symmetry

MDPI

Article

Execution Time Prediction for Cypher Queries in the Neo4j Database Using a Learning Approach

Zhenzhen He [1], Jiong Yu [1,*] and Binglei Guo [2]

[1] School of Information Science and Engineering, Xinjiang University, Urumqi 830049, China; hezhenzhen@stu.xju.edu.cn
[2] School of Computer Engineering, Hubei University of Arts and Science, Xiangyang 441053, China; binglei@hbuas.edu.cn
* Correspondence: yujiong@xju.edu.cn

Abstract: With database management systems becoming complex, predicting the execution time of graph queries before they are executed is one of the challenges for query scheduling, workload management, resource allocation, and progress monitoring. Through the comparison of query performance prediction methods, existing research works have solved such problems in traditional SQL queries, but they cannot be directly applied in Cypher queries on the Neo4j database. Additionally, most query performance prediction methods focus on measuring the relationship between correlation coefficients and retrieval performance. Inspired by machine-learning methods and graph query optimization technologies, we used the RBF neural network as a prediction model to train and predict the execution time of Cypher queries. Meanwhile, the corresponding query pattern features, graph data features, and query plan features were fused together and then used to train our prediction models. Furthermore, we also deployed a monitor node and designed a Cypher query benchmark for the database clusters to obtain the query plan information and native data store. The experimental results of four benchmarks showed that the average mean relative error of the RBF model reached 16.5% in the Northwind dataset, 12% in the FIFA2021 dataset, and 16.25% in the CORD-19 dataset. This experiment proves the effectiveness of our proposed approach on three real-world datasets.

Keywords: Cypher queries; pattern queries; query plan tree; execution time prediction; Neo4j database

Citation: He, Z.; Yu, J.; Guo, B. Execution Time Prediction for Cypher Queries in the Neo4j Database Using a Learning Approach. *Symmetry* **2022**, *14*, 55. https://doi.org/10.3390/sym14010055

Academic Editor: Roman Tsarev

Received: 23 November 2021
Accepted: 16 December 2021
Published: 1 January 2022

1. Introduction

Pattern query is a fundamental operation for graph query processing, which usually occurs in social network analysis [1], biological network analysis [2], transaction scheduling [3], knowledge graph search [4], and access control [5]. Particularly, these operations are accompanied by a complex query structure, high resource consumption, and long-running time [6,7]. Each data graph easily has billions, and even more, nodes, relationships, and attributions, and a graph topology structure containing local symmetry. Thus, graph pattern query performance prediction before its execution has been a significant issue in modern database management systems (DBMS) [8–10]. Furthermore, predicting query performance (i.e., execution time) is beneficial for many database tasks, including, but not limited to, the following:

- Query scheduling [11,12]: Through prediction technology, it obtains performance metrics to help database management systems decide the number of queries and how to run queries.
- Workload management [13–15]: With the help of prediction models, it can be applied to estimate the workload and improve the query execution efficiency.
- Resource allocation [16,17]: The computational resources (e.g., memory, CPU, cache size) are adjusted to meet the operating requirements of the load and then help the

systems achieve optimal utilization. In most cases, the goal of predicting query tasks is to ensure the timeliness of the load and to maximize resource utilization.

- Progress monitoring [18,19]: According to the result of the query performance prediction, it can provide support for process management by industrial managers and guide progress-monitoring systems to monitor the lifecycle of a process.

Recent studies have commonly applied performance prediction methods on SQL queries; however, they cannot be directly used on Cypher (https://neo4j.com/product/cypher-graph-query-language/ (accessed on 25 February 2021)) queries in the Neo4j (http://www.neo4j.com (accessed on 25 February 2021)) database. There are some basic differences in the data models compared with SQL queries. For example, Cypher queries contain nodes, relationships, and attributes. Additionally, for each query plan, special operators are provided to implement the query task, such as the *NodeByLabelScan* operator acting on nodes and the *Expand* operator acting on relationships. Thus, the performance prediction for Cypher queries is a particular challenge in the Neo4j database. First, the complex schemas and variation information descriptions are considered in Cypher queries. For example, if the filter condition is "Tom" in a query statement, it may return the result of "Tom Hanks", "Tom Cruise", etc. Second, traditional graph query performance prediction methods are mostly based on a query optimizer [20,21] which requires tuning by the database manager. Third, for graph query performance prediction problems, existing research work commonly uses the linear regression approach, which cannot obtain accurate prediction results for complex queries [22,23]. Moreover, graph query performance prediction features selection is also a challenge; the quality of the prediction models depends on the selected query features, and irrelevant features will increase noise during the training process.

To address these challenges, we proposed a learning-based graph pattern queries execution time neural network prediction approach. First, the graph pattern and native data were modeled and encoded as input features to the prediction network, and then the query plan operators were modeled and encoded as features to improve the accuracy of prediction results. Second, we used the RBF neural network to train the feature vectors and used the model to predict new Cypher queries. Compared to relational DBMS, the Neo4j database has not released a standard query benchmark. Thus, we designed a Cypher query benchmark for the Northwind, FIFA2021, and CORD-19 datasets. Additionally, we deployed a monitoring environment for the Neo4j database cluster.

The main contributions of this study are summarized as follows:

(1) To extract the features from Cypher queries, we designed a benchmark of four types of Cypher queries. Additionally, we deployed a monitoring environment for the Neo4j database to collect the query information.

(2) We proposed an effective feature-modeling and -encoding approach for Cypher queries. First, we extracted query plan features and encoded them into vectors. Second, to capture the query structure, we proposed a pattern-modeling method and encoded them into corresponding feature vectors. As far as we know, this is the first pattern used to capture the query structure in Cypher queries.

(3) We proposed an RBF method prediction approach to predict execution time, and three real-world datasets were used to verify its accuracy and effectiveness. The experimental results show that our approach outperformed other approaches.

The rest of this paper is organized as follows: Section 2 describes the substantial research works of query performance prediction; Section 3 briefly introduces the Cypher queries and prediction model architecture; Section 4 presents our query feature modeling and prediction approaches; the evaluation and analysis of the experimental results are performed in Section 5; and Section 6 details the conclusions of this paper.

2. Related Work

Query performance prediction is a significant issue in databases [24]. This section introduces some representative works closely related to our work.

2.1. Relational-Based Queries

The performance prediction approach has been used in relational databases, e.g., Oracle, PostgreSQL, and SQL Server. In [16], the boosted regression tree is proposed to estimate the CPU time and I/O of an individual operator. This approach does not rely on any functioning form and can even fit complex dependencies between input features and output results. In [25], the costs model used by query optimizers cannot meet query execution time prediction. Thus, the authors refined estimates for the query plan before execution and found that the machine-learning approach is better at estimation. However, this approach is only used in a static workload. To meet a dynamic workload for queries, the combination queueing model and buffer pool model are used to predict the query execution time [26]. To overcome the statistic information loss and highly expensive maintenance of the statistic-based approach, the authors first proposed the machine-learning approach [27], the k-nearest neighbor regression (KNN), and support vector machine (SVM) models to predict SPARQL query execution time. For concurrent queries, the contention approach was proposed in [16]. This approach introduces concurrent query intensity and query sensitivity to represent the impact of hardware resource contention and uses the query plan features' characteristics to estimate query performance. In [28,29], the authors focused on representing SPARQL queries with feature vectors and using them to train predictive models, and then used KNN regression and SVM models to predict query performance during the warm and cold stages. A plan-structured deep neural network model for predicting the latency of SQL queries was proposed in [30]. This approach did not need human-crafted feature selection and automatically discovered complex performance models at both the operator and query plan level. In [31], the embedding approach was proposed for concurrent queries to predict performance, and the authors used the graph-structured model to capture the operator feature and the correlations between different operators.

2.2. Graph-Based Queries

Graph query is the most primitive operation for information access, retrieval, and the analysis of graph data [32]. In [33], the authors provided a benchmarking framework for the systematic investigation of query evaluation performance. In [22], the authors developed modeling and learning algorithms for readability and graph pattern matching queries. The former provided three features for graph queries and predicted the execution time using the statistical learning models with LR, RT, RF, and SVM. The latter increased quality features based on the former and employed a multi-label regression model to predict graph query performance, including execution time, query answer quality, and memory consumption. Additionally, a novel long short-term memory (LSTM) learning method was proposed in [34], extracting the query plan features and dynamic system features to predict the query task execution time in the graph database. For graph queries, it is now in the research stage.

In summary, for query performance prediction problems, many studies used heuristic and statistics optimization techniques to estimate query cost, but in many cases, missing data that lead to the prediction result are not accurate. To solve these problems, the machine-learning approach is introduced to predict query performance. However, this approach can only predict a single query or simple query and cannot capture the correlation of complex query features.

Additionally, we also list some works using RBF neural networks. In [35], the authors proposed a simple RBF classifier to allow overlaps in the same pattern class. Additionally, [36] used the RBF network as the multi-label learning framework; each instance was trained to associate with a set of labels and used the frameworks to predict a label set for unseen instances. In [37], an RBF neural network was used to predict traffic flow time series, which has good prospects in traffic stream prediction. Furthermore, RBF neural networks are widely used for classification and regression tasks.

3. Cypher Queries

In this section, we present the overview of the Cypher query process pipeline and briefly describe the query plan and operators. Additionally, we also define the graph models and graph pattern query.

3.1. Overview of the Cypher Query Plan

Figure 1 shows an overview of the generated process of the Cypher query plan. First, the Cypher query statement is submitted to the Cypher parser, and then the parsed statement generates an execution plan by the Cypher query planner; the execution plan tells the database which operators will be selected to complete a query task, and the final query result will be returned to the user by the Cypher Runtime. Additionally, the cost estimator selects the cheapest plan from the candidate execution plan queue by cost in the planner.

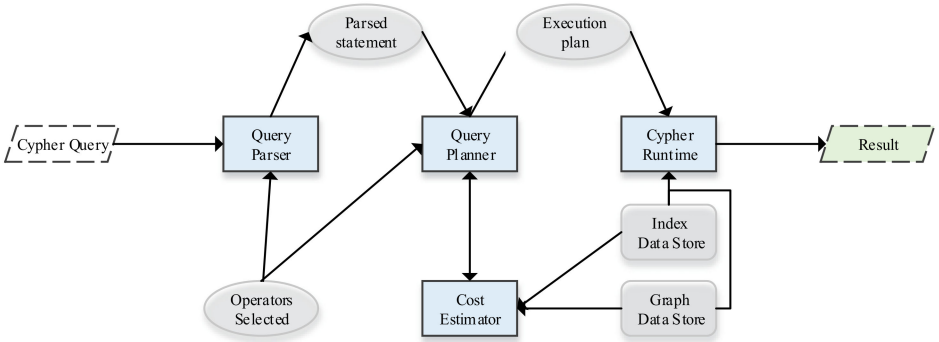

Figure 1. Processing pipeline of Cypher queries.

The execution plan is decomposed into query operators, each of which implements a specific piece of query work. The operators in the execution plan are chained together in a tree; each operator in the execution plan is represented as a node. Compared to SQL queries, it largely contains the same operators, such as Scan, Seek, and Join. The Expand operator is a unique operator used to find all edges from the given node. The Neo4j operator library provides 82 operators (http://www.neo4j.com/docs/cypher-manual\#execution-plans\#operators-summary (accessed on 27 February 2021).) to choose from. Table 1 lists some of the most used query operators. Eager operators are used to complete execution in its entirety before any rows are sent to their parents as input. Updating operators are used as an inquiry that updates the graph. Leaf operators, in most cases, are used to locate the starting nodes and relationships required to execute the query.

Table 1. Some of the most used query operators available in the Neo4j Operator Library.

Operators	Type	Specification
AllNodeScan	Leaf	Scans all nodes from the node store
NodeByLabelScan	Leaf	Utilizes the built-in index on node labels
NodeByIndexScan	Leaf	Scans all nodes in a user-defined index
NodeByIndexSeek	Leaf	Finds nodes using a user-defined index
Expand (All)	None	Traverses all relationships from a given node
CreateUniqueConstraint	Leaf, Updating	Creates a unique constraint on a property for all nodes having a certain label
NodeHashJoin	Eager	Combines two independent results based on an overlapping set of nodes
CartesianProduct	None	Combines two independent results without any overlapping components

3.2. Overview of Performance Prediction Architecture

We consider a set of queries $Q = Q_1, Q_2, \ldots, Q_n$, where each query Q_n is a Cypher query instance. The query execution time will be used as our primary evaluation metric to estimate the query performance. The problem of graph query performance prediction

is training a prediction model to predict the query execution time of each Cypher query instance. As shown in Figure 2, the Cypher query prediction process includes three parts: the training data generator, extraction feature, and an RBF neural network prediction model.

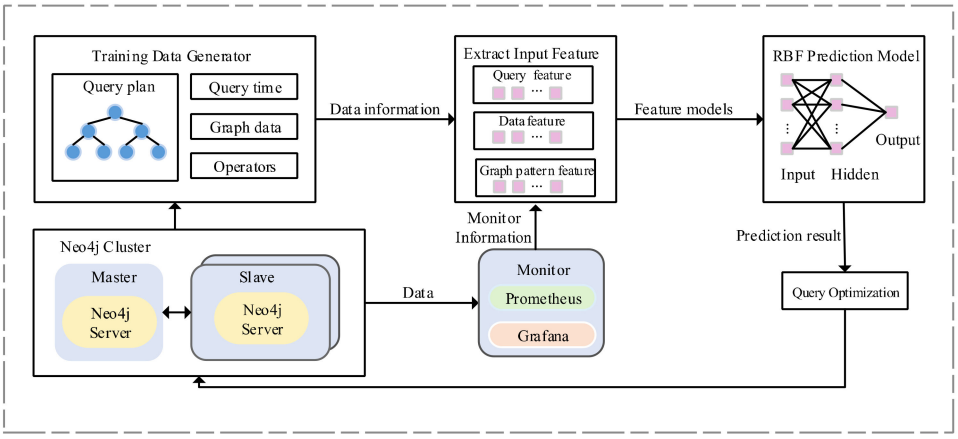

Figure 2. The overview of prediction architecture.

First, the data generator collects query plans, operators, and query records from the Neo4j database. Second, the data information from the generator is modeled and encoded to feature vectors by the feature extractor. Then, these feature vectors feed to the prediction model to train and then use the trained model to predict query performance without executing a query. Thus, users can complete optimization query tasks according to the prediction result. For database management systems, training data are produced by the Neo4j database cluster, which includes three nodes: one master node and two slave nodes. Furthermore, we deployed a monitor node to monitor the servers and collect data information.

4. Cypher Query Execution Time Prediction

To predict the execution time of a query, first, we extract the query plan and graph pattern data, then encode them into feature vectors. Second, we use an RBF neural network as our prediction model, which is used to estimate the query execution time without executing the queries.

4.1. Feature Extraction and Encoding

The performance prediction accuracy is highly dependent on features in the training sets; thus, it is crucial to know how well the feature is represented and how much information can be obtained from Cypher queries. In this section, we propose to capture the plan-level and pattern sequence of two types of features to represent Cypher queries.

Definition 1 (Cypher feature modeling). *Let* $Q_n = (\mathcal{H}, \mathcal{R})$ *denote Cypher queries, where* $\mathcal{H} = \{h_1, h_2, \ldots, h_n\}$ *is the query plan and* $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ *is the pattern in Cypher queries. The feature modeling is the transformation that maps* $Q_n \rightarrow f$*, where* $f \in F_{Q_n}$.

4.1.1. Encoding Query Plan

Each query statement can be transformed into a query plan tree composed of operators through the query planner. As shown in Figure 3, for each operator in the query plan tree, we can pre-define the size of the operator's encoding table according to the Operator Library in the Neo4j database. Then, we can encode operators into a one-hot feature vector. Specifically, we select the operator types, rows, and hits of the query plan tree features to represent the query plan.

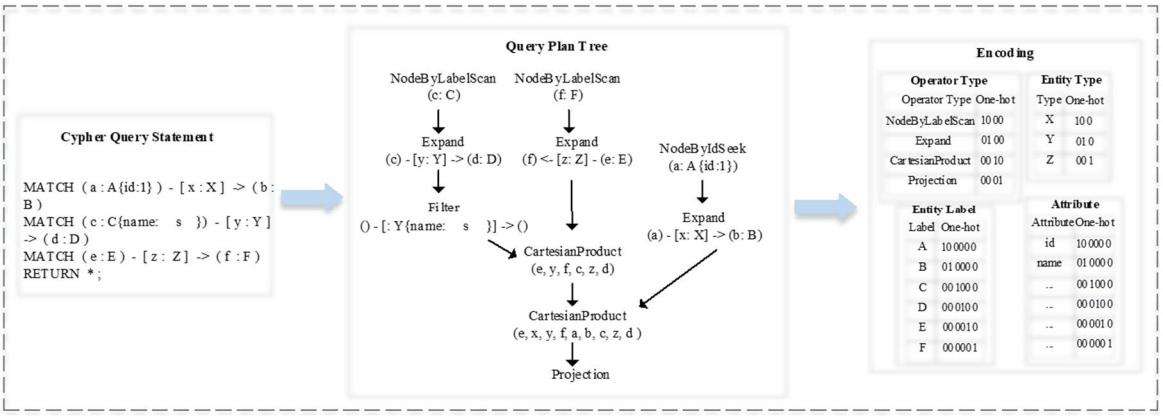

Figure 3. An example of encoding query plan tree for Cypher queries (The * symbol in the RETURN clause means return all query results).

Operator types: As shown in Table 1 and Figure 3, the task of executing a query is divided into different operators, each of which implements a specific piece of query work. We pre-define the size of the operator types' encoding table according to the Operator Library in Neo4j database and encode operator types into a one-hot vector. For example, the NodeByLabelScan can be encoded as "1000".

Rows: The execution of one query plan starts from the leaf nodes of the tree, and then the output of the child node becomes the input of the parent node. Hence, we used the number of rows produced by each operator as the query plan feature.

Hits: Each operator in the query plan tree performs part of the query or update task. The Neo4j storage engine uses the database hits to indicate the operator's work. Thus, the number of hits for each operator should be considered to represent the query plan feature.

For the query plan tree, we denoted the query plan features Operator types, Rows, Hits as O_{s_i}, R_{s_i}, H_{s_i}, then concatenated and normalized these features' vector as follows:

$$F_Q(plan) = [O_{s_i}, R_{s_i}, H_{s_i}] \tag{1}$$

where the s_i is the query plan tree nodes, Q are Cypher queries, and $F_Q(plan)$ is the query plan features vector. If it is padded with 0, it means the node has no features.

4.1.2. Encoding Graph Pattern

In this section, we define the graph model and pattern sequence. Then, we use the pattern sequence to capture the pattern query features, and then encode them into a feature vector.

Definition 2 (Graph model). *Define a graph $G = (N, R, L)$, whereN is the set of nodes, $R \subseteq N \times N \times L$ is a set of labeled directed edges, and L denotes an edge from node n_i to node n_{i+1} with label $l_i \in L$.*

Cypher queries use the pattern to describe the shape of query data; the circles are used to represent nodes, and arrows are used to represent edges. In the Neo4j database, there are the node pattern (e.g., (a)), edge pattern (e.g., (a--b)), and path pattern (e.g., (a-- () --b)). Thus, we use the pattern sequence as defined to describe the Cypher query's expression.

Definition 3 (Pattern sequences). *Given a Cypher query Q_n, an alternating sequence $P_j(Q_n)$ of the form is defined as $P_j(Q_n) = L_1T_1L_2 \ldots T_{n-1}L_n$, where the L_n denotes node labels and T_{n-1} denotes relationship types.*

To capture pattern query features, we used entity labels and types, attributes, and patterns to reflect pattern features.

Node label: Nodes and edges are the basic structures in a graph. For each node, the label attribute is used to describe the graph model, and it requires only one. Thus, we used one-hot to encode the corresponding node label. As shown in Figure 3, the leaf operators have node labels *F* and *C*; these can be encoded into "000001" and "001000".

Relationship type: The arrow describes the relationship between two nodes that have a specific type. For example, the Expand operator has types *X*, *Y*, and *Z* in Figure 3. Thus, the corresponding value of each relationship type can be encoded as "100", "010", and "001".

Attributes: The graph data model consists of nodes, relationships, and attributes. The attribute information is similar to columns in the SQL query tables. For example, the NodeByIdSeek and Filter operators have attribute: id and name, as shown in Figure 3. Thus, the corresponding values of the attributes can be encoded as "100000" and "010000".

Pattern: For each Cypher query, we can extract each query pattern sequence and then use one-hot encoding to encode them into a feature vector. As shown in Figure 3, the Cypher query statement showed three pattern types: $(:A)-[:X]\rightarrow(:B)$, $(:C)-[:Y]\rightarrow(:D)$, and $(:E)-[:Z]\rightarrow(:F)$; the pattern sequence can be denoted as AXB, CYD, EZF. Thus, the pattern AXB that uses the one-hot can be encoded into "100000100010000".

For each Cypher query statement, we denote the Node label, Relationship type, and Attributes as M_Q, R_Q, and A_Q. We then concatenate these feature vectors as follows:

$$F_Q(pattern) = \left[\mathrm{M}_Q,\ R_Q,\ A_Q\right] \tag{2}$$

Thus, the final feature vector is concatenated as follows and is fed into the prediction neural network to train.

$$F_Q = \left[F_Q(plan),\ F_Q(pattern)\right] \tag{3}$$

where the $F_Q(pattern)$ is the pattern feature in Cypher queries, $F_Q(plan)$ is the query plan features, and Q are the Cypher queries.

4.2. Prediction Model

To predict the performance of Cypher queries, we applied the radial-based function (RBF) models [38] to historically executed queries. An RBF neural network includes three-layer structures, including the input, hidden, and output layers. Additionally, the hidden layer maps vectors from a low-dimensional to a high-dimensional space. Compared to the other neural networks, it uses the radial-based function as an activation function to compute vectors and has a strong approximation and fast learning ability in their training phase. Once a prediction model is derived from the training queries, it can then be used to estimate the performance of newly requested queries without executing the query.

The RBF network is composed of a single hidden-layer feed-forward neural network. It has a good generalization ability and a fast-learning speed. The RBF network is denoted as below:

$$\mathcal{F}(v) = \sum_{i=1}^{q} \omega_i \rho(v, c_i) \tag{4}$$

where the q is the number of hidden neurons, ω_i is the weight of i-th hidden neurons, c_i is the center of i-th hidden neurons, and $\rho(v, c_i)$ is the RBF function, usually the Gauss RBF function. The Gauss function is denoted as

$$\rho(v, c_i) = \exp\left(-\beta_i ||v - c_i||^2\right) \tag{5}$$

To start training the RBF neural network, we first need to define a loss function. The loss function is denoted as

$$\min_{\mathcal{F}} L(y_i, \mathcal{F}(v)) = \frac{1}{|N|} \sum_{i=1}^{q} (y_i - \mathcal{F}(v))^2 \tag{6}$$

Given a loss function and a set of input query features, the next step is to adjust the weights of the RBF neural network to minimize the loss function.

5. Experimental Evaluation

5.1. Datasets

Three real-world graph datasets, namely, Northwind Twitter (http://data.neo4j.com/northwind (accessed on 25 January 2021)), FIFA2021 (https://www.kaggle.com/bryanb/fifa-player-stats-database (accessed on 27 January 2021)), and CORD-19 (http://blender.cs.illinois.edu/covid19/ (accessed on 27 January 2021)), were used in this experiment. All datasets need to be processed in a special format before being used in the Neo4j graph database.

Northwind Twitter Dataset: The social network Northwind dataset is provided by Twitter; it contains approximately 5000 entities (e.g., vertex of customer and product, the relationship of the purchase) and about 11,039 pieces of property information (e.g., name, city, address).

FIFA2021 Dataset: This dataset contains 17k+ unique players and more than 60 attributes extracted from the latest FIFA database. It contains an approximately 504 KB node store, 2.5 MB relationships store, and 1.2 MB property store in the graph database.

CORD-19 Dataset: This dataset contains a 2.1 MB node store (e.g., gene, disease, chemical, and organism nodes), a 4.3 MB relationships store (e.g., gene–chemical, chemical–disease associations, gene–disease associations), and a 13 MB property store in the graph database.

To evaluate the proposed prediction models, we developed a query generator to produce training and testing queries from the benchmark in datasets. Compared with the relational databases, the Neo4j DBMS has not released a standard query benchmark. Thus, we designed a Cypher query benchmark relevant to our prediction approach to obtain query workloads. Table 2 lists the basic query benchmark in the Neo4j DBMS.

Table 2. The basic benchmark for Cypher query (The $*$ symbol in N3 means a variable-length identifier in the Cypher syntax).

Operations	Type	Specification
N1	Node pattern Node pattern with label	(a) (a: A)
N2	Edge pattern Edge pattern with relationship type Edge pattern with node label and relationship type	(a)-[r]-(b) (a)-[r: R]-(b) (a: A)-[r: R]-(b: B)
N3	Path pattern Variable-length pattern with lower bound Variable-length pattern with upper bound Variable-length pattern with lower and upper bound Variable-length pattern with any length	(a)-[r1]-(b)-[r2]-(c) e.g., (a)-[$*$1..]-(b) e.g., (a)-[$*$..2]-(b) e.g., (a)-[$*$1..2]-(b) (a)-[*]-(b)
N4	Cycle pattern	(a)-[r1]-(a) (a)-[r1]-(b)-[r2]-(a) (a)-[r1]-(b)-[r2]-(c)-[r3]-(a)

5.2. Monitor

To better obtain metrics on the Cypher query performance prediction features and metrics, we deployed a monitoring environment for the database work node. The monitoring architecture is shown in Figure 4. In addition, the configuration information of the monitor node and work node is shown in Table 3.

As shown in Figure 4, the monitor architecture of the causal cluster includes four nodes, one monitor node, and three work nodes. In the monitor node, the Prometheus server and Grafana visualization tool are deployed to monitor and collect the data transfer information and the metrics of each query instance. The monitor metrics data are stored in the directory (../neo4j/metrics) in the database. Prometheus is an open-source monitoring

framework that provides a universal data model and convenient data collection and query interfaces, such as the Neo4j metrics data collection model.

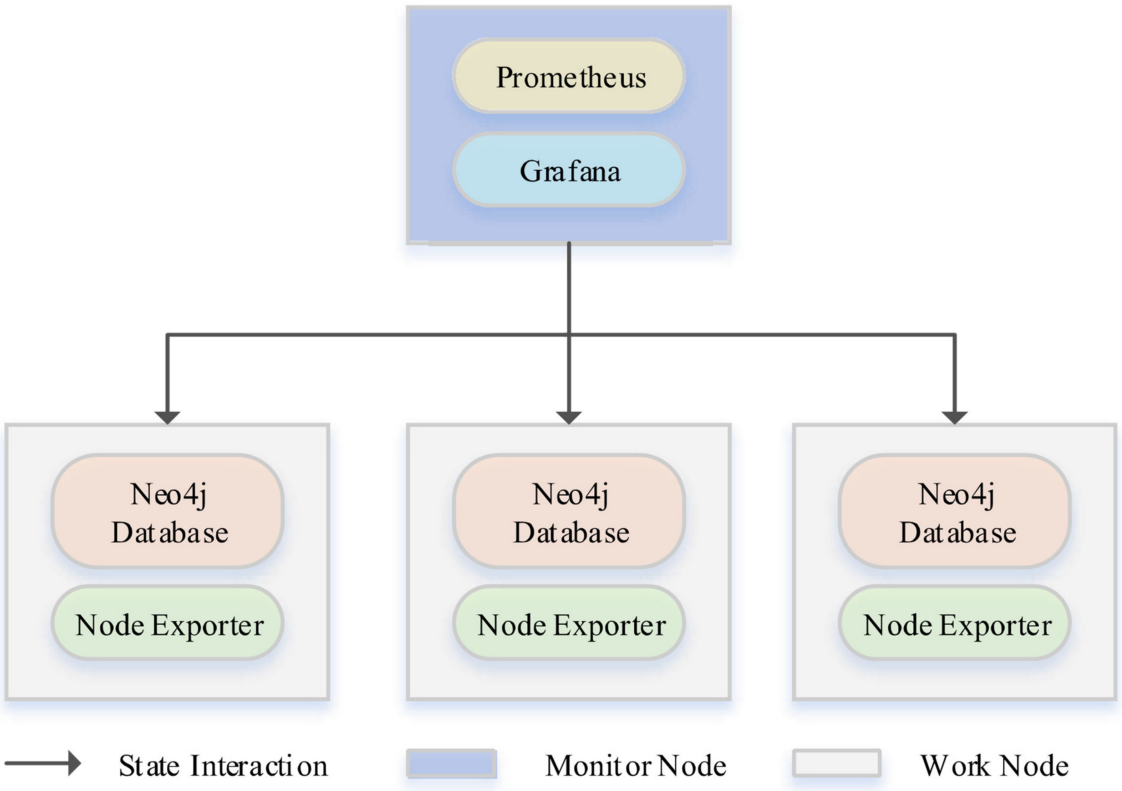

Figure 4. The monitor architecture of Neo4j Causal Cluster.

Table 3. Software configuration for monitoring environment.

Software	Version
Neo4j	neo4j-enterprise-3.5.0
JDK	jdk1.8.0_151
Prometheus Node	prometheus-2.22.1
Exporter Node	exporter-1.0.1
Grafana	grafana_7.3.1
OS	Ubuntu16

5.3. Evaluation Techniques

In this experiment, for comparative analysis, we mainly considered traditional machine-learning and deep-learning models.

Traditional machine learning: We implemented the traditional machine-learning approaches of line regression (LR), regression trees (RT), and support vector regression (SVR) that were proposed in [22]. The authors proposed five types of statistical machine-learning models (linear regression [39], regression tree [40], random forest [41], KNN [42], and SVM [43]) to predict graph query performance. In this experiment, for comparative analysis, we mainly considered three kinds of prediction models as the prediction models to predict query performance.

Neural network approach: We implemented the fully connected neural network (FCNN) prediction model that was proposed in [44]; the FCNN model includes on input layer, one hidden layer, and one output layer. This approach uses query structure, predicates, and heuristics features as input vectors and outputs the query prediction cost.

5.4. Experiment Results of Prediction Models

To evaluate the errors of these prediction models, we used the root mean square error (RMSE) as the evaluation metric. It is denoted as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{j}(\hat{y}_i - y_i)^2} \tag{7}$$

where n is denoted as the number of templates, $\hat{y}_i$ is denoted as the prediction values for Cypher queries, and y_i is denoted as the actual values.

Table 4 shows the result of the RMSE of each prediction model on the three datasets. In the Northwind dataset, the node numbers are much smaller than in the other datasets. Thus, the result of the RMSE of N1 was the smallest. Compared to N2, N3, and N4, the N1 benchmark had minimal complexity; the error was also lower than that of the other benchmarks. In the three kinds of statistical learning models, the LR model had the largest error; its RMSE was greater than that of the SVR and RT models. The result of the RMSE of the RBF model was the smallest.

Table 4. The result of root mean square error (RMSE).

Datasets	Benchmark	LR	SVR	RT	FCNN	RBF
Northwind	N1	2.28	2.49	2.22	2.53	1.93
	N2	15.96	15.53	15	15.88	13.92
	N3	15.25	18.61	13.89	15.21	13.99
	N4	18.87	15.24	10.64	13.87	10.68
FIFA2021	N1	6.26	6.75	6.06	6.01	5.95
	N2	16.52	6.6	6.44	7.23	6.23
	N3	11.04	9.39	9.38	15.02	8.84
	N4	7.56	7.07	7.18	10.16	5.09
CORD-19	N1	7.70	8.14	8.17	11.46	6.87
	N2	5.18	5.04	5.01	5.98	5.0
	N3	7.04	6.89	6.90	7.24	6.78
	N4	13.06	12.56	12.54	16.86	12.23

Additionally, we also used the mean relative error as our evaluation metric; the mean relative error is denoted as follows:

$$MreEror = \frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{y}_i - y_i|}{y_i} \times 100\% \tag{8}$$

where the $\hat{y}_i$ are the predicted values for Cypher queries, y_i are the actual values, and n is the number of templates.

Figure 5 shows the result of the mean relative errors of five models on the three kinds of datasets. Compared with the statistical learning models, the SVR model had the highest mean relative error. Compared to the neural network model, the RBF model's mean relative error was much lower than that of the FCNN models. In the N1 benchmark, the SVR models had the largest error of 43% in the Northwind dataset; the FCNN and SVR models had a small difference in mean relative error in the FIFA2021 dataset. Additionally, in the CORD-19 dataset, except for the FCNN model, the other model's error was almost the same. Similarly, the average mean relative error of the RBF model in the Northwind dataset reached 16.5%; in the FIFA2021 dataset, 12%; and in the CORD-19 dataset, 16.25%. The size and complexity of an experiment's datasets can affect the query performance prediction. With respect to the difference between the Northwind dataset and the CORD-19 dataset, the former has five types of nodes and four types of relationships, and the latter has three types of nodes and two types of relationships. Additionally, the FIFA dataset's underlying data are the simplest; the corresponding query category variables are also less

than those of the other datasets. Additionally, the prediction models can also directly affect the results predicted.

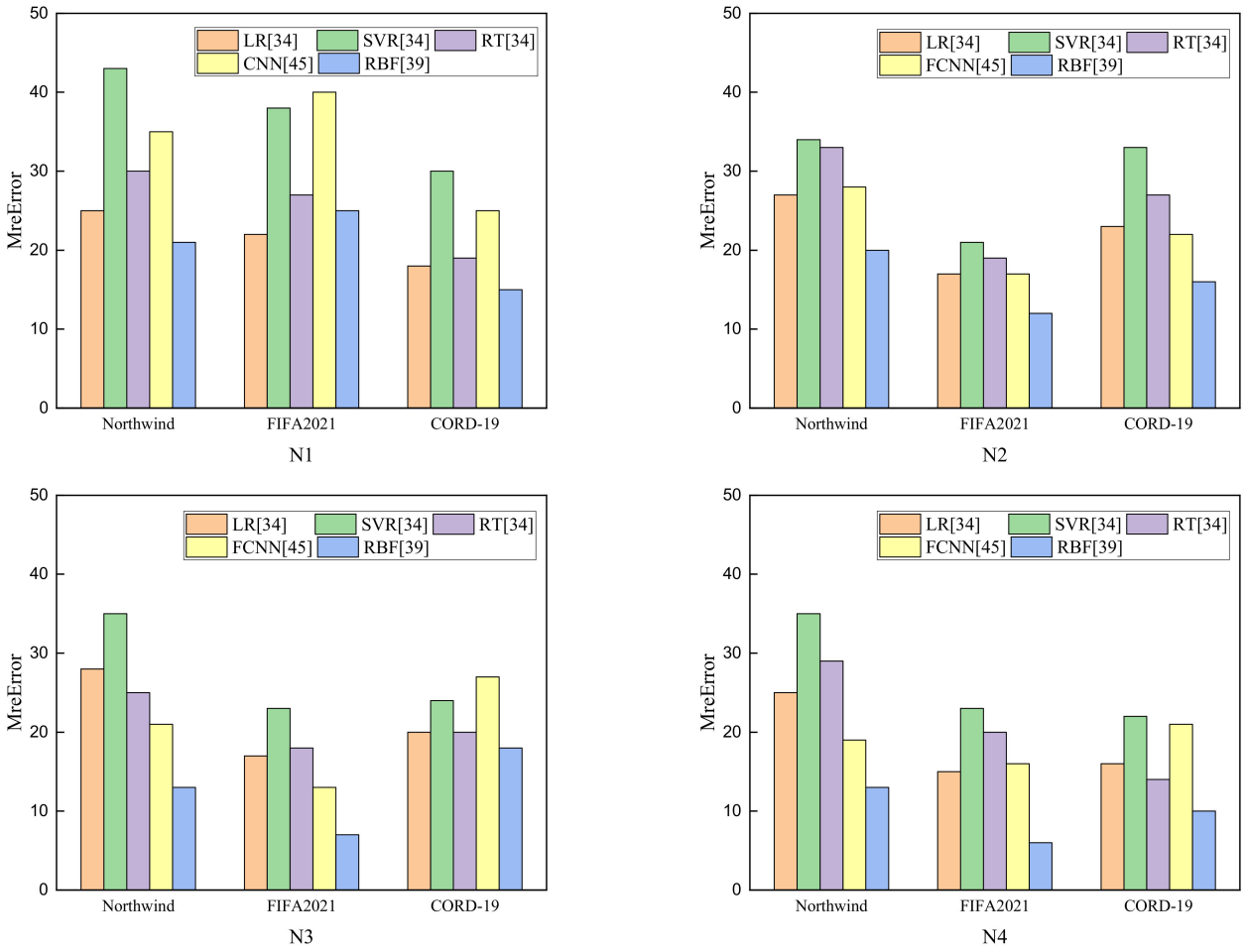

Figure 5. The mean relative error of the four benchmarks on the three datasets in five types of models: LR [22], SVR [22], RT [22], FCNN [44], and RBF [38].

Figure 6 shows a comparison between the predicted execution time and the actual execution time of the five models for four benchmarks on the Northwind dataset. In the N1, N2, and N3 benchmarks, we found that several actual values were much larger or much smaller than other actual values due to the influence of the database system (e.g., network bandwidth, CPU and I/O utilization, etc.). In Figure 6, there is a large error between the predicted value and the actual value of the SVR model. In contrast, this situation did not appear for other benchmarks; we will explore the main reasons for this in future work. Additionally, we also found that the predicted value of all the samples fluctuated with the fluctuation of the true value; thus, the accuracy and effectiveness of the prediction models are further verified. As shown in Figure 6, the predicted value of all the templates fluctuated, followed by the actual value, verifying the effectiveness of the five prediction models. The results for the FIFA2021 dataset and CORD-19 dataset are omitted for space considerations.

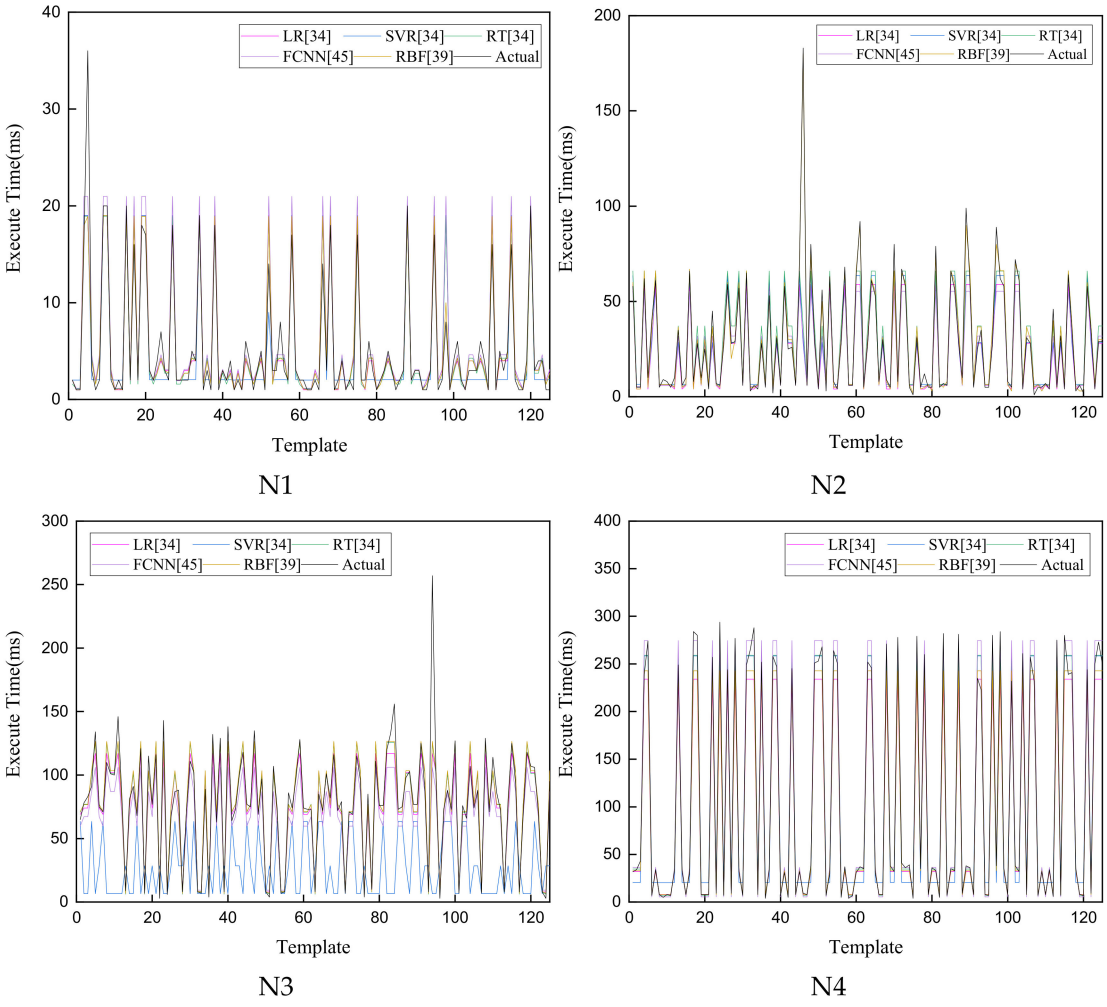

Figure 6. Comparison between the actual execution time and the predicted execution time in the Northwind dataset benchmarks (N1, N2, N3, and N4) with five types of models: LR [22], SVR [22], RT [22], FCNN [44], and RBF [38].

As shown in Figure 7, the performance accuracy becomes better and better as the training data size grows in the training process. However, in most other research work, less query training data are used in evaluation experiments. The reason for this is that obtaining features of a large collection of query samples is a time-consuming process. Additionally, the prediction effect may be poor for unexpected queries. Nevertheless, we will still consider expanding the size of the query samples to cover more of a variety of queries in the future. It can be concluded from Figure 8 that the normalized time of the FCNN and RBF models are much higher than that of the statistical learning models. The reason for this is that, compared with statistical models, neural network models need to learn more parameters.

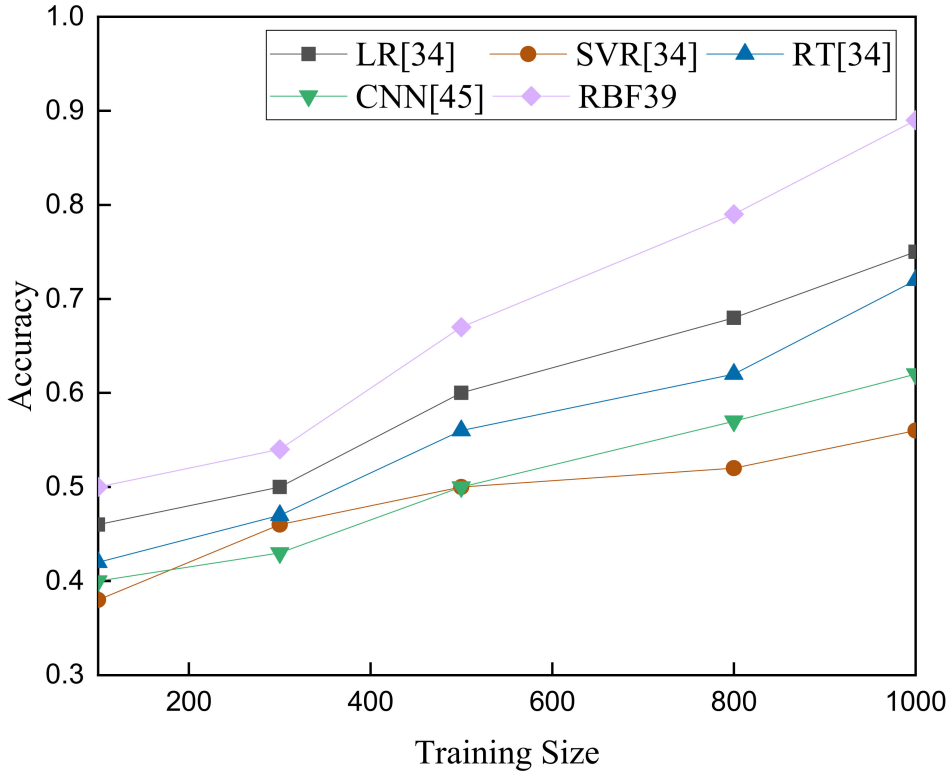

Figure 7. The accuracy of training size in five types of prediction models: LR [22], SVR [22], RT [22], FCNN [44], and RBF [38].

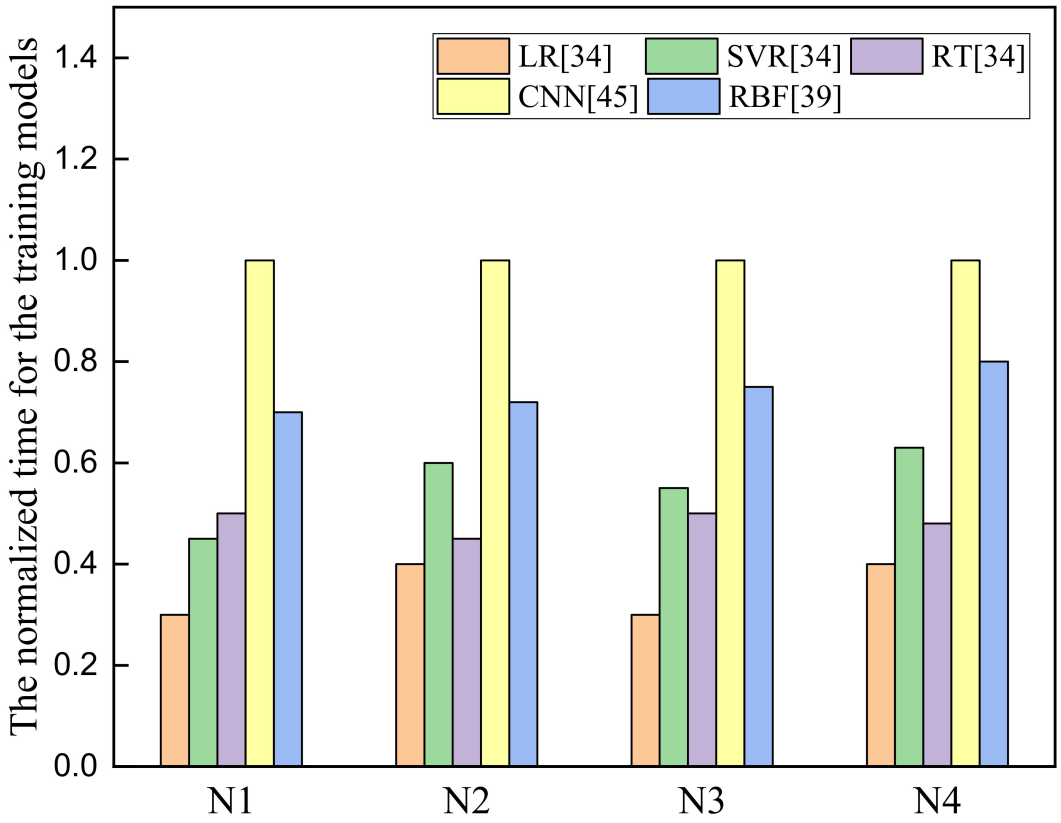

Figure 8. The normalized time of training for the five types of prediction models: LR [22], SVR [22], RT [22], FCNN [44], and RBF [38].

6. Conclusions

In this paper, we used an RBF neural network as a prediction model to predict the Cypher query execution time in the Neo4j database. We combined the query plan, query patterns, and data information to model corresponding features for each Cypher query and then encoded them into feature vectors. Additionally, we also deployed a monitoring environment for the Neo4j clusters to collect statistical information. In the experiment, we designed four types of Cypher query benchmarks in the three real-world datasets to verify the effectiveness of the prediction models. The experimental results of the four benchmarks show that the average mean relative error in the Northwind dataset reached 16.5%; in the FIFA2021 dataset, 12%; and in the CORD-19 dataset, 16.25%. Compared to the other datasets, the FIFA dataset's underlying data are simpler; the corresponding query category variables are also less than those of the other datasets. Thus, the more complex the underlying data, the lower the prediction accuracy.

The study of performance prediction for Cypher queries is in its infancy. To solve the crucial issue of graph queries in the fields of workload management, query scheduling, resource allocation, progress monitoring, etc., our future work can be considered as follows:

1. Query-driven and data-driven methods have their advantages and disadvantages; how to combine the two methods to improve performance prediction accuracy is important for research work.
2. Sparse and high-dimensional representation is a challenge in learning-based methods; existing research works have explored some methods of representation learning [30], but there is still a significant amount of research space for the dense input.
3. The execution time of a plan depends on the sum of the time cost of all operators. Therefore, in addition to the query plan and pattern features, we should consider the impact of the hardware deployment environment and the state of the database (e.g., buffer, stream queries, concurrent queries) for the query performance.

Author Contributions: Supervision, J.Y. and B.G.; Writing—review & editing, Z.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Natural Science Foundation of China, grant number 61862060.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Drakopoulos, G.; Kanavos, A.; Tsakalidis, A.K. Evaluating Twitter Influence Ranking with System Theory. In Proceedings of the 12th International Conference on Web Information Systems and Technologies (WeBIST), Rome, Italy, 23–25 April 2016; pp. 113–120.
2. Lysenko, A.; Roznovăţ, I.A.; Saqi, M.; Mazein, A.; Rawlings, C.J.; Auffray, C. Representing and querying disease networks using graph databases. *BioData Min.* **2016**, *9*, 1–19. [CrossRef]
3. Guirguis, S.; Sharaf, M.A.; Chrysanthis, P.K.; Labrinidis, A.; Pruhs, K. Adaptive scheduling of web transactions. In Proceedings of the IEEE 25th International Conference on Data Engineering, Shanghai, China, 29 March–2 April 2009; pp. 357–368.
4. Paulheim, H. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semant. Web* **2017**, *8*, 489–508. [CrossRef]
5. Rizvi, S.Z.R.; Fong, P.W.L. Efficient Authorization of Graph-database Queries in an Attribute-supporting ReBAC Model. *ACM Trans. Priv. Secur. (TOPS)* **2020**, *23*, 1–33. [CrossRef]
6. Dinari, H. A Survey on Graph Queries Processing: Techniques and Methods. *Int. J. Comput. Netw. Inf. Secur.* **2017**, *9*, 48–56. [CrossRef]
7. Scabora, L.C.; Spadon, G.; Oliveira, P.H.; Rodrigues, J.F.; Traina, C. Enhancing recursive graph querying on RDBMS with data clustering approaches. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, 30 March–3 April 2020; pp. 404–411.

8. Hauff, C.; Azzopardi, L. When is query performance prediction effective? In Proceedings of the 32nd international ACM SIGIR conference on Research and Development in Information Retrieval—SIGIR, Boston, MA, USA, 19–23 July 2009; pp. 829–830.
9. Zendel, O.; Shtok, A.; Raiber, F. Information needs, queries, and query performance prediction. In Proceedings of the 42nd International ACM uSIGIR Conference on Research and Development in Information Retrieval, Paris, France, 21–25 July 2019; pp. 395–404.
10. Holzschuher, F.; Peinl, R. Performance of graph query languages: Comparison of cypher, gremlin and native access in neo4j. In Proceedings of the Joint EDBT/ICDT 2013 Workshops, Genoa, Italy, 18–22 March 2013; pp. 195–204.
11. Li, J.; Ma, X.; Singh, K. Machine learning based online performance prediction for runtime parallelization and task scheduling. In Proceedings of the 2009 IEEE International Symposium on Performance Analysis of Systems and Software, Boston, MA, USA, 26–28 April 2009; pp. 89–100.
12. Macdonald, C.; Tonellotto, N.; Ounis, I. Learning to predict response times for online query scheduling. In Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, Portland, OR, USA, 12–16 August 2012; pp. 621–630.
13. Raza, B.; Kumar, Y.J.; Malik, A.K.; Anjum, A.; Faheem, M. Performance prediction and adaptation for database management system workload using Case-Based Reasoning approach. *Inf. Syst.* **2018**, *76*, 46–58. [CrossRef]
14. Duggan, J.; Cetintemel, U.; Papaemmanouil, O. Performance prediction for concurrent database workloads. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, Athens, Greece, 12–16 June 2011; pp. 337–348.
15. Raza, B.; Mateen, A.; Sher, M.; Awais, M.M. Self-prediction of performance metrics for the database management system workload. *Int. J. Comput. Theory Eng.* **2012**, *4*, 198. [CrossRef]
16. Li, J.; König, A.C.; Narasayya, V.; Chaudhuri, S. Robust estimation of resource consumption for SQL queries using statistical techniques. *arXiv* **2012**, arXiv:1208.0278. [CrossRef]
17. Duggan, J.; Papaemmanouil, O.; Cetintemel, U.; Upfal, E. Contender: A Resource Modeling Approach for Concurrent Query Performance Prediction. In Proceedings of the Extending Database Technology, Athens, Greece, 24–28 March 2014; pp. 109–120.
18. Murugesan, M.; Shen, J.; Qi, Y. Resource Estimation for Queries in Large-Scale Distributed Database System. U.S. Patent 10,762,539, 1 September 2020.
19. Kang, B.; Kim, D.; Kang, S.H. Periodic performance prediction for real-time business process monitoring. *Ind. Manag. Data Syst.* **2012**, *112*, 4–23. [CrossRef]
20. Zhao, P.; Han, J. On graph query optimization in large networks. In Proceedings of the VLDB Endowment, Singapore, 13–17 September 2010; pp. 340–351.
21. Das, S.; Goyal, A.; Chakravarthy, S. Plan before you execute: A cost-based query optimizer for attributed graph databases. In Proceedings of the International Conference on Big Data Analytics and Knowledge Discovery, Porto, Portugal, 6–8 September 2016; pp. 314–328.
22. Namaki, M.H.; Sasani, K.; Wu, Y. Performance prediction for graph queries. In Proceedings of the 2nd International Workshop on Network Data Analytics, Chicago, IL, USA, 19 May 2017; pp. 1–9.
23. Sasani, K.; Namaki, M.H.; Wu, Y. Multi-metric graph query performance prediction. In Proceedings of the International Conference on Database Systems for Advanced Applications, Gold Coast, QLD, Australia, 21–24 May 2018; pp. 289–306.
24. He, B.; Ounis, I. Query performance prediction. *Inf. Syst.* **2006**, *31*, 585–594. [CrossRef]
25. Wu, W.; Chi, Y.; Zhu, S.; Tatemura, J.; Hacigümüs, H.; Naughton, J.F. Predicting query execution time: Are optimizer cost models really unusable? In Proceedings of the 2013 IEEE 29th International Conference on Data Engineering (ICDE), Brisbane, QLD, Australia, 8–12 April 2013; pp. 1081–1092.
26. Wu, W.; Chi, Y.; Hacígümüş, H. Towards predicting query execution time for concurrent and dynamic database workloads. *Proc. VLDB Endow.* **2013**, *6*, 925–936. [CrossRef]
27. Hasan, R.; Gandon, F. A Machine Learning Approach to SPARQL Query Performance Prediction. In Proceedings of the International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) IEEE, Washington, DC, USA, 11–14 August 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 266–273.
28. Zhang, W.E.; Sheng, Q.Z.; Taylor, K. Learning-based SPARQL query performance prediction. In Proceedings of the International Conference on Web Information Systems Engineering, Shanghai, China, 8–10 November 2016; pp. 313–327.
29. Zhang, W.E.; Sheng, Q.Z.; Qin, Y. Learning-based SPARQL query performance modeling and prediction. *World Wide Web* **2018**, *21*, 1015–1035. [CrossRef]
30. Marcus, R.; Papaemmanouil, O. Plan-structured deep neural network models for query performance prediction. *arXiv* **2019**, arXiv:1902.00132. [CrossRef]
31. Zhou, X.; Sun, J.; Li, G. Query performance prediction for concurrent queries using graph embedding. *Proc. VLDB Endow.* **2020**, *13*, 1416–1428. [CrossRef]
32. Namaki, M.H.; Chowdhury, F.A.; Islam, M.; Doppa, J.; Wu, Y. Learning to Speed Up Query Planning in Graph Databases. *arXiv* **2018**, arXiv:1801.06766.
33. Izsó, B.; Szatmári, Z.; Bergmann, G. Towards precise metrics for predicting graph query performance. In Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), Silicon Valley, CA, USA, 11–15 November 2013; pp. 421–431.

34. Chu, Z.; Yu, J.; Hamdulla, A. A novel deep learning method for query task execution time prediction in graph database. *Future Gener. Comput. Syst.* **2020**, *112*, 534–548. [CrossRef]
35. Fu, X.; Wang, L. Data dimensionality reduction with application to simplifying RBF network structure and improving classification performance. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **2003**, *33*, 399–409.
36. Zhang, M.L. M l-rbf: Rbf neural networks for multi-label learning. *Neural Process. Lett.* **2009**, *29*, 61–74. [CrossRef]
37. Chen, D. Research on traffic flow prediction in the big data environment based on the improved RBF neural network. *IEEE Trans. Ind. Inform.* **2017**, *13*, 2000–2008. [CrossRef]
38. Broomhead, D.S.; Lowe, D. *Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks*; Royal Signals and Radar Establishment: Malvern, UK, 1988.
39. Ganapathi, A.; Kuno, H.; Dayal, U.; Wiener, J.L.; Fox, A.; Jordan, M.; Patterson, D. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In Proceedings of the 2009 IEEE 25th International Conference on Data Engineering, Shanghai, China, 29 March–2 April 2009; pp. 592–603.
40. Loh, W.Y. Classification and regression trees. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2011**, *1*, 14–23. [CrossRef]
41. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
42. Aha, D.W.; Kibler, D.; Albert, M.K. Instance-based learning algorithms. *Mach. Learn.* **1991**, *6*, 37–66. [CrossRef]
43. Shevade, S.K.; Keerthi, S.S.; Bhattacharyya, C. Improvements to the SMO algorithm for SVM regression. *IEEE Trans. Neural Netw.* **2000**, *11*, 1188–1193. [CrossRef] [PubMed]
44. Negi, P.; Marcus, R.; Mao, H. Cost-Guided Cardinality Estimation: Focus Where it Matters. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW), Dallas, TX, USA, 20–24 April 2020.