LYRIC: Deadline and Budget Aware Spatio-Temporal Query Processing in Cloud

Jaydeep Das, Shreya Ghosh, *Student Member, IEEE*,
Soumya K. Ghosh, *Senior Member, IEEE*, and Rajkumar Buyya, *Fellow, IEEE*

Abstract—With the enormous growth of wireless technology, and location acquisition techniques, a huge amount of spatio-temporal traces are being accumulated. This dataset facilitates varied location-aware services and helps to take real-life decisions. Efficiently handling and processing spatio-temporal queries are necessary to respond in real-time. Processing the vast spatio-temporal data requires scalable computing infrastructure. In this regard, an efficient query resolution system can be deployed if we predict the infrastructure requirement of the user query apriori along with the identification of the geospatial service chain. In this work, we propose a framework, namely *LYRIC* (deadLine and budget aware spatio-temporal querY pRocessing In Cloud), where the spatio-temporal queries are resolved efficiently considering user-defined deadline and budget constraint. Our framework shows high deadline completion accuracy in the range of 1.0 - 0.937, which is more accurate than SparkGIS, GeoSpark, GeoMesa and JUST. This also reduces the resource prediction error by 11 percent, considering the geospatial service chain than without it. The cost of the spatio-temporal query is reduced by ≈23% in LYRIC, further, the simulation study (using CloudSim) illustrates the efficacy and scalability of LYRIC in terms of optimal budget usage and execution time compared to four baseline approaches.

Index Terms—Spatio-temporal query, geospatial service chain, cloud computing, query budget constraint, user deadline

1 Introduction

The huge volume of spatio-temporal data instances has motivated the data science community to analyze and utilize the underneath knowledge. Spatio-temporal data-set consists of *objects* and *events* in spatial (location) and temporal (timestamp) context. These spatio-temporal data sources open up unprecedented opportunities to extract and leverage the usable knowledge and utilize it for smart living such as route planning, trip recommendation, weather prediction, etc. However, managing this huge volume of spatio-temporal data and obtaining optimized query performance is inevitably challenging tasks. There are several challenges in spatio-temporal query processing. First, unlike conventional database, the attributes of spatio-temporal database have different structure (*geometry*), such as *polygon*, *polyline* [1] etc. The *processing cost* of accessing a record in the spatio-temporal database depends on the *spatial* and *temporal* extent of the query itself. Therefore, an effective query processing framework is necessary to retrieve information from these huge spatio-temporal datasets. Moreover, Cloud paradigm is suitable to leverage the *pay-as-you-go* model based on the resources used in query processing.

In this regard, the primary objective of this work is to propose an effective spatio-temporal query processing framework, which is capable of providing query response within the user's deadline and budget. When an organization or a user submits a task containing bulk queries, the processing needs to be resolved within the *user-deadline* and *budget*. This *user-deadline* is the time frame provided by the user to get the query result from the time of the submission, and *budget* is the total price (example: Price of Google Cloud Platform services[1]) incurred for utilizing the compute, storage, and/or software services of the cloud servers. The query processing techniques must be optimized to store, search, and query the records defined in geographical space and time intervals. The traditional query processing tools do not work well with spatio-temporal databases due to the complex geometric structures and computations. On the other side, spatio-temporal query processing requires varied OGC[2] compliant geospatial services (Feature, Processing, Map services) to respond. Each of those geospatial services has a separate processing cost and execution time. For instance, say a user submits bulk-query with $10\,mins$ deadline, and $\$20$ budget threshold. The task requires 3 feature services and map services. The price of each of the services (deployed in the cloud) is $\$2$ for $10\,mins$ timespan. However, it is observed that the task can not be completed within the deadline utilizing the present configurations of the services. In such a scenario, a proper query processing plan, such as adding more compute resources for the feature service to reduce the time, needs to be adapted. However,

- *Jaydeep Das is with the Advanced Technology Development Centre, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal 721302, India. E-mail: jaydeep@iitkgp.ac.in.*
- *Shreya Ghosh and Soumya K. Ghosh are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal 721302, India. E-mail: shreya.cst@gmail.com, skg@cse.iitkgp.ac.in.*
- *Rajkumar Buyya is with the CLOUDS Laboratory, and the School of Computing and Information Systems, University of Melbourne, Melbourne, VIC 3010, Australia. E-mail: rbuyya@unimelb.edu.au.*

Manuscript received 16 July 2020; revised 6 Mar. 2021; accepted 4 Apr. 2021. Date of publication 13 Apr. 2021; date of current version 7 Oct. 2022.
(Corresponding author: Jaydeep Das.)
Digital Object Identifier no. 10.1109/TSC.2021.3073006

1. Google Cloud Pricing: https://cloud.google.com/pricing/list
2. Open Geospatial Consortium: https://www.ogc.org

1939-1374 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

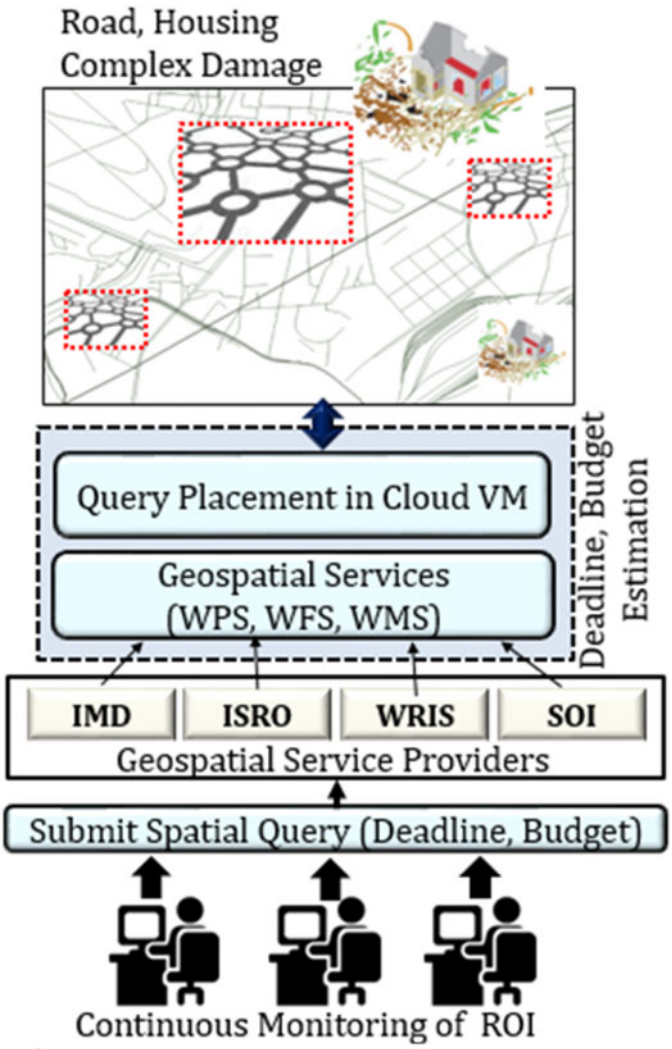

Fig. 1. Motivating scenario.

selecting an appropriate query execution plan considering both deadline and budget is difficult when several geospatial services are required to resolve the query. To address the issue, LYRIC implements *cooperative game theory* where the objective is to complete the task within the deadline and reducing the overall user budget. In other words, using the minimal resources [2] for the query processing in cloud servers within the user-defined deadline.

1.1 Motivating Example

Fig. 1 illustrates a motivating scenario. In the time of exigency (say, super-cyclone *Amphan*[3]), the normal lives are disrupted due to power cut, shortage of water supply, road blockage or even residential place collapse. In such a situation, several departments of state/ central govt. (such as electricity, communication, railway, highway, transportation, water resources, etc.) help seamlessly to *National Disaster Response Force (NDRF)* to continuously monitor the situation and taking appropriate steps to get back the normalcy. Extracting spatio-temporal information seamlessly from various data sources need proper spatio-temporal query execution and orchestration of geospatial services. For instance, national agencies of the Indian government, such as GSI[4] (Geological Survey of India), India-WRIS[5] (Water Resources Information System), SOI[6] (Survey of India), IMD[7] (Indian Meteorological Department), or ISRO[8] (Indian Space Research Organization) provide varied real-time feature services, data services, and map services to retrieve the present situation of the affected region. LYRIC provides a query execution plan considering the user's deadline and budget. This resolves the query seamlessly utilizing geospatial services in the cloud. Here, we provide an example scenario of an exigency situation in the Indian context. Our proposed framework is also suitable for any spatio-temporal query processing task with user budget and deadline constraints [3].

1.2 Contributions

The key contributions of this paper are as follows:

1) We propose an end-to-end framework, named *LYRIC*, to resolve the spatio-temporal query within the deadline and budget provided by the user. The framework analyzes the query and orchestrates several geospatial services required to resolve the query.
2) The framework is conducive to decompose the query into several components automatically. It generates the query parse tree and identifies the geospatial services, and builds *geospatial service chains* for the processing. Further, it predicts the resource requirements for resolving the spatio-temporal query efficiently.
3) LYRIC proposes a novel method of choosing an appropriate query execution plan using *cooperative game theory*. The query execution plan provides the configuration of the VMs in the cloud to run the geospatial services and generates the query result considering the deadline and the user's budget.
4) The framework has been implemented and tested using spatio-temporal traces in the laboratory testbed. The experimental observations yield encouraging results in terms of accuracy of task (bulk query processing) completion within the deadline, reduced delay in the query response, and reduced memory and CPU usages.

The rest of this paper is organized as follows. Section 2 discusses the related existing works. Section 3 presents the system model of our work, where we discussed spatio-temporal query types, geospatial service chaining, and its utility. We also define the cost model of spatio-temporal queries. Section 4 elaborates on the performance evaluation with experimental setup and results. Section 5 concludes the paper with future direction.

2 Related Work

This section discusses the existing works about query processing, geospatial web services, and resource management in the cloud. Marcus *et al.* [4] use supervised learning

3. Super cyclone Amphan caused huge damage in eastern India. https://en.wikipedia.org/wiki/Cyclone_Amphan

4. https://www.gsi.gov.in
5. https://indiawris.gov.in/wris/
6. http://www.surveyofindia.gov.in/
7. https://mausam.imd.gov.in/
8. https://www.isro.gov.in/

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

techniques for batch processing and reinforcement learning techniques for online processing of user queries. This learning technique has achieved query scheduling with proper cost and performance management, which satisfies the service level agreement(SLA) of user and service provider. Many researchers have also analyzed the performance and latency of analytical queries through machine learning techniques [5], [6], [7], [8].

A graph-based temporal relationship between entities like edge, vertices, properties has been proposed in [9]. Their approach is specifically beneficial for path queries over dynamic temporal graphs. The authors have utilized Granite distributed engine over Graphite ICM platform for experiments. Another graph embedded query performance prediction for concurrent queries has been proposed by [10]. This research also used the graph update and compaction algorithm to determine the query workload. Chu *et al.* [11] predict the query execution time using LSTM architecture in graph database. in this work, for encoding the query plan tree, they have utilized a post-order traversal algorithm, while RF and PCA help to perform feature engineering.

A resource modeling approach to measuring concurrent query performance is proposed by Duggan *et al.* [12], and prediction under concurrency is made in [13], [14], [15]. Concurrent Query Intensity (CQI) and Query Sensitivity (QS) are two matrices that determine the latency of concurrent queries. CQI helps to know how the resources are shared among concurrent queries, and QS defines how the query functions are changed in case of resource shortage. Popescu *et al.* [16] propose a framework to predict the runtime performance for a set of queries with a different dataset. They segment the queries and measure the performance using the machine learning model. Further, they attempt to predict the overall query runtime. The authors have considered only the tuple size and cardinality of the different datasets for estimating execution time.

Geospatial semantics and service-oriented architecture (SOA) based automatic compositions of geospatial services have been presented in [17]. DataType, serviceType, and association type ontologies have been used as a semantic schema in SOA. The authors have used geospatial services for ontology design, composition building, and semantic analysis. Geospatial services are used for knowledge transformation [18], [19], [20] using geospatial modeling, model instantiating, and model execution. Geospatial service orchestration in the cloud platform is described in [21], [22]. A cloud and agent-based geospatial service chain is proposed by [23], where geospatial tasks are executed with agents' movement in a single cloud environment. Agents act as part of the chain and interact with individual geospatial services. It prevents huge volumes of data transfer and service chain failure. A learning technique has been opted for allocating the virtual machines in the cloud platform in [24]. Web service composition-related literates have been carried out in [25].

Although there are several research works in this domain, all of these existing literature have some limitations. First of all, there is no clear indication of how performance characteristics (execution time and resource usages) of spatio-temporal queries can be predicted. As discussed earlier, the prediction of the performance of spatio-temporal queries is not straightforward. Most of the works need execution-time statistics of the queries and the count of the tuples processed. While this method adds more overhead, the simple count of tuples does not work in spatio-temporal queries. *In brief, the contributions of LYRIC are manifolds. First, it is capable of decomposing the queries into different segments and identifies several spatial-services. Our framework deploys a novel performance characteristics prediction technique and provides a query-plan to complete the query at minimal cost within the deadline.*

3 System Model

This paper has taken up the spatio-temporal query processing in cloud considering the user given deadline and budget to resolve the query. Spatio-temporal queries are generated by the user in bulk and submitted to the framework through a user interface. The query parser module breaks the query into a query tree with spatial and temporal information. Next, geospatial services are identified from the query tree, and a service chain is generated for processing. On the other side, decomposing the query helps to identify the resource (RAM, CPU cores, storage) requirement for processing spatio-temporal queries and predicting the execution time. The proposed framework, LYRIC, also considers the users' priority in resolving the query. The priority is determined by two parameters provided by the user: (i) deadline and (ii) budget to resolve it. LYRIC analyzes all of these factors and offers a query execution plan resolving the task within the deadline incurring minimal budget. Spatio-temporal queries are placed into Cloud VM, and finally, the results of queries are sent back to the user through the query interface. The overall activities of our approach are illustrated in a block diagram (Fig. 2).

The query parser generates the query tree for the incoming spatio-temporal query. The nodes of the query tree determine the types and number of required geospatial services. After identifying the geospatial services, LYRIC generates the service chain. Here, we have provided different queries based on whether it requires sequential processing of the services or parallel service processing. After generating the geospatial service chain for the particular query, LYRIC provides the query execution plan based on the users' priority. This geospatial service chain formation is one of the key modules that help to allocate the virtual machine to resolve the query effectively.

We also determine the resources like RAM, CPU cores, the storage requirement for a spatio-temporal query from the query tree, and predict the query execution time. Next, we use game theory to find a proper query processing plan to resolve the spatio-temporal query within the user-defined deadline and budget. We have shown the sequence diagram of the overall activity in Fig. 3.

3.1 Spatio-Temporal Query Types

Spatio-temporal query type identification is essential to estimate the resource (RAM, CPU cores, storage) requirement for the spatio-temporal query or batch of queries efficiently. We need to know the amount of spatio-temporal data that has to be processed to resolve the spatio-temporal query. The spatio-temporal data amount depends upon the number of tables selected for the spatio-temporal query. According to the number of table selections, we categorize the spatio-temporal queries into the following two types.

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

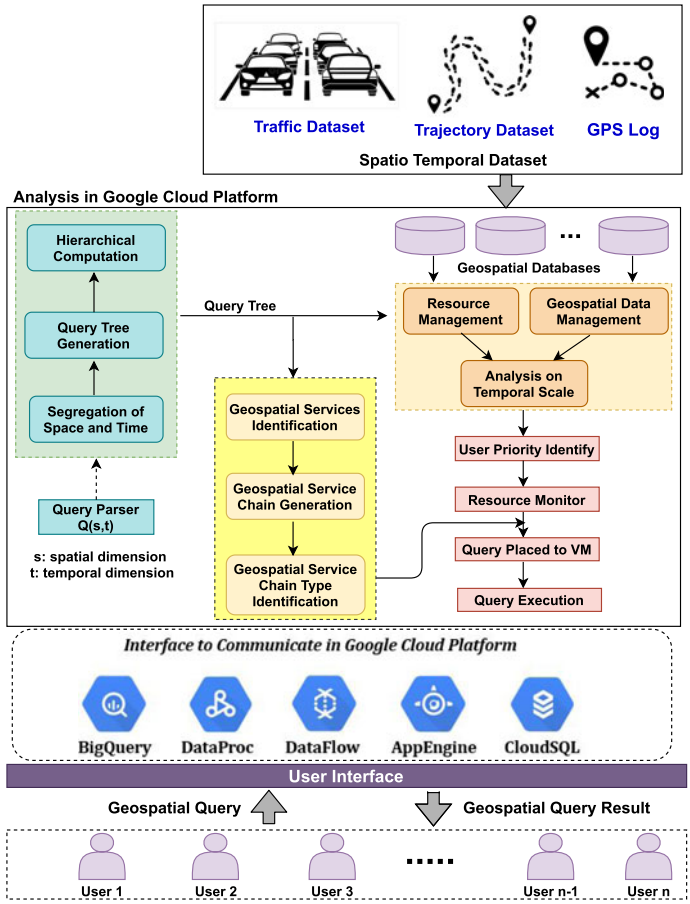

Fig. 2. Block diagram of LYRIC framework.

- *Single Clause Query* These types of spatio-temporal queries consist of one clause. It considers only one table for extracting the result from the database.

  Example: `Select <A> from Table <B> where <C>;`

  Here, $A = \{A_1, A_2, \ldots A_n\}$ is name of features, $B = \{B_1, B_2, \ldots B_n\}$ is the set of relations or tables, and $C = \{C_1, C_2, \ldots C_n\}$ is set of clause.

- *Nested Loop Query* These types of queries are associated with multiple clauses, which require either joining two or more relations or iteratively executing them. In general, the Cartesian product of the multiple tables are involved with these queries.

  Example: `Select <A> from Table <B> where <C> <conditions> (Select <A1> from Table <B1> where <C1>);`

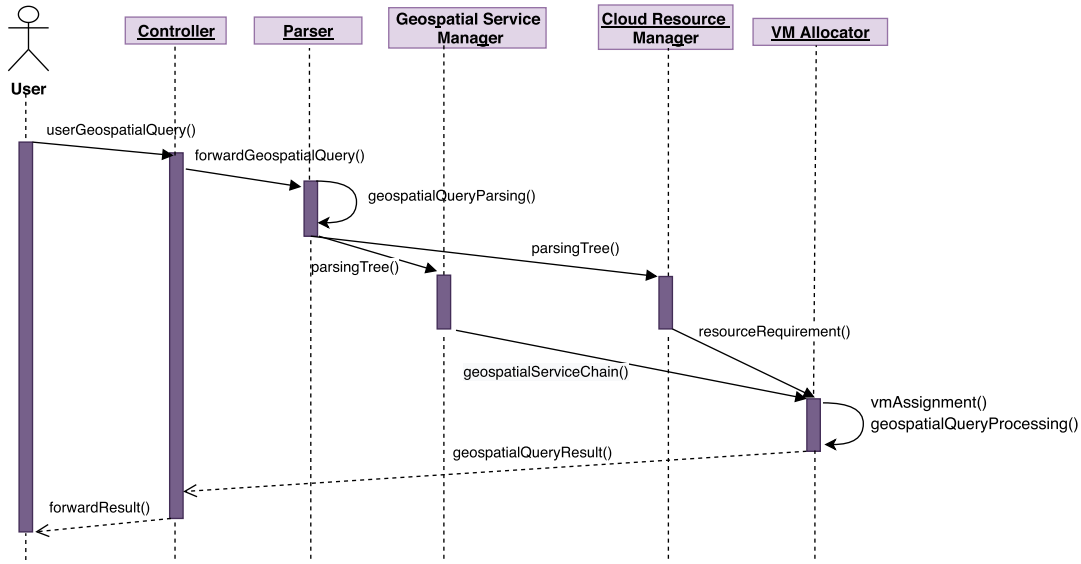

Fig. 3. Sequence diagram of overall architecture.

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

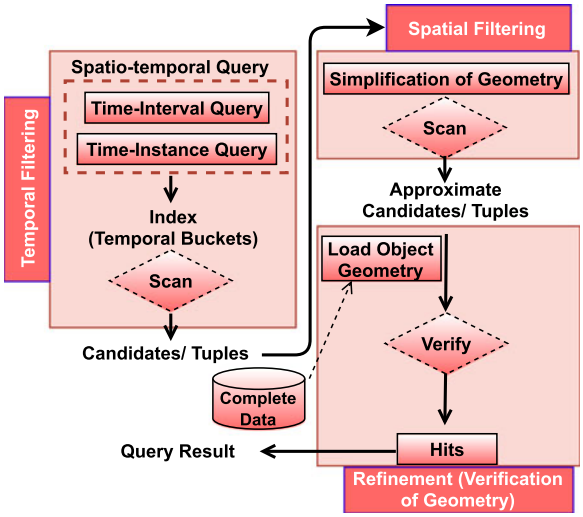

Fig. 4. Spatio-temporal query processing stages.

Next, the spatio-temporal query parse tree has been generated from the spatio-temporal query. It may be noted that spatio-temporal query processing is both CPU and I/O intensive, and we need to minimize the number of disk accesses to reduce the I/O cost. We have adapted *spatio-temporal indexing* which helps to cluster the temporal information and store the spatial objects efficiently such that the number of disk accesses is minimized. Typically, LYRIC solves a spatio-temporal query in three steps (see Fig. 4): (i) temporal filtering, (ii) spatial filtering, and (iii) refinement step. In the temporal filtering, given any of the two types of the query (time-interval and time-instance), we search the temporal index (buckets) and find the appropriate buckets satisfying the temporal clause. The selected candidates/ tuples are returned. Next, the spatial objects (returned tuples) are represented by simple approximation. Here, we have used MBR (minimum bounding region), and in this step, the spatial operations are carried out. In the final step (refinement), the exact geometry information of the approximate candidate set is examined. It may be noted, we estimate the cardinality of each of these stages and combine them to get the exact resource requirement. For instance, the "scan" procedure is I/O intensive (requires disk accesses), and "verify" (or, refinement) is CPU-intensive. For query parse tree generation, we follow the same steps. As depicted in the Fig. 5, for the following query : "Find all movement history having length greater than 1 km and within 50 m of Commercial building and within time-interval 09:00-0.9:15", the parse tree has been generated. For optimization purpose, it may be noted that "refinement" step needs to be pushed down. The selection of building type can be done earlier to reduce the cardinality (see the arrow sign in Fig. 5).

From the nodes of the tree, we can get the required geospatial services. We also get the geospatial service chain if we follow the tree's path from leaf nodes to the root node. The query parse tree generation has two major phases: (i) in the initial phase, the framework processes the *SQL-text*, and identifies the set of A, B, and C. It also identifies whether the query processing needs a nested loop for resolving it. Based on the conventional query parsing technique, it generates the parse tree. (ii) In the next phase, it analyzes the

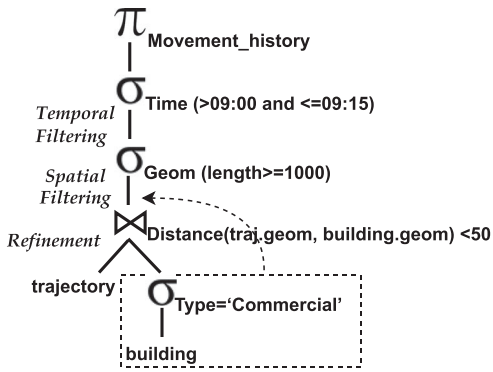

Fig. 5. Spatio-temporal query parse tree generation.

temporal extension of the query. Based on the temporal extension, new levels are generated at the leaf node. For each level, a specific amount of spatial data is processed. This hierarchical segmentation of spatio-temporal data in the query parse tree's leaf node helps to understand the amount of data that needs to be processed at each level of the execution effectively

$$select\ S_{fchr}\ from\ S_{data}\ where\ S_c.$$

- Let S_{fchr} be a collection of feature services available in the cloud in form of Web Feature Service(WFS), denoted as $S_{fchr} = < S_{fchr_1}, S_{fchr_2}, \ldots, S_{fchr_n} >$.
- Let S_{data} be a collection of data services available, denoted as $S_{data} = < S_{data_1}, S_{data_2}, \ldots, S_{data_n} >$.
- S_c is the query predicate which depends on the business logic of orchestration engine and based on the logic different Web Processing Services(WPS) are called and let S_{proc} be a collection of processing services available in the cloud in the form of WPS, denoted as $S_{proc} = < S_{proc_1}, S_{proc_2}, \ldots, S_{proc_n} >$.

To estimate the resource requirement, we have generated a time-based hierarchical parse-tree, where in each level a specific amount of computation is carried out. To estimate the temporal extension, we check the time-interval information in the clause provided in the query-statement. Based on the time-interval information, the whole execution is segmented into several time-buckets. Each of these time-buckets is assigned to each leaf node of the query-parse tree. The above-mentioned example illustrates the applicability of feature service and why it is important to segment a given spatio-temporal query as proposed. It helps to map the execution of a spatio-temporal query into web feature service (WFS), web processing service (WPS), and associate tables of a spatio-temporal database. While it is a generic form of any query-statement, however, this helps to correlate with the later section of the paper where a spatio-temporal query is segmented and different services such as WPS, WFS processes the query segments to reduce the execution time.

3.2 Geospatial Service Chaining

Several OGC compliant geospatial services, i.e., Web Feature Service (WFS), Web Processing Service (WPS), Web Map Service (WMS), are available. The brief descriptions of the geospatial services are given below:

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

- *Web Feature Service*[9]*:* This service allows us to retrieve featured data from stored datasets. The user specifies these features. There are different operations, i.e., GetFeature, GetCapabilities, GetPropertyValue available on WFS.
- *Web Processing Service*[10]*:* This geospatial service allows us to perform different types of spatial operations like buffering, intersection, overlaying on a point, polyline, or polygon. These operations are dependent upon the user's spatio-temporal query.
- *Web Map Service*[11]*:* This service allows us to integrate multiple map layers from one or more distributed spatio-temporal databases and displays one merged map according to the spatio-temporal query. The map images are in JPEG, PNG, TIFF format, which is displayed in a browser application.

These geospatial services are being executed as a sequential operation to generate the final result. Though multiple services are present in this chain, it still appears to be an aggregated one from the query-user's perspective. We categorise the geospatial service chain according to the number of geospatial services, i.e., WFS, WPS, WMS invocation. We represent six types of geospatial service chains in Fig. 6.

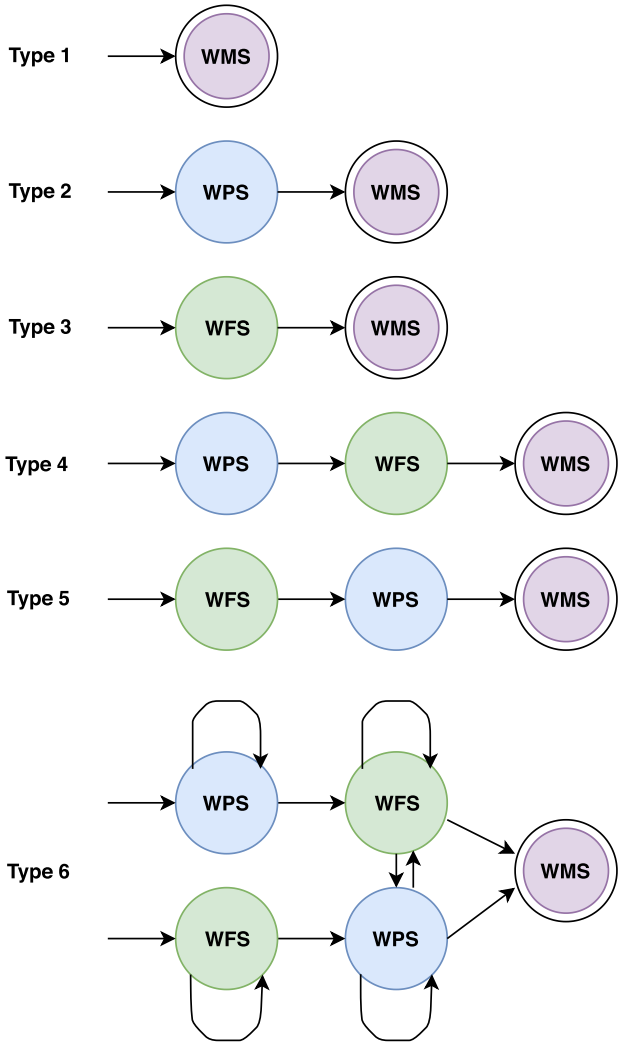

Fig. 6. Different geospatial service chains in a state diagram form.

- *Type 1: Only View* This type of geospatial query is only for visualizing a map. There is no such filtration or specification present here. Only web map service is responsible for this kind of geospatial query. The existing maps are displayed here.
  Considering our motivating example, Land Use Land Cover (LULC), Transportation (Road, Rail), Drainage map of super-cyclone affected areas (here, Area_X) relations are used.
  ```
  SELECT LULC FROM Area_X;
  SELECT Road Network FROM Area_X;
  ```
  The above geospatial queries retrieve data of the individual layers (LULC/ Road Networks) of Area_X and a *getMap* WMS displays these layers individually or combines in any one of the png, jpeg, and tiff format.
- *Type 2: Process and View* These types of geospatial queries are the combination of process and view. Processing is done over already existing maps. One WPS and WMS are responsible for resolving this kind of geospatial query.
  The following example helps to identify the locations of rail and road crossing bridges of Area_X. NDRF monitors these bridges, which are affected by super-cyclone or not, and proceeds accordingly.
  Example: Select crossing points of rail and road of an Area_X
  ```
  SELECT point.geom
  FROM X.rail ra, X.road ro
  WHERE Cross(ra.Shape, ro.Shape)=1;
  ```
  In this example, first, it will take the rail network layer and road network layer of area X. *LineIntersectionService* WPS is used to obtain the intersection points with lat/lon and *getMap* WMS displays the crossing points.
- *Type 3: Filter and View* A particular area or parameter is considered for this type of query. A specific feature of a map is visualized here. One WFS and WMS will resolve this kind of geospatial query.
  The geospatial query identifies the high roads of Area_X from the road network. It helps NDRF to clear the blockage on the high road due to super-cyclone.
  Example: High roads from Road Network
  ```
  SELECT *
  FROM X.Road_Network
  WHERE road_type = 'High_Road';
  ```
  This query filters the high roads from Area_X road network. *getFeature* WFS identifies the high roads and *getMap* WMS displays the roads.
- *Type 4: Filter over the Processed area and View* A WFS, WPS, and WMS together resolve this type of geospatial query. Filtration can be done after the processing of an existing map.
  NDRF team wants to check the conditions of the narrow bridges over rivers after super-cyclone. The following geospatial query helps to identify the same.
  Example: Identify the narrow bridges from all the

9. https://www.ogc.org/standards/wfs
10. https://www.ogc.org/standards/wps
11. https://www.ogc.org/standards/wms

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

intersections of road and water networks.

```
SELECT *
FROM Bridge B
WHERE B.type = 'narrow'
AND B.geom = (
SELECT point.geom
FROM Water_Network W, Road_Network R
WHERE Cross(W.shape, R.shape) = 1));
```

This geospatial query obtains two layers, i.e., a water network and road network, to process it to determine the junction points with lat/lon. It will use *LineIntersectionService* WPS for that. Now, in the bridge layers, it filters the narrow-type bridges with the same lat/lon as junction points by using *getFeature* WFS. Finally, *getMap* WMS displays the narrow bridges in a map.

- *Type 5: Process over Filtered area and View* Here processing is done after the filtration from the existing map. WFS comes before the invocation of WPS. At last, WMS visualizes the resultant map.
  Matla river is situated in the super-cyclone affected area. NDRF team identifies the damage of both the banks of *Matla* river that spread 1 kilometer of each side.
  Example: 1km buffer zones of *Matla* River

```
SELECT *
FROM Water_Network
WHERE W_Net_type = 'River' AND river_name =
'Matla'
AND Buffer(area.shape, 1);
```

  The above geospatial query first, filters the river with name 'Matla' using *getFeature* WFS, and then it creates buffers of 1 km over the river with *BufferFeatureCollection* WPS. Buffer of *Matla* will display as the result by using *getMap* WMS.
- *Type 6: Multiple Filter, Processed area and View*In this type of query, multiple filtration and processing can be done one after another. There should not be a specific chain of WFS and WPS. The sequence can be any combination of WFS and WPS.
  Suppose NDRF team starts rescue operations according to the population of the area. Densely populated areas, which are near to the highway, gets the highest priority and so on.
  Example: Finding the fifty towns which have above one thousand population nearest to the national highway NH36.

```
SELECT t.name, t.population, sdo_nn_dis-
tance
FROM interstates i, town t
WHERE i.highway_name = 'NH36'
AND sdo_nn(t.location, i.geom) = 'TRUE'
AND t.population > 1000
AND rownum < 51
ORDER BY sdo_nn_distance;
```

  The above geospatial query has many WFS like population counts, specific highway names. WPS helps to determine the nearest towns to the highway NH36 by using the Nearest Neighbour algorithm.

The process of determining the types of the geospatial service chain has been mentioned in Algorithm 1. The execution of the service-chain depends on the number and variety of geospatial services.

Algorithm 1. Determine the Types of Geospatial Query From Parse Tree

Input: Geospatial query parse tree
Output: Type of the geospatial query

```
start
the geospatial parse tree is generated from geospatial query
identify the leaf nodes of the tree, which are data nodes
the spatial operation is held on the parent node of the leaf
nodes
identify the spatial operations WFS, WPS, WMS
if only WMS required then
    Geospatial Query Type 1
else
    if first WFS and then WMS required then
      Geospatial Query Type 2
    else
      if first WPS and then WMS required then
        Geospatial Query Type 3
      else
        if first WPS, then WFS, lastly WMS required then
          Geospatial Query Type 4
        else
          if first WFS, then WPS, lastly WMS required
then
            Geospatial Query Type 5
          else
            multiple time WFS, WPS, and WMS required
            Geospatial Query Type 6
          end if
        end if
      end if
    end if
end if
end
```

3.3 Cost Model of Spatio-Temporal Queries

Query Plan (QP). Query plan consists of the segments $(S_a, S_b, \ldots, S_z)$ of the execution of the query along with a probable time-deadline of each such segment.

$QP_q := \{S_a[t_a, f_a] \ldots S_z[t_z, f_z]\}$, where query plan ($QP$) of the query $q(T)$ is given, where T is the user-deadline. t_a and f_a is start and finish timestamp of segment S_a. t_z and f_z is start and finish timestamp of segment S_z. For each execution segment, $[t_a, f_a]$ is given, and f_z-$t_a \leq T$. After determining the cost-effective execution strategy, a query plan is produced, and the execution is carried out based on this QP.

For instance, an user u_i submits a spatio-temporal query (Q) with a deadline t_i at time-instance T_a. Further, the user also provides a budget (Bt) for resolving the query. For efficiently resolving the query considering the user-defined timeline and budget, the framework needs to compute the probable execution time. We have also predicted the execution time of bulk queries submitted by the users. A trade-off is required to resolve the queries within the user-deadline at minimum cost. It also helps in the capacity planning of cloud VMs, i.e., whether the VMs should be upgraded or downgraded based on the workloads. The main objective here is to *predict resource usages, i.e., memory, CPU usages, and*

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Comparisons With Existing Works and LYRIC

Feature	Related Works				LYRIC
	[5], [6]	[13]	[17], [18], [19]	[20]	[Proposed Work]
Geospatial query execution	✓	✓	✘	✓	✓
Spatial service chain consideration	✘	✘	✓	✓	✓
Query placement to VM	✘	✘	✘	✘	✓
Priority-basis query resolution	✘	✘	✘	✘	✓
Apriori estimation of resource requirements	✘	✓	✘	✘	✓

disk accesses. Also, LYRIC analyzes the query's probable run-time and user-deadline. Based on the predicted resource usages, LYRIC computes the cost and compares it with the user's budget. Based on these two factors, it produces an appropriate query plan and configures the VMs.

The cost model of our framework follows four steps: (i) estimation of input and output cardinality; (ii) computing the CPU cost based on the cardinality estimation; (iii) computing the I/O cost based on the estimated number of accessed pages; and finally (iv) combining CPU cost, I/O cost along with WMS (map service cost for visualizing the results on map). We discuss the cardinality estimation procedure (including analyzing temporal extension) in the next section. The query parse tree and sequential-sampling approach help to get the estimation of cardinality, and the indexing and storage method is utilized to estimate the I/O cost.

3.4 Cost Estimation and Prediction of Run-Time of Spatio-Temporal Queries

In the context of the spatio-temporal query, we define our problem space into two broad aspects: (i) static spatial objects (these are "fixed location assets", such as buildings, road-segments, lakes, mountains, etc.) where location information does not change with time; and (ii) moving objects in the two-dimensional space (moving agents, say, trajectory traces of people, vehicles, etc.). It may be noted, that temporal information is crucial in the latter case, since the location changes with time-instances, along with the data size. For the spatial query, the cost is determined by the cardinality of the relational tables. And for the spatio-temporal queries, we take the temporal extension and multiply it with the time required to process one unit of spatio-temporal operations. Let us explain the cost incurred for resolving spatio-temporal queries and how our framework, LYRIC, predicts the queries' execution time. Table 2 shows the notations used in the cost model. The cost of a query (Q) can be written as

$$Cost(Q) = cs \times np + cr \times nr + cc \times nt + ci \times nti + co \times nc. \tag{1}$$

We have used these five parameters of PostgreSQL's model in our cost model. It may be noted that an accurate query time predictor requires an accurate estimation of these variables. The CPU operations mean the spatial operations such as *buffer*, *intersection* etc. along with common SQL operations (count, aggregate etc.). We consider the following query as an example:

TABLE 2
Notations Used in Cost Model

Notation	Meaning
cs	I/O cost to access a page sequentially
cr	I/O cost to access a page randomly
cc	CPU processing cost of a tuple
ci	CPU processing cost of a tuple (using index)
co	CPU processing cost to carry out an operation
np	Number of sequentially scanned pages
nr	Number of randomly accessed pages
nt	Number of tuples processed/ accessed
nti	Number of tuples processed/ accessed (Indexing)
nc	Number of CPU operations
qt	Query execution time

Q_a:`SELECT count(*) FROM Road_Network WHERE road_type = 'High Road' AND road_id = 'N' AND Buffer(area.shape, 1);`

Here, the relation *Road_Network* is memory resident, and two CPU operations, $count(*)$ and BUFFER() are present. Two conditions need to be satisfied. Suppose road_id has clustered index, and road_type is an attribute with a non-clustered index. Therefore, the query plan consists all five parameters ($np_{Q_a}, nr_{Q_a}, nt_{Q_a}, nti_{Q_a}, nc_{Q_a}$). In general, we generate such query plans where varied combinations of cost variables are required and solve the following equation:

$$qt = NC, \tag{2}$$

where N is the cardinality of tuples or operations, and C is the CPU or I/O cost. By solving the system of linear equations, we can find out the accurate value of C. It may be noted that PostgreSQL uses the default values for I/O and CPU cost, which does not capture the real-life computational cost.

In order to estimate the run-time, two important facts need to be considered: (i) calibration of cost units and (ii) estimation of input and output cardinality of the query. Calibration of cost units means to get the exact value of cost (say, CPU operation time) to execute a single unit of task. It may be noted that PostgreSQL uses default values for this purpose, however those are not accurate and it definitely varies with the hardware and software of the VMs where query will be executed. Here, we come up with the fundamental idea to design particular query template such that it isolates specific cost parameters from others. As we presented six types of geospatial service chain in Section 3.2, those types help to calibrate the unit costs. For example: *SELECT LULC FROM Area_X*. In this (type 1: Only View) query template, there is no I/O cost (we assume, Area_X is memory resident), and only cpu_tuple_cost and WMS cost (getMap) are involved. We execute the query without the WMS service and take the execution time (say, t_1). Next, we call WMS service, and note the execution time (say, t_2). Therefore, the WMS cost is $t_2 - t_1$. In our framework, we divide the study area (geographical extent) in uniform grids and temporal extent in buckets of 15 minutes. Say, the query processes n_g grids, then

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

$$\begin{aligned} t_1 &= T_g \times n_g \\ t_2 &= n_g \times (T_g + WMS_g), \end{aligned} \tag{3}$$

where WMS_g is the time taken to call getMap service and visualize for one grid, and T_g is the time for CPU operation for one grid. From these equations, we get the values for T_g and WMS_g. In the similar way, we utilize all six types of service chains with different query sets to get the CPU cost and geospatial service cost of different spatio-temporal service chains. Also, to make the procedure more robust, we use multiple queries for each of the service chain template, and get the best-fitting of the costs. This is one-time phenomenon for any computing platform or VMs, therefore, the time taken to carry out this task does not effect the overall time-complexity of LYRIC.

Next, we need to estimate the cardinality of the input and output tuples of the query plan. Here, LYRIC utilizes a variant of *sequential-sampling* method[26]. We follow the similar steps of [5] for each WPS and WFS of the query plan. It makes the estimator more efficient and suitable for spatio-temporal queries. After this cardinality estimation and refinement stage, LYRIC effectively predicts the query execution time (qt).

For estimating the cost, we segregate into: start-cost: initial cost (c_1) before operator produce the first output tuple; and total-cost (c_2) : the total cost when all output tuples are generated. Thus, the execution cost can be represented as $ec = c_2 - c_1$. Let us illustrate with two broad categories of queries as depicted in Section 3.1, i.e, *single clause query* and *nested loop query*. For example, if single clause query aims to select an object's location in a particular time-instance, then

$$\begin{aligned} c_1 &= 2 \times c_o \times in_t \times \log in_t \\ ec &= c_t \times in_t, \end{aligned} \tag{4}$$

where in_t is the number of input tuples for the operator. Here, we have discretized the temporal information into buckets, and those temporal buckets are in sorted order. Hence, we can deploy any sorting algorithm for the data-instances, therefore $\log$ factor is present in the equation. Next, for the nested loop query

$$\begin{aligned} c_1 &= c_1 \; of \; inner \; clause + c_1 \; of \; outer \; clause \\ ec &= c_t \times in_t^{inner} \times in_t^{outer} + in_t^{outer} \times (ec \; of \; inner \; clause), \end{aligned} \tag{5}$$

where the number of input tuples from inner and outer clauses are presented by in_t^{inner} and in_t^{outer} respectively.

Let us illustrate, the cardinality estimator process here. Prior to that, let us define two terms, namely, *rank (μ)* and *selectivity (ω)*

$$\begin{aligned} \mu &= \frac{\omega - 1}{per \; tuple \; cost \; of \; a \; spatial \; operation} \\ \omega(q) &= \frac{cardinality(output(q))}{cardinality(input(q))}. \end{aligned} \tag{6}$$

Here, the *per tuple cost of spatial operation* is computed by the calibration of cost units as described before. While creating the query parse tree, we put the predicates or services based on the ascending order of μ. Thus, we optimize the query processing in terms of operations. The calculation of cardinality of input is straightforward which is the product of the cardinalities of the spatio-temporal relations (tables) that are input to the operator. We obtain this information from the metadata or system catalogue.

Say, in the spatio-temporal database SD, there are M relations or tables $\{ST_1, ST_2, \ldots, ST_M\}$. Each of the relations (say, ST_i) are partitioned into p_i blocks, and each block stores g spatial grids and bu temporal buckets. Therefore, $|ST_i| = p_i \times (g, bu)$. Now, consider two operators: *select (σ)* and *cross-product ($\times$)*. The selectivity of the operations are as follows:

$$\begin{aligned} \omega_{\sigma(ST)} &= \frac{|\sigma_{clause}(g, bu)|}{|(g, bu)|} \\ \omega_{\times(ST)} &= 1. \end{aligned} \tag{7}$$

The pair (g, bu) is the spatial and temporal extents of the input relation. Now, we can say, if a relation ST_i is segmented into p grids, then $\omega_{p_i} = \frac{|\sigma_{clause}(p_i)|}{|p_i|}$ $(1 \leq i \leq n)$, then $E[\omega_{p_i}] = \omega_{ST}$, where, we have taken n random samples of grids from ST relation. In this way, when we get the value of *selectivity*, it is straightforward to get the output cardinality.

Now, we present the estimation process of output cardinality for different spatio-temporal queries. For *spatial intersection query*, let us assume $P_i(O_1)$ is the probability that object O_1 intersects the grid G_i. Again, the probability of intersecting exactly p grids is: $P(O_1, p)$. Then, the expected number of spatial grids that O_1 will intersect is given as (g is the total number of grids present in a particular relation/table)

$$\begin{aligned} E(O_1) &= \sum_{p=0}^{g} p \times P(O_1, p) \\ &= P_i(O_1). \end{aligned} \tag{8}$$

Now, for point query, an object is likely to be present at any point of the grid. Hence, we get

$$\begin{aligned} E(O_1) &= area \; of \; the \; grids \; present \; in \; the \; relation \\ &= \sum_{i=1}^{g} a_1 \times a_2, \end{aligned} \tag{9}$$

where a_1 and a_2 are the length of the sides of the grids. For range-query $O_1(a_1 \times a_2)$, we have

$$E(O_1) = \textstyle\sum_{i=1}^{height-1} g_i \times \prod_{j=1}^{2} (n\_ext_{i,j} + a_j), \tag{10}$$

where $height$ is the height of the spatial indexing tree (we have used R* tree [27]), and the average node extent in j dimension and i level is $n\_ext_{i,j}$. Thus, following these approaches, we estimate the cost and predict the run-time of the spatio-temporal queries.

3.5 Query Plan Generation Using Game Theory

At this stage, we have the predicted query execution time (qt) and user-deadline (T) along with the budget (Bt) of the query. LYRIC's objective is to provide a query execution plan such that the total cost is minimized and $qt \leq T$. To obtain such a query execution plan, we deployed a

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

cooperative game theory, where joint actions taken by the group of players provide collective payoffs. In general, $P(p_1, p_2, \ldots p_n)$ number of players are present in cooperative game theory, and for each subset of players, a vector of payoff (R) is defined. The tuple (P, r) is defined as a characteristic function. In our problem set-up, we define varied services of the query plan, such as, *WFS-controller*, *WPS-controller*, and *WMS-controller* as group of players. Each of the players may choose a strategy $s = \{s_1, s_2, \ldots s_j\}$ to adapt such that the aggregate payoff is maximized. In other words, all the players cooperate in the game, taking varied strategies such that the outcome of the game is the agreement condition from all the players. When we consider two influencing factors, *budget* (Bt) and user-deadline (T) of the queries, the *bargaining* method can resolve the problem.

As discussed, there are P players participating in the game, where players are the spatial-services required for the query. Each non-empty set of P is termed as a coalition. For each coalition (say, A), we denote a set $R(A) \in \mathbb{R}^{|A|}$ - which is the payoff vector and feasible for coalition A. The collection (Co) of the coalition is denoted to be balanced if the following statement is satisfied

$$\sum_{A \in Co,\, j \subseteq A} we(A) = 1 \quad when \;\; we(A) \in [0,1] \;\; for\ each\ j \in P, \tag{11}$$

where $we(A)$ is the set of weights for each $A \in Co$. We have considered (P, R) as a transferable utility case of game theory in our approach. Since one of the services (or player) can losslessly transfer its utility to another service (or player). Typically, for each $A \subseteq P$, there exists a real number $r(A)$

$$R(A) = \{g \in \mathbb{R}^{|A|} : \sum_{i \in A} g_i \leq r(A)\}, \tag{12}$$

where g is the set of vectors. In our problem, each of the strategy consists of $(resource, executiontime)$. The set of payoffs define the set of allocations of games (P, r)

$$G(P, r) = \{g \in \mathbb{R}^n : \sum_{i \in P} g_i = g(P) \leq r(P)\}. \tag{13}$$

Thus, the core of the game is deployed using the following equations:

$$Core(P, r) = \{g \in G(P, r) : \forall A \subseteq P, g(A) \geq r(A)\}. \tag{14}$$

Furthermore, it is already well-known that if the game is balanced, the core is non-empty. The bargaining solution is the function of

$$f(U_g) : \sum^{|P|} \rightarrow U_g, \tag{15}$$

where $\sum^{|P|}$ is the class of all bargaining problems. Based on the cooperative game theory, there is a unique solution $s(U_g)$ of bargaining problem. It can be achieved by

$$s(U_g) = \arg\max_{y \in U_g} (y_1 - d_1) \times (y_2 - d_2) \times \ldots (y_p - d_p), \tag{16}$$

where d is the disagreement point. After each step, the payoff is the percentage of the completion of the task, and the utility (U_g) of the game is defined as

$$U_g = \begin{cases} |T - qt| \times |Bt - acost| & if((T > qt)\ and \\ (Bt > acost)) & \\ (-1) \times |T - qt| \times |Bt - acost| & otherwise \end{cases}, \tag{17}$$

where $acost$ is the actual cost of the query execution. The user-deadline, budget, and query execution time are represented by T, Bt, and qt respectively. It is obvious that maximizing the utility function both benefits in terms of budget and reduces the response time.

After resolving the budget and deadline trade-off using the game theory approach, we deploy a query scheduling module to assign the query to appropriate VMs and the time-stamp. The basic steps of the process are represented in Algorithm 2.

Algorithm 2. Scheduling Algorithm of Spatio-Temporal Query Placement to Virtual Machine

Input: Spatio-temporal query
Output: VM allocated to spatio-temporal query
1: start
2: Receive spatio-temporal query along with response deadline from user
3: Search and select available VM by the VM manager according to the type of the spatio-temporal query
4: Calculate weights for each available VM
5: Sort VMs in increasing weights
6: Assign VM to a spatio-temporal query according to the weights of the VM
7: If the requirement of the spatio-temporal query is not satisfied, go to step 6
8: If the requirement of the spatio-temporal query is not satisfied, and VMs are not available. Notification send to VM manager
9: Receives the suspended VM list from the VM manager. If there is no suspended VMs, the assignment is a failure. Go to step 11
10: Add suspended VMs to the available VM list. Go to the step 4
11: Send the assignment result to the VM manager
12: end

3.6 Example Scenario

This section presents a sample query and corresponding service chains and execution process. *Geospatial Query:Determine the housing complex of Area A with more than 10 acres situated within 500 meters of state highway R*

```
SELECT area_name FROM Area_A WHERE area ≥ 10
AND area_type='Housing_Complex' AND road =
'State_Highway' AND road_name = 'R' AND Overlap
(road.shape, Buffer(area.shape, 0.5)) ORDER BY
area desc;
```

To resolve the above geospatial query, the following geospatial services are required. (1) WFS getFeature service for *area ≥ 10*WFS getFeature service for *area_type = 'Housing_Complex'*(2) WPS BufferFeatureCollection service for *Buffer (area.shape, 0.5)*(3) WFS getFeature service for *road = 'State_Highway'*(4) WFS getFeature service for *road name = 'R'*(5)

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

WPS IntersectionFeatureCollection service for *Overlap* operation (6) WFS getFeature service for *list preparation*(7) WMS getMap service for *display* final result. Hence, from the above lists of geospatial services, we can get the following geospatial service chain, which belongs to Geospatial Chaining Type 6 (refer Section 3.2).

WFS → WFS → WPS → WFS → WFS → WPS → WFS → WMS In the next step, we generate the query parser tree, where the leaf nodes store the spatial and temporal extents of the query. We also already know the cost units of several services and CPU and I/O cost from calibration module. Then, we compute the cardinality estimation and refinement from the process described in Section 3.4. Now, given the user deadline and budget, LYRIC forms the pay-off matrix with players: five WFS, 2 WPS, and 1 WMS. All these players participate in the cooperative game and find out the coalition, where each of them gets a specific amount of time for execution and budget or resources to utilize. Finally, the game objective is to minimize the difference between (deadline, total execution time of all players) and (budget allocated, total resource consumption of all players). Based on the game coalition, the query plan is generated and executed.

4 Performance Evaluation

To illustrate the efficacy of the proposed architecture, we have designed and executed a large set of experiments on mobility datasets. The intuition behind taking the mobility dataset is that the movement data is dynamic and accumulates on a large scale. Furthermore, most of our real-life spatio-temporal queries are associated with the movement, traffic, and road datasets. Therefore, we consider the use case of mobility-related queries. However, the framework is generic to handle any types of spatio-temporal datasets and resolve queries efficiently.

4.1 Experimental Test-Bed

We have used real-life mobility traces of three geographical regions, *Kharagpur* (22.346010, 87.231972), *Durgapur* (23.5204, 87.3119), and *Waranangle* (17.9689, 79.5941) in India. The study area of these regions are $12.6\,km^2$, $5.23\,km^2$, and $4.08\,km^2$ respectively. The number of participants in this study is 204, 42, and 76 from three regions, respectively, for a timespan of 12 months. We developed a web interface to extract their movement path and utilized the *Google Map Timeline*, *Google Map Services* to extract the underlying road networks. It was observed that the road networks of these three regions have more than 50,000 edges, and the cardinality of the road network makes the information extraction and resolving mobility queries more challenging factor. On the other side, the *reverse geocoding* technique was used to extract the point-of-interests (POIs) (such as commercial places like shopping malls, banks, markets, or academic and residential areas, etc.) from the map database. The mobility traces are collected in $60\,secs$ to $180\,secs$ time interval. The total size of the dataset is $64\,GB$, $36\,GB$, and $49\,GB$, respectively.

The experiment is conducted in the VMs of *Google Cloud Platform*, where we have used several computational and storage features of Google Cloud. Our test-bed consists of several types of compute engine instances. For instance, we have created 5 general-purpose VM instances (Ubuntu-16.04 LTS) ranging from $4\,vCPU$, and $15\,GB\,memory$ to $32\,vCPUs$, and $120\,GB\,memory$. Each such instance's approximate cost is $\$0.134$ per hour to $\$1.065$ per hour. These VMs are used for common workloads and having low cost and more flexibility. Along with these general-purpose VMs, we have also created 2 memory-optimized and 3 compute-optimized instances, which are used for memory and compute-intensive workloads. The initial configuration that we have selected is $40\,vCPUs$, $961\,GB\,memory$, and $30\,vCPUs$, $120\,GB\,memory$ respectively. The approximate cost is $\$1.253$ per hour. Next, we create an instance group with these compute engine instances for auto-scaling and load-balancing based on the user requirement and predefined budget. We have also deployed spatial tools and databases, namely, *QGIS*, and *PostgreSQL with PostGIS extension*. To store and manage, spatio-temporal data-instances, we need 'geometry' or 'geom' [1] property, which illustrates the type (say, road: line geometry; building: polygon geometry etc.) of the data instance. We have used $Postgresql\ 12.6\ (Postgis\ 3.0)$ database for storing and managing the dataset. To store and retrieve spatio-temporal data effectively, we have utilized a novel spatio-temporal storage structure [28] for indexing temporal information and R* tree for storing and managing spatial objects. We have adapted the K-level spatio-temporal hashing technique for storing the spatio-temporal datasets. Here, the data instances are segmented and stored in different temporal buckets, and the storage technique is implemented using a hashing scheme that considers the spatial-proximity feature (nearby locations are stored in nearby buckets). Therefore, in the initial level, the spatial features of the data instances are stored, and from the next level, the data is stored into different temporal buckets.

4.2 Performance Results

It is quite obvious that spatial query resolution requires analyzing a large number of spatial data[29]. Moreover, the performance is dependent on I/O and computational efficacy. To consider these, we have used both I/O and spatial query metrics. The I/O performance is measured by sequential and random reads along with bulk loading. The efficacy of spatial query resolution is measured by *r-query (range)*, *p-query (point)* and *t-query (trajectory)*. The range or r-query ($RangeQ(\mathcal{S}, \mathcal{T})$) finds all trajectory segments which intersects the given spatial ($\mathcal{S}$) and temporal ($\mathcal{T}$) extent

$$RangeQ(\mathcal{S}, \mathcal{T}) \rightarrow Traj,$$

where $Traj$ is the set of trajectory segments within spatial($\mathcal{S}$) and temporal($\mathcal{T}$) extent. The t-Query or trajectory-based query finds all trajectory segments of a moving agent (a) within the time interval ($\mathcal{T}$)

$$TrajectoryQ(a, \mathcal{T}) \rightarrow Traj.$$

Here, $Traj$ is the output trajectory of the query. We have used both the r-query and t-query in our evaluation set-up.

From Fig. 7, the accuracy based on the specific deadline is shown with a varied number of concurrent queries ranging from 500 to 10000. We have compared the accuracy of the result with four well-known baselines, *SparkGIS*, *geoSpark*, *GeoMesa*[12] and *JUST* [30]. The accuracy is computed based

12. http://www.geomesa.org/

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

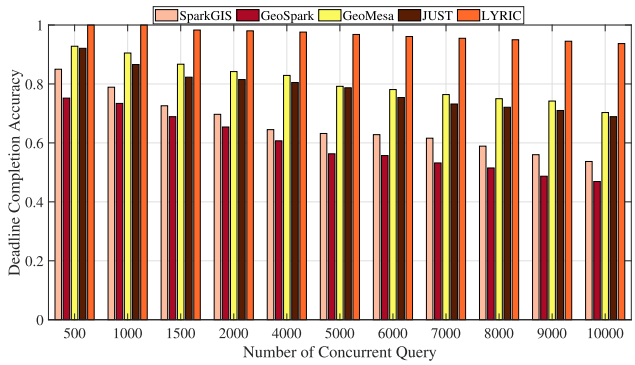

Fig. 7. Task completion accuracy within deadline.

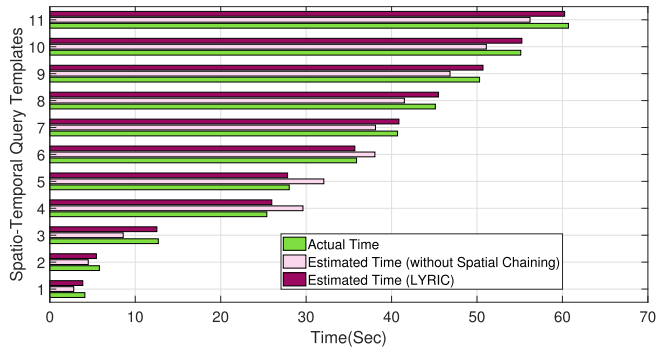

Fig. 9. Prediction of query execution time.

on the percentage of completion of the task in the user-deadline. It is observed that with more numbers of concurrent queries, LYRIC, significantly outperforms than others. It shows high deadline completion accuracy in the range of $1.0 - 0.937$. Whereas, in the same set-up, SparkGIS, GeoSpark, GeoMesa and JUST provide $0.85 - 0.537$, $0.752 - 0.469$, $0.928 - 0.703$, $0.921 - 0.689$ respectively. Fig. 8 illustrates the memory footprints for concurrent query processing compared to the other four popular methods. The memory footprints denote the amount of main memory segments accessed or referred for the query processing during execution. It is observed that the savings of memory footprints are significantly better than the existing methods. The key reason behind this result is that LYRIC computes the resource requirements apriori and assigns the required resources effectively. It also reduces the percentage of under-utilization of resources accordingly.

We evaluate the prediction accuracy of the execution time of the spatio-temporal queries. To depict the efficacy, we experiment in two set-ups: (i) prediction module without geospatial service chain and (ii) considering the geospatial service chain. The latter method shows better accuracy in Fig. 9.

The reason is that the geospatial chain is one of the integral parts of providing the query result to the end-users. For instance, a few segments of query processing can be carried out in parallel. In contrast, few segments depend on the previous one, thus require sequential processing. Since LYRIC explores and identifies such geospatial service chains automatically and predicts the execution time, it achieves a more accurate result. It is observed that the prediction module without the geospatial service chain provides 14.50 percent error (differences in actual execution time and prediction time), while LYRIC achieves only 2.68 percent prediction error. Therefore, LYRIC can reduce the prediction error by 11 percent incorporating the geospatial service chaining method.

To demonstrate the effectiveness of LYRIC, we have evaluated our proposed framework using three cost metrics, namely, *Web-Feature Service Cost*, *CPU Cost* and *Web Map Service Cost*. The *CPU cost* is categorized in two classes, namely, cost related to static or spatial query operations and cost related to spatio-temporal query tasks. It is observed from Fig. 10 that there are marginal differences between *Web-Feature Service Cost* and *Web Map Service Cost*, however, *CPU cost* is significantly less in LYRIC compared to other methods. The major reason is that LYRIC efficiently generates the query plan and subsequently executes it for better execution time and less budget to complete the tasks. In this work, we have not used any new method for WMS or WFS cost reduction, except that we have augmented spatio-temporal indexing method [28] for less feature extraction delay. Therefore, the WFS cost and WMS cost are marginally different from other baselines, however, LYRIC outperforms in a significant margin in CPU cost, thus, facilitating better query execution time and budget requirements.

To evaluate the framework's effectiveness, we have used the CloudSim [31] toolkit, where we simulated the same environment as in GCP. We simulate the query arrival scenario. As discussed, we define a set of six types of spatio-temporal queries in the list. For each such type, we initially write 120 queries manually, each having a spatial and temporal variable. From the list of such 120 queries, we generate 120×10^5 queries in the list varying the spatial and temporal domain. It may be noted that in the simulation process, we provide a bounding box for the spatial variable. Otherwise,

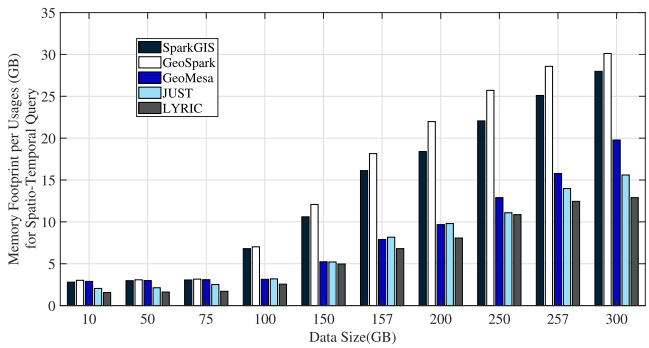

Fig. 8. Comparison of memory footprints for concurrent query processing.

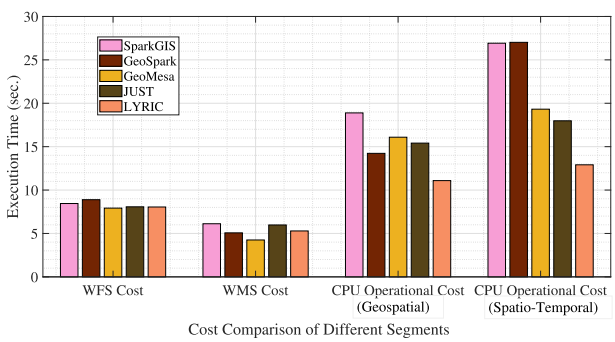

Fig. 10. Cost comparisons with baselines.

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

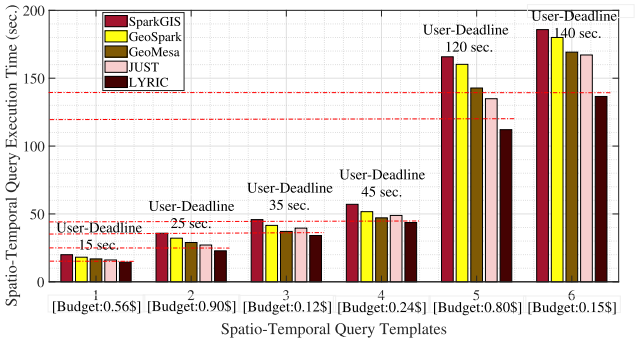

Fig. 11. Comparison of actual execution time and deadlines.

any random pick of spatial variables may lead to a region outside our datasets' study-region. For each query, there are two other parameters, *user-deadline (T)* and *budget (Bt)*. The query arrival rate is determined by the well-known *Gaussian distribution*. To evaluate LYRIC performance under varied workloads, we experiment with different arrival rates.

In the experimental set-up, we have considered several samples of deadlines and budgets (which are set-up based on real-life knowledge of a GIS domain expert). Figs. 11 and 12 illustrate the specific values of the deadline, allocated budget, and actual execution time and budget required to resolve the task. We have shown results for six spatio-temporal query templates and queries are randomly selected using CloudSim toolkit. In Fig. 11, in the experimental set-up, strict budget allocation estimation and the deadline (query task completion time) have been provided. It has been observed that for most of the cases (baselines), query execution time has exceeded the deadline provided. Again, in Fig. 12, we have specified that the deadline should be strictly maintained, and it is observed that the allocated budget has been overshoot in a significant margin for the baselines. The key reason is that for all other distributed platforms for spatial and spatio-temporal query processing, no method/algorithm is present to optimize both deadline and budget allocation. Moreover, the baseline methods do not consider the service chain and provision concurrent processing based on service chain execution in the query processing. However, in our case, LYRIC identifies the service chains, generates the query plan, and concurrently processes the query using a game theory approach to satisfy both deadline and budget optimally.

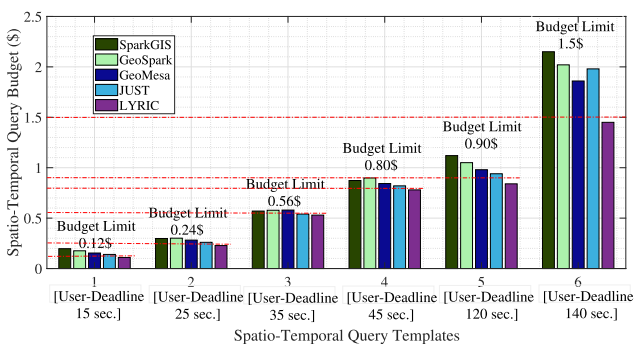

Fig. 12. Comparison of budget requirements with baselines.

5 Conclusion and Future Work

Accurate estimation of query execution time, and the requirement of the computational resources, are challenging tasks. It helps in query scheduling based on the user-deadline and budget of the query processing. Furthermore, we can monitor the progress of the query processing. For bulk query processing, a particular query taking an unreasonably long time can be identified and eliminated apriori. It helps in system sizing or obtaining the approximate estimation of the total budget or resource utilization. The proposed framework, LYRIC, has three main components. First, it models the cost of an incoming spatio-temporal query based on the known PostgreSQL's cost model. However, instead of the default parameters used by PostgreSQL, LYRIC extracts the accurate CPU and disk-access cost and effectively predicts the query execution time. Next, it identifies several spatio-temporal services required to complete the query processing task and further decomposes it into a query tree. LYRIC is capable of considering the user-defined timeline and given budget for each query. The framework utilizes the concept of cooperative game theory to obtain the trade-off between more resources and budget or cost. LYRIC is deployed in GCP, and real-life experiments with mobility datasets and simulations yield encouraging results.

As part of the future work, we will extend the framework in a multi-cloud environment where several cloud service providers take part in a game with their geospatial services. The user can select geospatial services and cloud resources based on their geospatial service requirements and budget. We will also utilize advanced machine learning techniques to append more features of spatio-temporal datasets to enhance the accuracy of the prediction. We believe that LYRIC will act as a foundation for the deadline and budget-aware spatio-temporal query resolution framework.

Acknowledgments

This work was supported in part by Research Grant from the Department of Science and Technology, Government of India under Geospatial Chair Professor Award. Jaydeep Das and Shreya Ghosh contributed equally to this paper.

References

[1] S. Shekhar and S. Chawla, *Spatial Databases: A Tour*. Upper Saddle River, NJ, USA: Prentice Hall, 2003.

[2] V. Cardellini, V. Di Valerio, and F. L. Presti, "Game-theoretic resource pricing and provisioning strategies in cloud systems," *IEEE Trans. Services Comput.*, vol. 13, no. 1, pp. 86–98, Jan./Feb. 2020.

[3] G. Yao, Q. Ren, X. Li, S. Zhao, and R. Ruiz, "A hybrid fault-tolerant scheduling for deadline-constrained tasks in cloud systems," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2020.2992928.

[4] R. Marcus, S. Semenova, and O. Papaemmanouil, "A learning-based service for cost and performance management of cloud databases," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 1361–1362.

[5] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigümüs, and J. F. Naughton, "Predicting query execution time: Are optimizer cost models really unusable?," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 1081–1092.

[6] W. Wu, Y. Chi, H. Hacígümüş, and J. F. Naughton, "Towards predicting query execution time for concurrent and dynamic database workloads," *Proc. VLDB Endowment*, vol. 6, no. 10, pp. 925–936, 2013.

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.

[7] A. Ganapathi *et al.*, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 592–603.
[8] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik, "Learning-based query performance modeling and prediction," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 390–401.
[9] S. Ramesh, A. Baranawal, and Y. Simmhan, "A distributed path query engine for temporal property graphs," in *Proc. 20th IEEE/ACM Int. Symp. Cluster Cloud Internet Comput.*, 2020, pp. 499–508.
[10] X. Zhou, J. Sun, G. Li, and J. Feng, "Query performance prediction for concurrent queries using graph embedding," *Proc. VLDB Endowment*, vol. 13, no. 9, pp. 1416–1428, 2020.
[11] Z. Chu, J. Yu, and A. Hamdulla, "A novel deep learning method for query task execution time prediction in graph database," *Future Gener. Comput. Syst.*, vol. 112, pp. 534–548, 2020.
[12] J. Duggan, O. Papaemmanouil, U. Cetintemel, and E. Upfal, "Contender: A resource modeling approach for concurrent query performance prediction," in *Proc. 17th Int. Conf. Extending Database Technol.*, 2014, pp. 109–120.
[13] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala, "Modeling and exploiting query interactions in database systems," in *Proc. 17th ACM Conf. Inf. Knowl. Manage.*, 2008, pp. 183–192.
[14] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala, "QShuffler: Getting the query mix right," in *Proc. IEEE 24th Int. Conf. Data Eng.*, 2008, pp. 1415–1417.
[15] M. Ahmad, A. Aboulnaga, and S. Babu, "Query interactions in database workloads," in *Proc. 2nd Int. Workshop Testing Database Syst.*, 2009, Art. no. 11.
[16] A. D. Popescu, V. Ercegovac, A. Balmin, M. Branco, and A. Ailamaki, "Same queries, different data: Can we predict runtime performance?," in *Proc. IEEE 28th Int. Conf. Data Eng. Workshops*, 2012, pp. 275–280.
[17] P. Yue, L. Di, W. Yang, G. Yu, and P. Zhao, "Semantics-based automatic composition of geospatial web service chains," *Comput. Geosci.*, vol. 33, no. 5, pp. 649–665, 2007.
[18] P. Zhao, L. Di, G. Yu, P. Yue, Y. Wei, and W. Yang, "Semantic Web-based geospatial knowledge transformation," *Comput. Geosci.*, vol. 35, no. 4, pp. 798–808, 2009.
[19] L. Di, P. Zhao, W. Yang, and P. Yue, "Ontology-driven automatic geospatial processing modeling based on web-service chaining," in *Proc. 6th Annu. NASA Earth Sci. Technol. Conf.*, 2006, pp. 27–29.
[20] P. Yue, J. Gong, and L. Di, "Augmenting geospatial data provenance through metadata tracking in geospatial service chaining," *Comput. Geosci.*, vol. 36, no. 3, pp. 270–281, 2010.
[21] J. Das, A. Dasgupta, S. K. Ghosh, and R. Buyya, "A geospatial orchestration framework on cloud for processing user queries," in *Proc. Int. Conf. Cloud Comput. Emerg. Markets*, 2016, pp. 1–8.
[22] V. W. Chu, R. K. Wong, S. Fong, and C.-H. Chi, "Emerging service orchestration discovery and monitoring," *IEEE Trans. Services Comput.*, vol. 10, no. 6, pp. 889–901, Nov./Dec. 2017.
[23] X. Tan *et al.* "Cloud and agent-based geospatial service chain: A case study of submerged crops analysis during flooding of the yangtze river basin," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 3, pp. 1359–1370, Mar. 2015.
[24] J. Das, A. Dasgupta, S. K. Ghosh, and R. Buyya, "A learning technique for VM allocation to resolve geospatial queries," in *Recent Findings in Intelligent Computing Techniques*, vol. 1. Berlin, Germany: Springer, 2019, pp. 577–584.
[25] C. Jatoth, G. R. Gangadharan, and R. Buyya, "Computational intelligence based QoS-aware web service composition: A systematic literature review," *IEEE Trans. Services Comput.*, vol. 10, no. 3, pp. 475–492, May/Jun. 2017.
[26] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami, "Selectivity and cost estimation for joins based on random sampling," *J. Comput. Syst. Sci.*, vol. 52, no. 3, pp. 550–569, 1996.
[27] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1990, pp. 322–331.
[28] S. Ghosh, S. K. Ghosh, and R. Buyya, "MARIO: A spatio-temporal data mining framework on Google cloud to explore mobility dynamics from taxi trajectories," *J. Netw. Comput. Appl.*, vol. 164, 2020, Art. no. 102692.
[29] K. Lee *et al.*, "Lightweight indexing and querying services for big spatial data," *IEEE Trans. Services Comput.*, vol. 12, no. 3, pp. 343–355, May/Jun. 2019.
[30] R. Li *et al.*, "JUST: JD urban spatio-temporal data engine," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1558–1569.
[31] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.

Jaydeep Das received the BTech degree from the West Bengal University of Technology, Kolkata, India, and the ME degree from Jadavpur University, Kolkata, India. He is currently working towards the PhD degree within the Advanced Technology Development Centre, Indian Institute of Technology Kharagpur, Kharagpur, India. His research interests include geospatial cloud computing, geospatial information system, and geospatial query resolution in the cloud, fog, edge computing environments.

Shreya Ghosh (Student Member, IEEE) received the BTech degree from the Department of Computer Science and Technology, Indian Institute of Engineering Science and Technology, Shibpur (IIEST Shibpur), India, in 2015. She is currently working toward the PhD degree with IIT Kharagpur, Kharagpur, India. She is currently a research scholar with the Department of Computer Science and Engineering, IIT Kharagpur, India. Her current research interests include spatial informatics, trajectory data mining, and cloud computing. She is the recipient of the prestigious TCS fellowship.

Soumya K. Ghosh (Senior Member, IEEE) received the MTech and PhD degrees in computer science and engineering from the Indian Institute of Technology (IIT) Kharagpur, Kharagpur, India. He is currently a professor with the Department of Computer Science and Engineering, IIT Kharagpur. He was with the Indian Space Research Organization, Bengaluru, India. He has authored or coauthored more than 300 research papers in reputed journals and conference proceedings. His current research interests include spatial data science, spatial web services, and cloud computing. He is the recipient of National Geospatial chair professor award from the Department of Science and Technology, Government of India.

Rajkumar Buyya (Fellow, IEEE) is a Redmond Barry distinguished professor and the director of the Cloud Computing and Distributed Systems Laboratory (CLOUDS) Laboratory, University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in cloud computing. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=145, g-index=319, and more than 1,11,300 citations). Recently, he is recognized as a "Web of Science Highly Cited researcher" for five consecutive years since 2016 by Thomson Reuters, and Scopus researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier for his outstanding contributions to cloud computing.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: University of Melbourne. Downloaded on October 07,2022 at 04:45:12 UTC from IEEE Xplore. Restrictions apply.