al. saw the GeoSPARQL as a forthcoming standard that could unify various vocabularies and query languages developed for spatial reasoning in the past decade, and implemented it with the Parliament repository [16].

One trend for the triple stores is the geospatial support. For example, OWLIM[10] supports geospatial index in its Standard Edition (SE), and Virtuoso[11] also provides geospatial functions in its commercial edition. Several open source projects have tried to

[4] http://portal.opengeospatial.org/files/?artifact_id=44722

[5] http://tupeloproject.ncsa.uiuc.edu/

[6] http://www.w3.org/2003/01/geo/wgs84_pos

[7] http://www.w3.org/TR/owl-time/

[8] http://jena.apache.org/

[9] http://www.openrdf.org/

[10] http://www.ontotext.com/owlim

[11] http://virtuoso.openlinksw.com/

extend Jena to provide geospatial support, a recent one of which is the GeospatialWeb[12]. Neo4J[13], a graph database belonging to the NoSQL category, has been extended to manage geospatial data and semantic triples respectively in two projects – Neo4J spatial and Blueprint[14]. Open SAHARA has published the uSeekM[15] as a geospatial extension to the triples stores that support Sesame API. Different from others mentioned above, it does not provide underlying implementation for either geospatial functions or triple storage, but instead incorporates existing tools, e.g., external spatial index and triple stores, by taking advantage of the Sesame's query evaluation framework. Our implementation in this paper is based on this uSeekM project. We have also found some limitations on using it to manage the spatiotemporal sensor data, and improved it in several aspects as discussed in the later part of this paper.

2.3 Benchmark Evaluations

Evaluations on triples stores have been undertaken from different perspectives. Rohloff et al. conducted an evaluation for triples stores in 2007 [19], which include the Sesame OWLIM, and DAML DB[16], Jena, AllegroGraph[17], etc. Their conclusion and recommendation were that DAML DB and BigOWLIM had the best performance at that time. Paulson et al. did an evaluation for the provenance management purpose [20], for which both the APIs (i.e., Jena and Sesame API) and triple stores are evaluated. They chose Jena API due to its support for a wide variety of RDBMS backend since Sesame did not support at that time. There are more reports can be found in [21], [22], [23]. Also, Aquin et al. noticed that semantic web applications are expected to be running on small devices such as notebook, and then tested how different semantic tools (Jena, Sesame and Mulgara) can be used in a resource-limited environment [24].

3. LIMITATIONS AND OUR SOLUTIONS

3.1 Main Limitations

1) Most systems are reinventing the wheels for incomplete geospatial functions support. For example, Virtuoso provides 14 geospatial functions which are much fewer than PostGIS and only available through its commercial version. The Neo4j spatial does not provide a complete set of topological operators and spatial reference systems. Other databases, such as MySQL and Mongo DB, also limit their geospatial functions to a small subset, which is far from sufficient for advanced geospatial applications. It is reasonable for those systems to have some limitations because none of them is intended to be a Geodatabase. However, for people who are working on interdisciplinary area such as sensor web, it is always difficult for them to manage the spatiotemporal data within a semantic context.

2) The performance for complex SPARQL query is not acceptable, e.g., a query with many graph pattern constraints and both geospatial and temporal filters. According to our test, a spatiotemporal query takes a very long time on those systems, due to the lack of index support. A sub-query is "selective" if most of the candidates it returns are finally selected by the overall query. For a spatiotemporal query, if both the spatial and temporal sub-queries are not "selective", the overall query does not benefit much from single indexes. Most tripe stores only provide triple indexes rather than value indexes on each node. An important reason is that any single index is not selective for complex queries, which are usually the case for most of the SPARQL queries.

3.2 Our Proposed Solutions

1) **Adoption of a hybrid architecture**. Rather than rebuilding GIS functions into any specific systems and using one of them as the only solution, we adopt a loosely coupled hybrid architecture, which includes different systems managed by a middleware named Semantic Spatiotemporal Data Engine (SSTDE). The architecture of its current implementation is shown in Figure 1. The core part of SSTDE is the "Hybrid Store", which includes a "Manager" and a "Query Optimizer". The hybrid store communicates with individual sub-systems through the "Notifier" and "Evaluator", and provides standard RDF operations to end users, i.e., triple writing and SPARQL query, through a SPARQL endpoint. The "Manager" synchronizes both repositories and maintains the consistency between sub-systems by notifying all of them when adding or removing triples. The triple store manages the universal triple graph and the Geodatabase indexes the spatiotemporal data. The "Query Optimizer" analyzes the query request and creates the best query plan, which decomposes the query into segments, executes them on different subsystems through the individual "Evaluator", and finally integrates the results. The "Notifier" for each system receives commands from the manager, i.e., adding or removing triples, and updates the local storage accordingly. For the triple store, it simply delegates the same commands to the local API, while for the Geodatabase, it usually needs to project the updated universal graph to its local view by running a SPARQL query. The "Evaluator" translates a sub-query request to a local query language, execute it and bind the results to the SPARQL model. The details of the functions for each of them will be discussed in the next section. Although similar architectures have been proposed in some other researches, our focus is to provide a geospatial support and optimize the spatiotemporal SPARQL query.

2) **Use of Composite Index (Graph Index).** Composite index is widely used in RDBMS systems, e.g., indexing two columns in a table to improve the performance of queries on both of them. However, it is hard to implement on triple stores due to the unstructured nature of the graph model. However, the triple data always use some ontologies as their schemas. Even though the whole graph universe is still open and unrestricted, there exist many small immutable patterns imposed by the ontologies, which

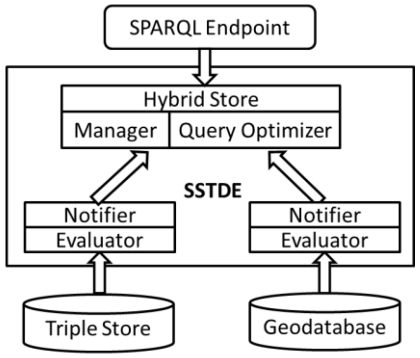


Figure 1. The hybrid system managed by SSTDE

[12] http://code.google.com/p/geospatialweb/

[13] http://neo4j.org/

[14] https://github.com/tinkerpop/blueprints/wiki/

[15] https://dev.opensahara.com/projects/useekm/

[16] http://www.daml.org

[17] http://www.franz.com/agraph/allegrograph/

are always used by SPARQL queries. Having considered this feature, we have found it highly feasible to index the sub-graphs that comply with certain patterns. When a SPARQL query is issued, the query optimizer finds out all the indexes that can be used and selects the best one. Part of the query is replaced by the index query thus the performance is improved. For a complex query, a proper composite index will be selected, which results in much more performance gain comparing to a single index.

A graph pattern out of a bigger RDF graph can be found in our previous work [14]. It includes the location of a sensor and time of the observation, which could then be used as a composite index for such spatiotemporal queries.

4. IMPLEMENTATION

Our SSTDE implementation is based on the uSeekM project. This framework helps us quickly construct a hybrid system, while we further extend it to address the limitations discussed in the above section. The basic idea is to delegate more than one sub-query to graph index query, for which a graph pattern matching algorithm is developed to optimize the query plan. Also, SSTDE maintains consistency across sub-systems while updating triples. The communication with Geodatabase is orchestrated by an "Index Manager",which is part of the "Manger" component as shown in Figure 1.

4.1 Query Delegation Model

A SPARQL query can be seen as a collection of evaluation nodes arranged in a binary tree, an example of which is shown in Figure 2. There are two types of evaluations: triple pattern matching and value filtering. Also, the intermediate nodes "Join" and "Union" are used to specify the logic relations between any two nodes. The overall evaluation is a post-order traversal of the tree, during which the data are passed from the previous node to the next. Each evaluation can be conducted individually. The idea of a hybrid system is to delegate some of those evaluation nodes to

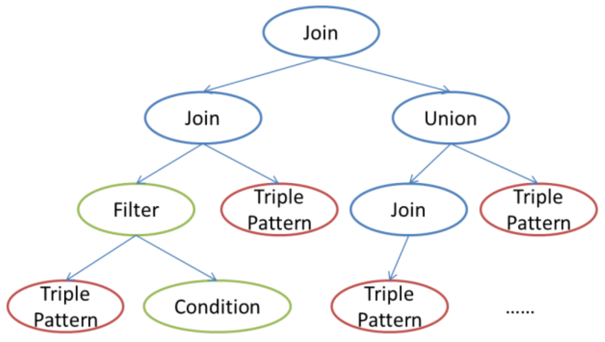


Figure 2. A SPARQL evaluation tree

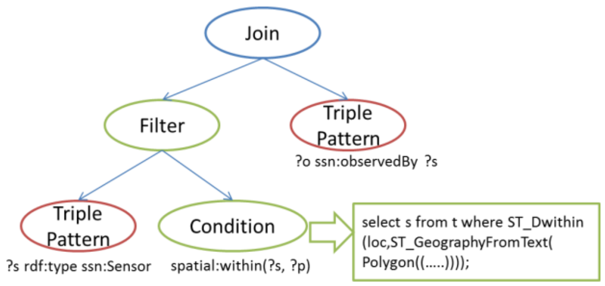


Figure 3. Delegating the evaluation nodes to different backend repositories

different backend repositories, which can provide certain function sets, e.g., geospatial functions and achieve the best performance.

Figure 3 shows an example of delegating a geospatial query to PostGIS. The uSeekM project builds on such a structure, which however can replace only one evaluation node each time. As we have discussed before, its performance for spatiotemporal query is not satisfying. Our approach is based on the graph pattern shown in SSNO, which continuously indexes the graphs when writing data, and optimizes the query based on graph matching algorithm.

4.2 Graph Pattern and Graph Index

In SPARQL specification, a graph pattern is described as a set of interconnected triple pattern. To use them in our algorithms, we formally describe these two concepts as follows:

Definition 1 (Triple Pattern): A triple pattern is defined by 3 elements: subject, predicate and object, i.e., $tp = \{s, p, o\}$. Each of the elements can be either a fixed value or a variable. They are also noted as $s(tp)$, $\mathrm{p}(tp)$, $o(tp)$.

Definition 2 (Graph Pattern): A graph pattern consists of a set of interconnected triple patterns, $gp = \{tp_1, tp_2, \ldots, tp_n\}$. For each tp_i, there must be one or more tp_j $(i \neq j)$, where $(s(tp_i) = s(tp_j)) \cup (\mathrm{o}(tp_i) = \mathrm{o}(tp_j)) \cup (\mathrm{s}(tp_i) = \mathrm{o}(tp_j)) \cup (\mathrm{o}(tp_i) = \mathrm{s}(tp_j))$. Each single triple pattern in a graph pattern is noted as $gp[m]$, $1 \leq m \leq n$, the number of triple patterns in a graph pattern g is also noted as $num(g)$.

Each graph pattern can be translated to a relational schema in the Geodatabase. Figure 4 shows how we translate a graph index configuration into a PostGIS table. The XML excerpt on the left side is customizable by users, where we can see the graph pattern is expressed by a set of triple patterns similar to a subset of a SPARQL query. Each variable is used to create a column in the table, while the constant values are still preserved by SSTDE as configurations for the latter graph matching process. The "Literal" definition is used to map the literal data types to those supported by the specific databases. Currently the SSTDE supports the mapping from XSD data types and OGC WKT types to PostGIS data types. As Figure 4 shows, the "coord" and "timevalue" are assigned with proper data types. Also, indexes are created for each column when creating the tables.

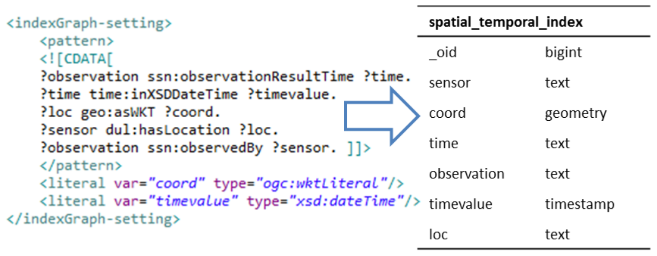
4.3 Optimization for SPARQL Query

Assuming that we have built up the graph index, Figure 5 shows the workflow for optimizing a SPARQL query.

4.3.1 Match Graph Index

Each graph index is associated with a graph pattern, so is a SPARQL query. The query optimizer needs to find out which index can be used to optimize the SPARQL query. We introduce an "inclusion" predicate to represent such a relation between two graph patterns.

Definition 3 (Element Inclusion): For each element *e* in a triple



```
<indexGraph-setting>
    <pattern>
    <![CDATA[
    ?observation ssn:observationResultTime ?time.
    ?time time:inXSDDateTime ?timevalue.
    ?loc geo:asWKT ?coord.
    ?sensor dul:hasLocation ?loc.
    ?observation ssn:observedBy ?sensor. ]]>
    </pattern>
    <literal var="coord" type="ogc:wktLiteral"/>
    <literal var="timevalue" type="xsd:dateTime"/>
</indexGraph-setting>
```

spatial_temporal_index	
_oid	bigint
sensor	text
coord	geometry
time	text
observation	text
timevalue	timestamp
loc	text

Figure 4. Creating a Table in PostGIS for a Graph Index

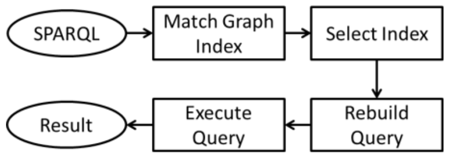


Figure 5. Workflow of SPARQL Query Optimization

pattern *tp*, i.e., $e \in \{s(tp), p(tp), o(tp)\}$, if e_i is a variable, or $e_i = e_j$, then e_i includes e_j, noted as $e_i \geq e_j$.

Definition 4 (Pattern Inclusion): A triple pattern tp_i includes tp_j, noted as $tp_i \geq tp_j$, if $(s(tp_i) \geq s(tp_j)) \cap (p(tp_i) \geq p(tp_j)) \cap (o(tp_i) \geq o(tp_j))$.

Definition 5 (Sub-Graph Pattern): If the collection of triple patterns of graph pattern gp_i is a subset of that of gp_j, then gp_i is a subset pattern of gp_j, noted as $gp_i \subseteq gp_j$.

Definition 6 (Graph Inclusion): A graph pattern gp_i includes gp_j, noted as $gp_i \geq gp_j$, if there exists one or more graph $gp_k \subseteq gp_j$, $num(gp_k) = num(gp_j) = n$, and for each $1 \leq m \leq n$, $gp_j[m] \geq gp_k[m]$, then gp_j includes gp_k, noted as $gp_j \geq gp_k$. Also, $gp_j \geq gp_i$.

Definition 7 (Graph Algebra): For graph pattern gp_i and gp_j, $gp_i - gp_j$ removes every triple pattern in gp_j from gp_i.

According to definition 6, the essence of the graph matching algorithm is to find out one or more sub-graph pattern from the SPARQL that is included by the indexed graph pattern. The evaluation of the selected sub-graph pattern will be replaced by an index query. Figure 6 shows an excerpt of the pseudo code that selects all the possible sub-graph patterns from the first input that is included by the second input. The "`matchGraph`" method returns a set of matched subset patterns from the input graph1 that are included by graph2. It iterates all the possible combinations of triple patterns in graph1 that form the required subset patterns. The "`removePattern`" method is used by "`matchGraph`" to remove a triple pattern that has already been matched from the candidate graph pattern it belongs to.

To save the space, the pseudo code does not show the name

```
Set<Graph>   matchGraph(GraphPattern   graph1,   GraphPattern
graph2){
   Initialize colGraph as a set of graph patterns;
   Add graph1 to colGraph
   FOR EACH triple pattern tp in graph2
      Initialize colGraphT as a temporary graph pattern set;
      FOR EACH gCand in colGraph
         Collection<Graph>  candArray  =  removePattern(gCand,
tp);
         Add all elements of candArray to colGraphT;
      END FOR
      Replace colGraph with colGraphT;
   END FOR
   FOR EACH graph g in colGraph
      replace g with (graph1-g);
   END FOR
   RETURN colGraph;
}
Set<Graph> removePattern(GraphPattern g, TriplePattern tp){
         Initialize colGraph to be a collection of graph
         FOR EACH triple pattern tpg in g and tpg ≥ tp
                  produce gc as a copy of g
                  remove  tpg  from  gc  and  add  it  the
                  colGraph;
         END FOR
         RETURN colGraph;
}
```

Figure 6. Pseudo code for finding sub-graphs

mapping process, which maps the variable names used in graph1 and graph2 for each matched subset pattern. It is used to create an actual SQL query clause when one index is finally selected for optimization.

4.3.2 *Select Index*

The "`matchGraph`" method returns a set of matched graph patterns. A further selection is needed to decide which one is the best. Most triple stores provide triple indexes, thus the essential purpose of the graph index is to optimize the filters, which in general, can be classified into three categories: numeric filter, text comparison and function call. If a graph pattern contains variables used in a filter, we associate that filter to it. For each matched graph pattern, the numbers of the filters of each of the above types are denoted as *nfn*, *nft*, *nff*. Each filter type is assigned with different weight values, i.e., *wfn*, *wft*, *wff*, and the score can be calculated as follows:

$$s = nfn \times wfn + nft \times wft + nff \times wff$$

This score is used to select the best matched graph pattern from the collection returned by the matchGraph method. Further, if there is more than one index defined in the configuration file, the matchGraph will be invoked multiple times with each index graph as its second input. The best pattern will then be selected from all the returned sets. Currently, we focus on the spatiotemporal query, so that the spatial functions, e.g., within, contains, etc., are usually the most selective. We then assign the weight values as: *wfn* = 100, *wft* =10, *wff*=10.

4.3.3 *Rebuild Query*

If there is no available index for a query, it will be executed in a default manner, i.e., evaluating the graph patterns on the triple store and filtering each of the candidates. Otherwise the best index will be used to optimize the query as follows:

1) Generate an index evaluation object, which translates the query on the indexed graph pattern to a SQL. The filters are translated to local functions, e.g., regular expression, PostGIS spatial functions, etc. The modifiers, i.e., "limit", "order by" are also carried to the SQL. As aforementioned, the translation is based on the name mapping process.

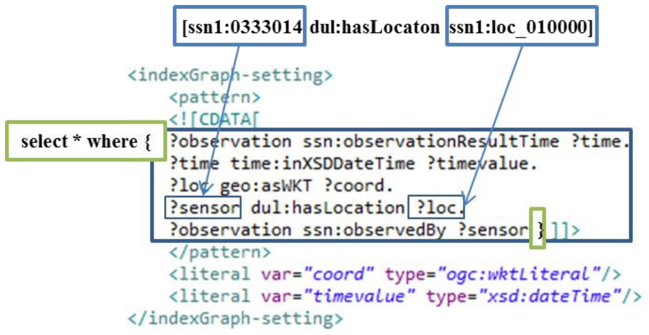
2) Substitute the matched graph pattern in the SPARQL as well as its associated filters with the index evaluation object. All the filters and matched triple patterns are removed from the evaluation tree. In terms of Sesame API, the filters are replaced with a constant Boolean value "true", while the triple patterns are removed by replacing their parent nodes with their sibling nodes.

3) Move the index evaluation node to the most left-bottom position, to make it the first one to be visited in a post-order traversal process.

The execution of the query will be conducted by the Sesame API, which visits the evaluation nodes in a post-order, and pass the data from one node to the next. The indexer needs to bind the data back to the variables in the SPARQL, where the name mapping is used again.

4.4 Synchronization of Data Update

SSTDE dynamically creates tables in the PostGIS for each graph index as shown in Figure 4. Each field is indexed according to their types. To update the index database while writing triples, a decision should be made on whether adding or deleting a triple leads to any changes to the indexed data. This is performed by the "Notifier" in Figure 1.



```
<indexGraph-setting>
    <pattern>
    <![CDATA[
    ?observation ssn:observationResultTime ?time.
    ?time time:inXSDDateTime ?timevalue.
    ?loc geo:asWKT ?coord.
    ?sensor dul:hasLocation ?loc.
    ?observation ssn:observedBy ?sensor }]]>
    </pattern>
    <literal var="coord" type="ogc:wktLiteral"/>
    <literal var="timevalue" type="xsd:dateTime"/>
</indexGraph-setting>
```

Figure 7. SPARQL query for updating the index

1) If the indexed graph pattern contains only one single triple pattern, the updated triple is compared with this triple pattern and then the index data is updated accordingly.

2) If the graph pattern contains more than one triple pattern, a SPARQL query will be executed to find out if there is an instance of the graph pattern to be added or removed.

The SPARQL query is built by the graph pattern, but some variables are replaced by the concrete values in the updated triple. The query result will be either added or removed from the index. An example is shown in Figure 7, which uses the graph index defined in Figure 4. The graph pattern is simply wrapped into a SPARQL query clause, and the variables "sensor" and "loc" are replaced with the values from the updated triple.

4.5 Index Manager

The index manager is the essential part of the "Manager" in Figure 1. Its main functions include:

1) Notifying different index to update its local data. The maintenance of the hybrid system is expensive, which needs to run several queries when writing a single triple. The index manager optimizes this by indexing the indexes with the predicates in their triple patterns. It quickly identifies if an updated triple would probably lead to new graph instances.

2) Selecting the best index for a query from the index list. If multiple indexes are matched for a query, the index manager selects the one that has the highest score and uses it for optimization.

3) Updating the index in a batch mode. In a transaction mode, batch processing significantly improves the performance.

4) Supporting various vocabularies. It is very easy to change the literal type "ogc:wkt" to another proprietary URI created by individual persons or companies. Also, the GeoSPARQL predicates are also very easy to be replaced.

5. BENCHMARK EVALUATION

Before developing the SSTDE, we have conducted a review and hands-on testing on some existing triple stores in terms of spatiotemporal query support and performance. We then narrowed down to a few candidates that support the spatial data type and spatial index. We focus on the open source community so that commercial products are not considered. The final comparison list includes three candidates: OpenRDF + uSeekM, Neo4j + Blueprint + uSeekM, and Parliament. In theory, uSeekM supports any triple stores that provide Sesame API implementations, but there are some problems with specific products. For example, Virtuoso does not support a query evaluation using external data binding in its Sesame API, and Bigdata has a limitation on the length of each triple element, which makes it impossible to store a long geospatial coordinate string. Through our testing, we have found that the current uSeekM implementation does not optimize the Neo4J + Blueprint, i.e., the spatial triple pattern is not the first one to be evaluated, so that we modify the source code accordingly and add it to the solution list. The testing data includes both sensor and observation data, which are converted to RDF triples according to the SSN ontology and GeoSPARQL vocabulary. The testing data include national maps from USGS, event data from National Oceanic and Atmospheric Administration (NOAA), sensor data from CUAHSI WaterOneFlow Web service, etc., as listed in [14]. Table 1 lists the statistical information of the testing data.

Table 1. Statistics of the testing data

Item	Count
Total triples	6,828,735
Total geometries	45,832
Polygons	14,380
Line	27,727
Point	3,725

We use five testing queries with different GeoSPARQL predicates and graph patterns, which are listed as follows:

1) A query to find the geometries within a polygon.

```
SELECT ?geometry ?wkt WHERE {
?geometry geo:asWKT ?wkt.
FILTER(geo:within(?wkt, "POLYGON((-91 33, -75 33, -75
42, -91 42, -91 33))"^^geo:wkt))}
```

2) A query to find the sensors within the Illinois state. It uses the real coordinate string converted from USGS National Map.

```
SELECT ?sensor ?coord WHERE {
?sensor rdf:type ssn:Sensor.
?sensor dul:hasLocation ?loc.
?loc geo:asWKT ?coord.
FILTER(geo:within(?coord, "POLYGON((-90.6415100097656
42.5092811584473,-90.6359710693359
42.5093994140625,...))"^^ogc:wktLiteral))}
```

3) A query to find the tornados that intersects with the Illinois state.

```
SELECT ?tornado ?path WHERE {
?tornado rdf:type phe:Tornado.
?tornado spa:hasGeometricalObject ?pathobj.
?pathobj geo:asWKT ?path.
FILTER(geo:intersects(?path, "POLYGON((-
90.6415100097656 42.5092811584473,-
90.6359710693359....))"^^ogc:wktLiteral))}
```

Table 2. Ontology prefixes used in the queries

Prefix	URI
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
time	http://www.w3.org/2006/time#
ssn	http://purl.oclc.org/NET/ssnx/ssn
dul	http://www.loa-cnr.it/ontologies/DUL.owl
geo	http://www.opengis.net/ont/OGC-GeoSPARQL/1.0/
ogc	http://www.opengis.net/
flu	http://sweet.jpl.nasa.gov/2.2/realmLandFluvial.owl#
spa	http://sweet.jpl.nasa.gov/2.0/spaceObject.owl
phe	http://sweet.jpl.nasa.gov/2.2/phenAtmoPrecipitation

4) A query to find all the watersheds that are contained by Illinois State.

```
SELECT ?watershed ?boundary WHERE {
?watershed rdf:type flu:Watershed.
?watershed spa:hasGeometricalObject ?bdobj.
?bdobj geo:asWKT ?boundary.
FILTER(geo:contains("POLYGON((-90.6415100097656
42.5092811584473,-
90.6359710693359.....))"^^ogc:wktLiteral, ?boundary))}
```

5) A query to find all the observations made by all the sensors within Illinois and ordered by the observation time.

```
SELECT ?observation ?timevalue ?coord ?property ?value
?sensor WHERE {
?observation rdf:type ssn:Observation.
?observation ssn:observedProperty ?property.
?observation ssn:observationResult ?result.
?observation ssn:observationResultTime ?time.
?time time:inXSDDateTime ?timevalue.
?result ssn:hasValue ?value.
?value dul:hasDataValue ?datavalue.
?observation ssn:observedBy ?sensor.
?sensor rdf:type ssn:Sensor.
?sensor dul:hasLocation ?loc.
?loc geo:asWKT ?coord.
FILTER(?timevalue >= "2011-05-15T23:30:00.000-
06:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>)
FILTER(geo:within(?coord, "POLYGON((-90.6415100097656
42.5092811584473,-90.6359710693359 42.5093994140625,-
90.6353530883789 ....))"^^ogc:wktLiteral))} order by
desc(?timevalue)
```

The coordinate string for Illinois State is shortened in the query examples. Table 3 shows the testing results in terms of running time (milliseconds) for different solutions. All the queries are executed 10 times and the results are averaged. It is apparent that uSeekM performs much better than the other three candidate solutions, while the SSTDE further improves it significantly for the queries 2 to 5. Query 1 is very simple as SSTDE and uSeekM optimize it in a similar way. SSTDE performs better for Queries 2 to 4 because it replaces both the triple pattern evaluations and filters with the index query, while the uSeekM only replaces the filters. The query 5 is a spatiotemporal query and shows the most distinct results. uSeekM stands on the second place, which is almost 200 times slower than SSTDE.

6. DISCUSSION

SSTDE achieves significant performance gain for spatiotemporal SPARQL query. However, there are still some limitations which were found during the evaluation and need to be addressed:

1) The size of the hybrid store grows fast. The size of the testing data is about 5G in OpenRDF but only 1G in JSON files. A further evaluation on the balance between data size and data access performance for existing triple stores is needed. On the other hand, the graph index is sometimes redundant, e.g., the sensor locations are recorded multiple times for each observation. It is reasonable to allow users to optimize the index data structure.

2) Maintaining the consistency in the hybrid system introduces additional overhead. Comparing to direct writing to triple stores, the performance is reduced due to the additional index. Synchronizing the data writing over multiple systems is expensive. Asynchronous and batch processing could be possible solutions, which would cause the "transient inconsistency" for the overall system, but the goal is to keep it in an acceptable range.

3) The performance of PostGIS reduces when the data size increases. This is reasonable but an interesting observation is that a Solid State Drive (SSD) provides much better performance than a common hard disk. The performance of SSD is close to a constant value while that of a common hard disk decreases rapidly. Thus, more effort should be put on optimizing the PostGIS, e.g., clustered database.

4) The scalability is limited by both triple stores and Geodatabase. SSTDE frequently issues SPARQL queries to synchronize the data writing, for which high performance triple indexes are essential. Despite the advances of triple stores on managing billions of triples, the limitation will still be easily reached for a single tripe store. Distributed triple stores, as well as distributed spatial indexes are the promising solution.

7. CONCLUSION AND FUTURE WORK

This paper presents our recent effort on leveraging the semantic tools to manage the sensor data and building applications on top of them. The current implementation is available at [25]. Since existing systems are not able to handle complex spatiotemporal queries in an acceptable turn-around time, we develop the SSTDE as a middleware that incorporates the semantic repositories and a traditional Geodatabase under a hybrid framework.

As more and more people have realized that no single solution could meet all the requirements in terms of the data size, complexity and real time performance, we expect to investigate a much larger scale system, where multiple types of backend repositories are leveraged, and the data engine should be

Table 3. Execution time (millisecond) for 5 queries by different solutions

Solution	Q1	Q2	Q3	Q4	Q5
OpenRDF + uSeekM	493	2109	408	540	40942
Neo4j+BluePrint+uSeekM	220493	755412	376825	1307944	N/A
Parliament	28363	12556	3752	22479	44879
Neo4j+BluePrint+ Optimized uSeekM	4418	7346	6804	6652	2065644
SSTDE	**445**	**73**	**35**	**43**	**221**

intelligent enough to know how to maintain the system and optimize the query through a simple SPARQL endpoint. The experiences we have gained in this research are very valuable for handling the future Big Data challenge.

8. ACKNOWLEDGMENTS

The authors thank Microsoft Research and the Institute for Advanced Computing Applications and Technologies at the University of Illinois at Urbana-Champaign for partially funding this work, as well as National Research Foundation Singapore through Singapore-MIT Alliance for Research and Technology's Future Urban Mobility group for the support for writing this paper.

9. REFERENCES

[1] Compton, M., et al., "The SSN Ontology of the W3C Semantic Sensor Network Incubator Group", Journal of Web Semantics, 2012.

[2] Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., and Lemmens, R., "New generation Sensor Web Enablement," Sensors, vol. 11, pp. 2652-2699, 2011.

[3] Leavitt, N. Will NoSQL Databases Live Up to Their Promise? Computer, 2010, 43:12–14.

[4] Kazemitabar, S.J., Banaei-Kashani, F., McLeod, D. Geostreaming in cloud. (2011) Proceedings of the 2nd ACM SIGSPATIAL International Workshop on GeoStreaming, IWGS 2011, pp. 3-9.

[5] Sheth, A., Henson, C., Sahoo, S.: Semantic Sensor Web. Internet Computing, IEEE 12(4) (2008) 78–83

[6] Phuoc, D.L., Hauswirth, M.: Linked Open Data in Sensor Data Mashups. In Kerry Taylor, Arun Ayyagari, D.D.R., ed.: Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09) in conjunction with ISWC 2009. Volume Vol-522., CEUR (2009)

[7] Kessler, C. and Janowicz, K. (2010), Linking Sensor Data - Why, to What, and How?. In Kerry Taylor, Arun Ayyagari, David De Roure (Eds.): The 3rd International workshop on Semantic Sensor Networks 2010 (SSN10).

[8] Wei, M., Zhao, T., Usery, E.L., and Varanka, D. A Conceptual Design Towards Semantic Geospatial Data Access GIScience, September, 2008

[9] Lee, J., Liu, Y., Yu, L. SGST: An Open Source Semantic Geostreaming Toolkit. IWGS '11 Proceedings of the 2nd ACM SIGSPATIAL International Workshop on GeoStreaming, 2011. pp. 17-20

[10] Eid, M., Liscano, R. and Saddik, A.E. A Universal Ontology for Sensor Networks Data. In 2007 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications, pp. 59-62, 2007.

[11] Goodwin, J.C, Russomanno, D.J., Qualls, J. Survey of semantic extensions to UDDI: implications for sensor services. In Proceedings of the International Conference on Semantic Web and Web Services, pp. 16-22, 2007.

[12] Calder, M., Morris, R.A., Peri, F. Machine reasoning about anomalous sensor data. In Ecological Informatics 5(1), pp. 9-18, 2010.

[13] Calbimonte, J.P, Jeung, H, Corcho, O., Aberer, K. Semantic Sensor Data Search in a Large-Scale Federated Sensor Network. In Proceedings: 4th International Workshop on Semantic Sensor Networks 2011. 23 October, 2011. Bonn, Germany.

[14] Yu, L and Liu, Y. Using Linked Data in a Heterogeneous Sensor Web: Challenges, Experiments and Lessons Learned. Workshop on Sensor Web Enablement 2011. October 6th and 7th. Banff, Alberta, Canada.

[15] Raskin, R. and Pan, M.J. Knowledge representation in the Semantic Web for Earth and Environmental Terminology (SWEET). Computers & Geosciences 31, pp:1119-1125.

[16] Battle, R., Kolas, D. Enabling the geospatial Semantic Web with Parliament and GeoSPARQL. Accepted by Semantic Web – Interoperability, Usability, Applicability an IOS Press Journal. Accessed at http://www.semantic-web-journal.net/sites/default/files/swj176_3.pdf

[17] Barnaghi, P., Meissner, S, Presser, M., Moessner, K. Sense and sens'ability:semantic data modelling for sensor networks. In ICT Mobile Summit, 2009.

[18] Patni, H., Sahoo, S.S., Henson, C., Sheth, A.: Provenance Aware Linked SensorData. In Kärger, P., Olmedilla, D., Passant, A., Polleres, A., eds.: Proceedings of the 2nd Workshop on Trust and Privacy on the Social and Semantic Web, Heraklion, Greece, May 31, 2010.

[19] Rohloff, K., et al. An evaluation of triple-store technologies for large data stores. In: On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, 1105-1114.

[20] Paulson, P., Gibson, T., Schuchardt, K., Stephan, E. Provenance Store Evaluation. 2008. accessed at http://www.pnl.gov/main/publications/external/technical_reports/PNNL-17237.pdf

[21] The National Center for Biomedical Ontology. Comparison of Triple Stores. 2009. Accessed at http://www.bioontology.org/wiki/images/6/6a/Triple_Stores.pdf

[22] Revelytix, Inc. Triple Store Evaluation Performance Testing Methodology. Accessed at http://www.revelytix.com/sites/default/files/TripleStoreEvaluationAnalysisResults.pdf

[23] Berlin Free University, Berlin SPARQL Benchmark (V3 Results). 2011. Accessed at http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V6/Vincent

[24] D'Aquin, M., Nikolov, A., Motta, E. How much Semantic Data on Small Devices?. In: EKAW 2010 Conference - Knowledge Engineering and Knowledge Management by the Masses, 2010.

[25] SSTDE source base. https://github.com/SSTDE/0.1.

[26] Schmidt, M. *et al*. An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In Proceedings of the International Semantic Web Conference (ISWC 2008), 2008

[27] Huang, V., and Javed, M.K. Semantic sensor information description and processing. In 2nd International Conference on Sensor Technologies and Applications, 2008.