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, improved networking, and location acquisition techniques, a huge amount of spatio-temporal traces are being accumulated. These dataset facilitates varied location-aware services and helps in deciding real-life scenarios. The analysis and extraction of meaningful information from these massive volumes of the spatio-temporal dataset is a challenging task. Efficiently handling and processing those queries is necessary to respond in real-time. Processing the vast geospatial 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 geospatial-service chaining. In this work, we propose a framework, namely, *LYRIC* (deadLine and budget aware spatio-temporal querY pRocessing In Cloud) where the geospatial queries are resolved efficiently considering user-defined deadline and budget constraint. We perform experiments with real-life datasets in Google Cloud Platform(GCP) and CloudSim simulator for illustrating the efficacy of our proposed system. Our framework shows high deadline completion accuracy in the range of 1.0 - 0.937, which is more accurate than the SparkGIS and GeoSpark framework. It also reduces the resource prediction error by 11% incorporating the geospatial service chaining method than without it.

Index Terms—Spatio-Temporal Query, Geospatial Service, Cloud Computing, Query Budget Constraint, User Deadline

1 Introduction

The huge volume of spatio-temporal data-instances has motivated the data science community to analyse and utilize the underneath knowledge. Spatio-temporal dataset 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 a smart-living such as route-planning, trip-recommendation, weather prediction, etc. However, managing this huge volume of data and obtaining optimized query performance is inevitably challenging tasks. There are several challenges in spatio-temporal query processing. Firstly, 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 datasets. Moreover, Cloud paradigm is suitable to leverage the *pay as-you-go* model based on the resources used in query processing.

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

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-deadline and budget. When an organization or an individual user submits a task (say, 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: Pricing of Google Cloud Platform services[1]) incurred for utilizing the compute, storage 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-interval. 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 services (processing, feature, or map service) to respond; each of those has 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[2] and map services and the price of each of the services (deployed in the cloud) is $\$2$ in $10\,mins$ timespan. However, it is observed that the task can not be completed within the deadline utilizing the present configurations of the services. In such 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, selecting an appropriate query plan considering both deadline and budget is difficult when several spatial services are required to resolve the query.

1. https://cloud.google.com/pricing/list
2. https://www.ogc.org/standards/wfs

To address the issue, LYRIC implements *cooperative game theory* where the objective is to complete the task within the deadline and reducing the budget. In other words, using the minimal resources [2] for the query processing in cloud servers within the user-deadline.

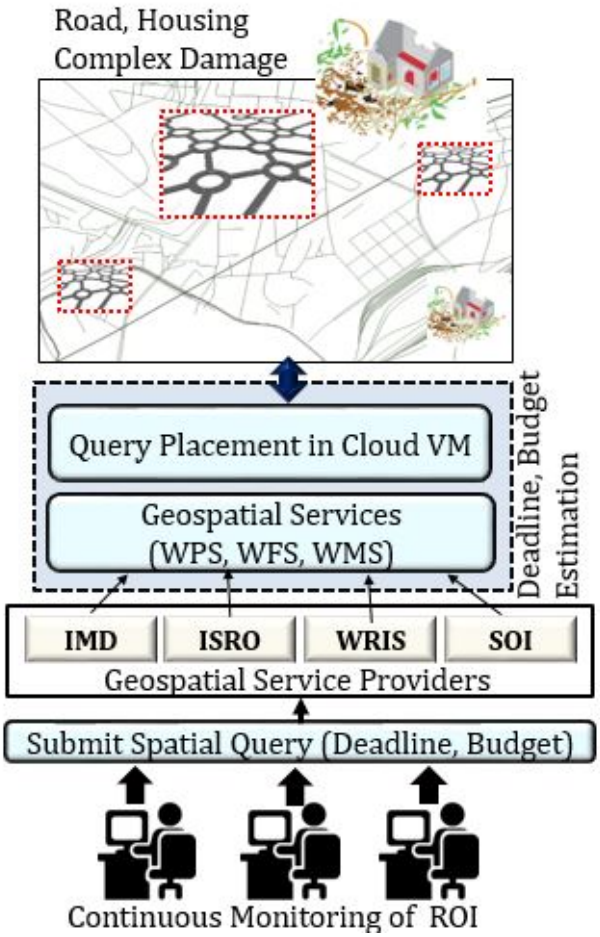

Fig. 1. Motivating scenario

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 situation, several departments (such as electricity, communication, railway, highway, transportation, water resources, etc.) of *National Disaster Response Force (NDRF)* need to work seamlessly to continuously monitor the situation and taking appropriate steps to get back the normalcy. It is obvious that extracting spatio-temporal information from various sources need proper query execution framework and orchestration of geospatial services. For instance, national agencies of Indian government, such as GSI (Geological Survey of India[4]), India-WRIS (Water Resources Information System[5]), SOI (Survey of India[6]), IMD (Indian Meteorological Department[7]), or ISRO (Indian Space Research Organisation[8]) provide varied real-time feature services, map services to retrieve the present situation of the affected region. LYRIC provides a query execution plan considering the user's deadline and budget and resolves the query efficiently utilizing the 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 type of spatio-temporal query processing task with user budget and deadline constraints [3].

Our Contributions:

The key contributions of this paper are as follows:

1) We propose an end-to-end framework, named *LYRIC* to resolve geospatial query within the deadline and budget provided by the user. The framework analyses 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 service chains* required 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 provides 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 energy consumption, delay in the query response, and cost in terms of memory and CPU usages.

The rest of this paper is organized as follows. Section 2 discusses on the related existing strategies. Section 3 discusses the system model of our work, where we discussed geospatial query types, geospatial service chaining, and its six categories. 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

In this section, we discuss the existing works in Marcus et al. [11] used supervised learning techniques for batch processing and reinforcement learning techniques for online processing of user queries. Through this learning, they achieved a query scheduling with proper cost and performance management, which met the service level agreement(SLA) of user and service provider. Many researchers have also

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/

TABLE 1
Comparisons with existing works and LYRIC

Feature	Related Works [4], [5]	[6]	[7], [8], [10]	[9]	**LYRIC** [Proposed Work]
Geospatial query execution	✓	✓	✘	✓	✓
Spatial service chain consideration	✘	✘	✓	✓	✓
Query placement to VM	✘	✘	✘	✘	✓
Priority-basis query resolution	✘	✘	✘	✘	✓
Apriori estimation of resource requirements	✘	✓	✘	✘	✓

analysed the performance and latency of analytical queries through machine learning techniques [4], [5], [12], [13].

A graph-based temporal relationship between entities like edge, vertices, properties has been proposed in [14]. Their approach is for path queries over dynamic temporal graphs. They used Granite distributed engine over Graphite ICM platform for experiments. Another graph embedded query performance prediction for concurrent queries has been proposed by [15]. They also used the graph update and compaction algorithm to determine the query workload. Chu et al. [16] predicts the query execution time using LSTM in graph database. Encoding the query plan tree, they used a post-order traversal algorithm. RF and PCA help to do feature engineering.

A resource modeling approach to measuring concurrent query performance is proposed by Duggan et al. [17], and prediction under concurrency is made in [6], [18], [19]. 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. [20] proposed to predict the runtime performance for a set of queries with a different dataset. They segmented the queries and measured the performance using the machine learning model. Later they tried to predict the overall query runtime. They considered only tuple size and cardinality of the different dataset for estimating execution time.

Geospatial semantics and service-oriented architecture (SOA) based automatic compositions of geospatial services have been made in [7]. DataType, ServiceType, and Association type ontologies had been used as a semantic schema in SOA. They used geospatial services for ontology design, composition building, and semantic analysis. Geospatial services are used to knowledge transformation [8]–[10] 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 are 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 has been done in [25].

Although, there are several research works in this domain, but all of these existing literature has 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. Firstly, 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

In this paper, we have 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 identify the resource (RAM, CPU cores, storage) requirement for processing geospatial queries and predicting the execution time. Our 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 analyses all of these factors and offers a query execution plan resolving the task within the deadline incurring minimal budget. Geospatial queries are places into Cloud VM, and finally, the results of queries are sent back to the user through the dashboard/ 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 geospatial query. The query tree nodes determine the types and number of geospatial services are required. After identifying the geospatial services, LYRIC generates the service chain. Here, we have provided different types of queries based on whether it requires sequential processing of the services or parallel service processing. After generating the service-chain for the particular query, LYRIC provides the query-execution plan based on the users' priority. This geospatial chain formation is one of the key modules that help allocate the virtual machine to resolve the query effectively.

We also determine the resources like RAM, CPU cores, storage requirement for a geospatial 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 geospatial query within the user-defined deadline and budget. We have shown the sequence diagram of the overall activity in fig 3.

3.1 Geospatial Query Types

Geospatial query type identification is essential to estimate the resource(RAM, CPU cores, storage) requirement for the

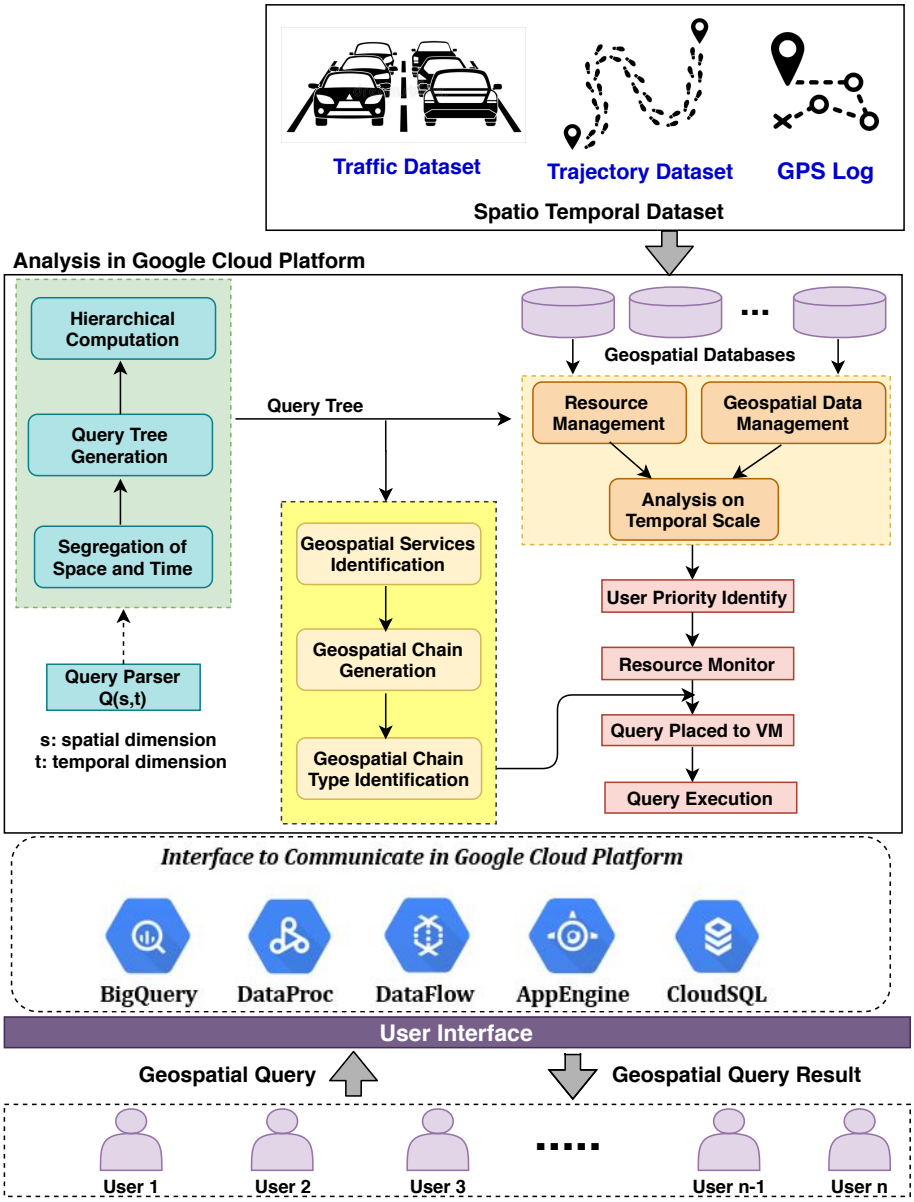

Fig. 2. Block diagram of LYRIC framework

geospatial query or batch of queries. We need to know the amount of geospatial data that has to be processed to resolve the geospatial query. The geospatial data amount depends upon the number of tables selected for the geospatial query. According to the number of table selection, we categorize the geospatial queries into the following two types.

- *Single Clause Query* These types of geospatial queries are with one clause. It considers only one table for extracting the result from the database.
  Example: `Select <A> from Table <B> where <C>;`
- *Nested Loop Query* These types of queries are associated with multiple clauses. 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>;`

A geospatial query tree can be generated from a geospatial query. 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.

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

- Let S_{fchr} be a collection of feature services available in the cloud in form of WFS, denoted as $S_{fchr} =< S_{fchr_1}, S_{fchr_2}, \cdots, S_{fchr_n} >$.
- Let S_{data} be a collection of data services available, denoted as
  $S_{data} =< S_{data_1}, S_{data_2}, \cdots, S_{data_n} >$.
- S_c is the query predicate which depends on the business logic of orchestration engine and based on the logic different WPS services are called and let

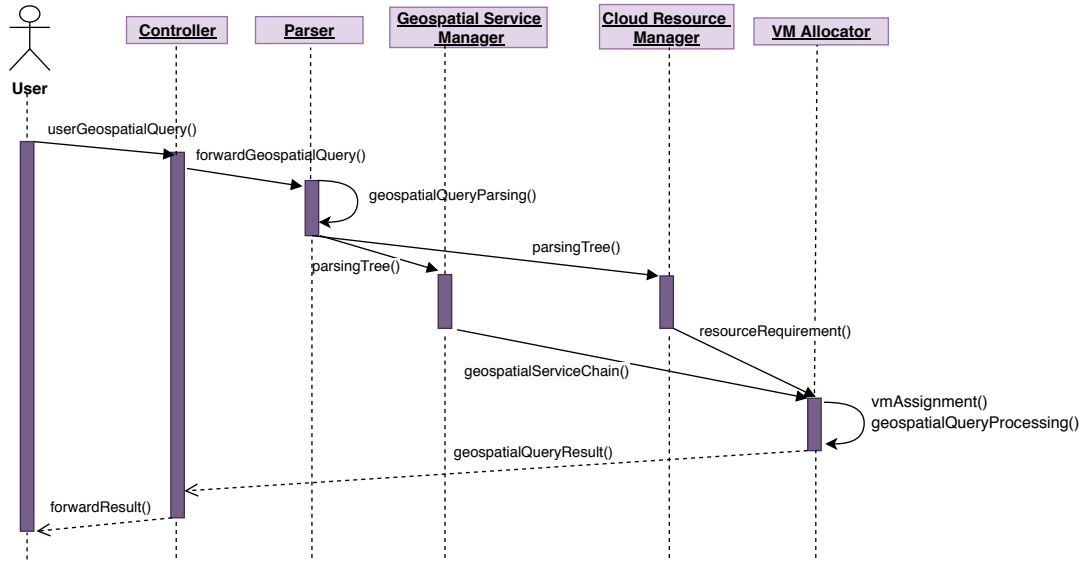

Fig. 3. Sequence diagram of overall architecture

S_{proc} be a collection of processing services available in the cloud in the form of WPS, denoted as $S_{proc} = < S_{proc_1}, S_{proc_2}, \cdots, S_{proc_n} >$.

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 description of the geospatial services are given below:

- *Web Feature Service*[9]*:* This service allows to retrieve featured data from stored data. 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 geospatial operations like buffering, intersection, overlaying on a point, polyline or polygon. These operations are depending upon the user's geospatial query.
- *Web Map Service*[11]*:* This service allows us to integrate multiple map layers from one or more distributed geospatial databases and displays one merged map according to the geospatial query of a user. The map images are in JPEG, PNG, TIFF format, which is displayed in a browser application.

These geospatial services are executing sequential operation to generate the final result. Though multiple services are made in this chain, it still appears to be aggregated one chain to the query-user. We categories the geospatial service chain according to the number of geospatial services, i.e., WFS, WPS, WMS, involvement. We represent the six types of geospatial service chains in Fig.4.

- *Type 1: Only View* This type of geospatial queries is only for visualizing a map. There is no such filtration

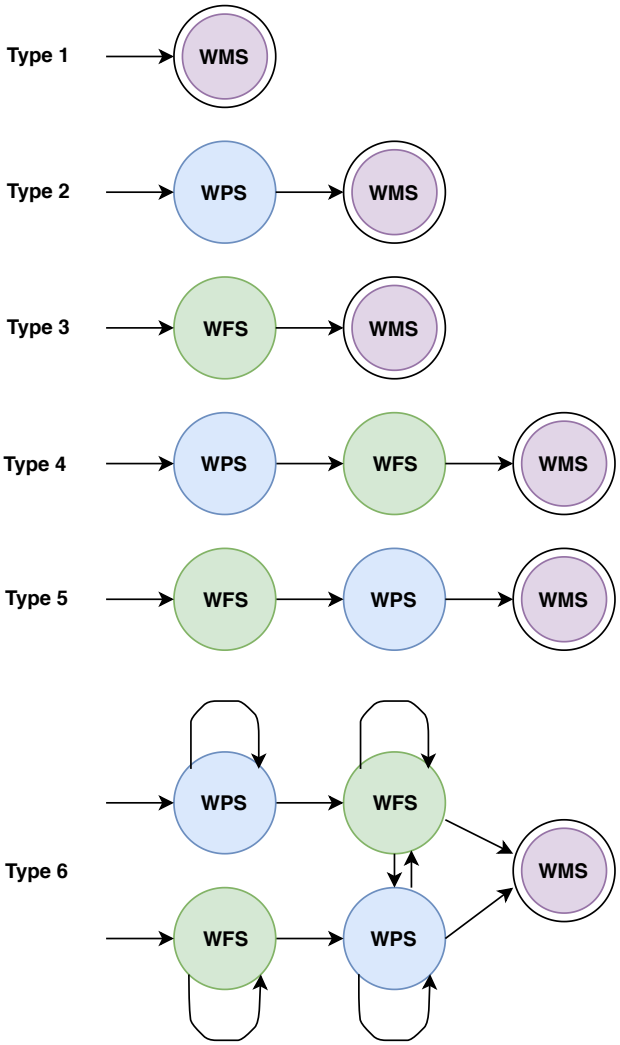

Fig. 4. State diagram of geospatial services for geospatial queries

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

or specification present here. Only web map service is responsible for this kind of geospatial queries. Already 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)

```
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 service display these layers individually or combine 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 web processing service and web map service are responsible for resolving this kind of geospatial queries.
  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 proceed 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* processing service is used to obtains the intersection points with lat/lon and *getMap* service 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 web feature service and web map service will resolve this kind of geospatial queries.
  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* web feature service identify the high roads and *getMap* web map service display the roads.
- *Type 4: Filter over the Processed area and View* A web feature service, web processing service, and web map service together resolve this type of geospatial queries. 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 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* processing service for that. Now, in the bridge layers, it filters the narrow-type bridges with the same lat/lon as junction points by using *getFeature* web feature service. Finally, *getMap* map service 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. Web feature service comes before the web processing service. At last, web map service visualize the resultant map.
  Matla river is situated in the super-cyclone affected area. NDRF teams identify 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, filter the rivers with name 'Matla' using *getFeature* service, and then it creates buffers of 1 km over filtered river with *BufferFeatureCollection* processing service. Buffer of *Matla* will display as a result by using *getMap* map service.
- *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 web feature service and web processing service. The sequence can be any combination of feature and processing service.
  Suppose NDRF teams start 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_distance
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 feature services like population counts, specific highway names. Processing service helps to determine the nearest towns to the highway NH36 by using the Nearest Neighbour algorithm.

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 service required **then**
    Geospatial Query Type 1
else
    if first WFS and then WMS service required **then**
        Geospatial Query Type 2
    else
        if first WPS and then WMS service required **then**
            Geospatial Query Type 3
        else
            if first WPS, then WFS, lastly WMS service required **then**
                Geospatial Query Type 4
            else
                if first WFS, then WPS, lastly WMS service required **then**
                    Geospatial Query Type 5
                else
                    multiple time WFS, WPS, and WMS service 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_q): Query plan consists of the segments of the execution of the query along with a probable time-deadline of each such segment. $QP_q := \{S_a[t_a, f_a] \dots S_b[t_b, f_b]\}$, where query plan ($QP$) of the query $q(T)$ is given, where T is the user-defined deadline. For each execution-segment, a time deadline (t_a, f_a) is given, and $f_b - t_a \leq T$. After determining the cost-effective execution strategy, a query plan is produced, and the execution is carried out based on this QP.

For instance, say, an user u_i submits a ST-query (Q) with a deadline t_i at time-instance T_a. Further, the user also provides a budget or price (Pr) for resolving the query. For efficiently resolving the query considering the user's 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. An efficient trade-off technique is required to resolve the queries within the timeline and 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: memory, CPU usages, and disk accesses. Also, LYRIC analyses the query's probable run-time and the deadline given by the user.* Based on the predicted resource usages, LYRIC computes the cost and compares it with the user-defined budget threshold. Based on these two factors, it produces an appropriate query-plan and configures the VMs.

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

Let us explain the cost incurred for resolving spatio-temporal queries and how our framework, LYRIC, predicts the execution time of the queries. Table 2 shows the notations used in the cost-model. The cost of a query (Q) can be written as:

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

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: 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. Say, road_id has clustered index, and road_type is an attribute with an unclustered 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:

$$qt = NC \quad (2)$$

where N is the cardinality of tuples/ 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.

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. We follow the similar steps of [4] for each WPS and WFS services of the query-plan. It makes the estimator more efficient and suitable for spatio-temporal queries. After this cardinality estimation

and refinement stage, LYRIC predicts the query execution (qt) time effectively.

At this stage, we have the predicted query-execution time (qt) and user-deadline (T) along with the budget (Pr) 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 cooperative game theory, where joint actions taken by the group of players provide collective payoffs. In general, $P(p_1, p_2, ...p_n)$ number of players are present in cooperative game theory, and for each subset of players, a vector of payoff (Pv) is defined. The tuple (P, Pv) 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* or *WMS-controller* as group of players. Each of the players may choose a strategy $s = \{s_1, s_2, \dots\}$ 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, namely, *budget* (Pr) and user-defined timeline (T) of the queries, the *bargaining* method can resolve the problem. Typically, we try to find out the optimal point of the utility function as defined:

$$var(P) = \{g \in \mathbb{R}^{|S|} : \sum_{i \in S} g_i \leq var(S)\} \tag{3}$$

where S is the list of various strategies taken by the players. In our problem, each of the strategy consists of $(resource, execution - time)$. Thus, the core of the game is deployed using the following equations:

$$G(P, Pv) = \{g \in \mathbb{R}^n : \sum_{i \in P} g_i = g(P) \leq Pv(P)\} \tag{4}$$

The bargaining solution is the function of:

$$f(U_g) : \sum^{|P|} \rightarrow U_g \tag{5}$$

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; which can be achieved by:

$$s(U_g) = \arg\max_{y \in U_g} (y_1 - d_1) \times (y_2 - d_2) \times \dots (y_p - d_p) \tag{6}$$

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

$$U_g = \begin{cases} |T - qt| \times |Pr - acost| & if((T > qt)\ and \\ & (Pr > acost)) \\ (-1) \times |T - qt| \times |Pr - acost| & otherwise \end{cases} \tag{7}$$

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

Algorithm 2 Scheduling algorithm of geospatial query placement to virtual machine)

Input: Geospatial query
Output: VM allocated to geospatial query

1: start
2: Receive geospatial query along with response deadline from user
3: Search and select available VM by the VM manager according to the type of the geospatial query
4: Calculate weights for each available VM
5: Sort VMs in increasing weights
6: Assign VM to a geospatial query according to the weights of the VM
7: If the requirement of the geospatial query is not satisfied, go to step 6
8: If the requirement of the geospatial 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

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 queries are associated with the movement, traffic, and road datasets. Therefore, we consider the use-case of the 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, namely, *Kharagpur* (22.346010, 87.231972), *Durgapur* (23.5204, 87.3119) and *Waranangle* (17.9689, 79.5941) in India. The study-area of these regions is $12.6\,km^2$, $5.23\,km^2$, and $4.08\,km^2$ respectively. The number of participants who participated in the study is 204, 42, and 76 in the 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 POIs (point-of-interests) (such as commercial places like shopping malls, banks, market, or academic and residential areas, etc.) from the map database. The mobility traces are collected in $60\,secs$-$180\,secs$ 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 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, *PostgreSQL with PostGIS extension* and *QGIS*.

4.2 Performance Results

It is quite obvious that spatial query resolution requires analysing a large amount of spatial data [26]. 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 ($RangeQ(S,T)$) finds all trajectory segments which intersects the given spatial (S) and temporal (T) extent.

$$RangeQ(S,T) \rightarrow Traj$$

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

$$TrajectoryQ(a,T) \rightarrow Traj$$

Here, $Traj$ is the output trajectory of the query.

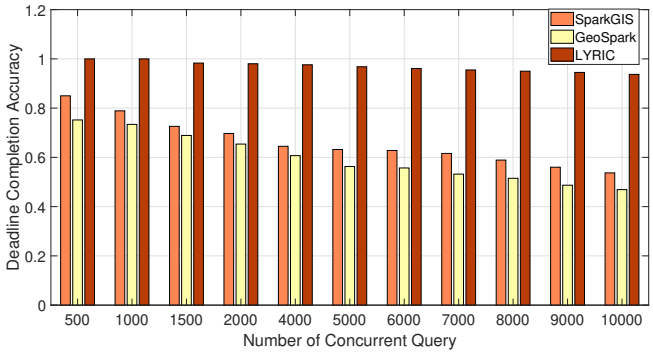

Fig. 5. Task completion accuracy within deadline

As shown in Fig. 5, the accuracy based on the specific deadline is shown with a varied number of concurrent queries ranging from 500 to 10000. We have compared

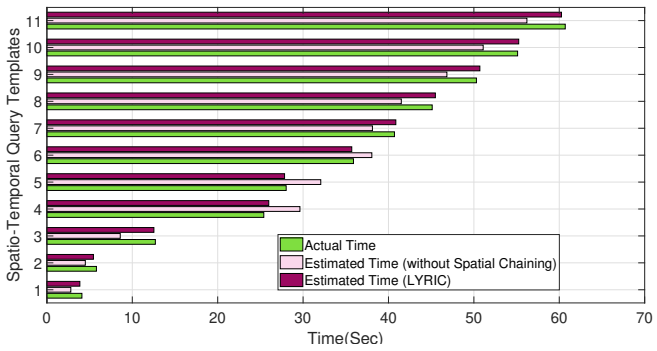

Fig. 6. Prediction of query execution time

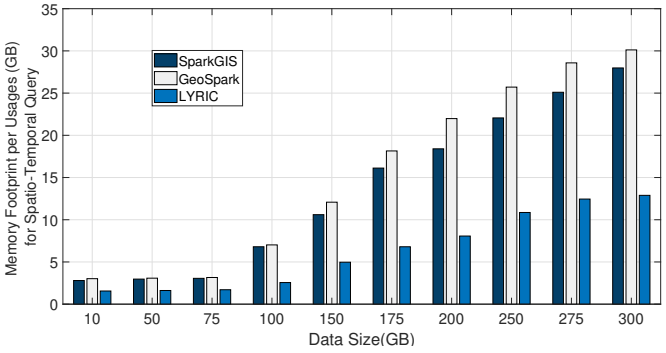

Fig. 7. Comparison of memory footprints for concurrent query processing

the accuracy result with two well-known baselines, namely, *SparkGIS* and *geoSpark*. The accuracy is computed based on the percentage of completion of the task in the user-given deadline. It is observed that with more numbers of concurrent queries, our framework, LYRIC, significantly outperform others. It shows high deadline-completion accuracy in the range of $1.0 - 0.937$. In the same set-up, SparkGIS and GeoSpark provide $0.85 - 0.53$ and $0.752 - 0.469$, respectively. Fig. 7 illustrates the memory footprints for concurrent query processing compared to other two popular methods. The amount of memory footprints refers to the main memory segment the query processing refers to while 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 spatial chaining and (ii) considering the spatial chain. As illustrated in the Fig. 6, the later method shows better accuracy. The reason is that spatial chain is one of the integral parts of providing the query result to the end-users; for instance, few segments of the 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 chain automatically and predicts the execution time, it achieves a more accurate result. It is observed that the pre-

diction module without spatial chaining provides 14.50% error (differences in actual execution time and prediction time), while LYRIC achieves only 2.68% prediction error. Therefore, LYRIC can reduce the prediction error by 11% incorporating the geospatial service chaining method.

To evaluate the framework's effectiveness, we have used the CloudSim [27] toolkit, where we simulated the same environment as in GCP. We simulate the query (or job) 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 20 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, any random pick of spatial variables may lead to a region outside the study-region of our datasets. For each query, there are two other parameters, *user-defined time (t)* and *budget (c)*. The query arrival rate is determined by well-known *Gaussian distribution*. To evaluate LYRIC performance under varied workloads, we experiment with different arrival rates.

5 Conclusions and Future Work

Accurate estimation of query-execution time, as well as computational resources, is a challenging task. It helps in query-scheduling based on the deadline and budget of the query processing. Furthermore, we can monitor the progress of the queries, and for bulk query-processing, particular queries taking unreasonably long time can be identified and eliminated apriori. Also, it helps in system sizing or obtaining the approximate estimation of total budget or resource utilization. Our 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 the 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 in a query-tree. LYRIC is capable of considering the user-defined timeline and given budget for each query. The framework uses a variant 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 as well as simulations yield encouraging results.

In future, we aim to extend the framework to incorporate in a multi-cloud environment where several cloud service providers can take part, and the user can select based on their requirements and pricing. 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 of deadline and budget-aware spatio-temporal query resolution framework, and more specific applications can be deployed over it.

References

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

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

[4] 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 *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 1081–1092.

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

[6] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala, "Modeling and exploiting query interactions in database systems," in *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM, 2008, pp. 183–192.

[7] P. Yue, L. Di, W. Yang, G. Yu, and P. Zhao, "Semantics-based automatic composition of geospatial web service chains," *Computers & Geosciences*, vol. 33, no. 5, pp. 649–665, 2007.

[8] P. Zhao, L. Di, G. Yu, P. Yue, Y. Wei, and W. Yang, "Semantic web-based geospatial knowledge transformation," *Computers & Geosciences*, vol. 35, no. 4, pp. 798–808, 2009.

[9] P. Yue, J. Gong, and L. Di, "Augmenting geospatial data provenance through metadata tracking in geospatial service chaining," *Computers & Geosciences*, vol. 36, no. 3, pp. 270–281, 2010.

[10] L. Di, P. Zhao, W. Yang, and P. Yue, "Ontology-driven automatic geospatial processing modeling based on web-service chaining," in *Proceedings of the sixth annual NASA earth science technology conference*. Citeseer, 2006, pp. 27–29.

[11] R. Marcus, S. Semenova, and O. Papaemmanouil, "A learning-based service for cost and performance management of cloud databases," in *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 2017, pp. 1361–1362.

[12] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, 2009, pp. 592–603.

[13] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik, "Learning-based query performance modeling and prediction," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 390–401.

[14] S. Ramesh, A. Baranawal, and Y. Simmhan, "A distributed path query engine for temporal property graphs," in *Proceedings of* 20^{th} *IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*. IEEE, 2020, pp. 499–508.

[15] X. Zhou, J. Sun, G. Li, and J. Feng, "Query performance prediction for concurrent queries using graph embedding," *Proceedings of the VLDB Endowment*, vol. 13, no. 9, 2020.

[16] Z. Chu, J. Yu, and A. Hamdulla, "A novel deep learning method for query task execution time prediction in graph database," *Future Generation Computer Systems*, 2020.

[17] J. Duggan, O. Papaemmanouil, U. Cetintemel, and E. Upfal, "Contender: A resource modeling approach for concurrent query performance prediction." in *EDBT*, 2014, pp. 109–120.

[18] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala, "Qshuffler: Getting the query mix right," in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, 2008, pp. 1415–1417.

[19] M. Ahmad, A. Aboulnaga, and S. Babu, "Query interactions in database workloads," in *Proceedings of the Second International Workshop on Testing Database Systems*. ACM, 2009, p. 11.

[20] A. D. Popescu, V. Ercegovac, A. Balmin, M. Branco, and A. Ailamaki, "Same queries, different data: Can we predict runtime performance?" in *2012 IEEE 28th International Conference on Data Engineering Workshops*. IEEE, 2012, pp. 275–280.

[21] J. Das, A. Dasgupta, S. K. Ghosh, and R. Buyya, "A geospatial orchestration framework on cloud for processing user queries," in *Proceedings of International Conference on Cloud Computing in Emerging Markets (CCEM)*. IEEE, 2016, pp. 1–8.

[22] V. W. Chu, R. K. Wong, S. Fong, and C.-H. Chi, "Emerging service orchestration discovery and monitoring," *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 889–901, 2015.

[23] X. Tan, L. Di, M. Deng, A. Chen, F. Huang, C. Peng, M. Gao, Y. Yao, and Z. Sha, "Cloud and agent-based geospatial service chain: A

case study of submerged crops analysis during flooding of the yangtze river basin," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 3, pp. 1359–1370, 2014.
[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*. Springer, 2019, vol. 1, pp. 577–584.
[25] C. Jatoth, G. Gangadharan, and R. Buyya, "Computational intelligence based qos-aware web service composition: a systematic literature review," *IEEE Transactions on Services Computing*, vol. 10, no. 3, pp. 475–492, 2015.
[26] K. Lee, L. Liu, R. Ganti, M. Srivatsa, Q. Zhang, Y. Zhou, and Q. Wang, "Lightweight indexing and querying services for big spatial data," *IEEE Transactions on Services Computing*, vol. 12, no. 3, pp. 630–643, 2018.
[27] 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," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

Jaydeep Das received the B.Tech degree in information technology from West Bengal University of Technology, Kolkata, India, and the M.E. degree in software engineering from Jadavpur University, Kolkata, India. He is currently working towards the PhD within the Advanced Technology Development Centre, Indian Institute of Technology Kharagpur, India. His broad area of research includes geospatial cloud computing, geospatial information system and geospatial query resolution in mobile cloud environment.

Shreya Ghosh received the B.Tech. degree from IIEST Shibpur, India, in 2015. She is currently a Research Scholar with the Department of Computer Science and Engineering, IIT Kharagpur, India working towards her PhD. Her current research interests include spatial informatics, trajectory data mining and cloud computing. Shreya is the recipient of TCS fellowship.

Soumya K. Ghosh received the M.Tech and Ph.D. degrees in Computer Science and Engineering from Indian Institute of Technology (IIT) Kharagpur, India. He is currently a Professor in the Department of Computer Science and Engineering, IIT Kharagpur. Prior to joining IIT Kharagpur, he worked for Indian Space Research Organization in the area of Satellite Remote Sensing and GIS. His research interests include cloud computing, spatial information retrieval and knowledge discovery. He has published over 200 articles in journals and conference proceedings. He is a member of IEEE.

Rajkumar Buyya is a Redmond Barry distinguished professor and the director of the Cloud Computing and Distributed Systems Laboratory (CLOUDS) Laboratory, University of Melbourne, Australia. He received Ph.D. degree in Computer Science and Software Engineering from Monash University, Melbourne, Australia, in 2002. His research interests are cloud computing, fog computing, big data, and Internet of things. He is a fellow of IEEE and Web of Science highly cited researchers.