DR-QPO: DISCRETE RANK BASED QUERY PATTERN OPTIMIZATION TOWARDS PARALLEL QUERY PLANNING AND EXECUTION FOR DISTRIBUTE TRIPLE STORES

K.SHAILAJA[1], DR. P.V. KUMAR[2], DR S.DURGA BHAVANI[3]
[1]Department of CSE, Methodist College of Engineering and Technology, Hyderabad, Telangana State, India
[2]UCE, OU, HYDERABAD, Telangana State
[3]School of IT, JNTUH Hyderabad, Telangana State, India
Email: [1] shailajamtech2006@yahoo.co.in, [2] pvkumar58@gmail.com, [3] sdurga.bhavani@gmail.com

ABSTRACT

This manuscript proposed and explored a novel strategy for query pattern optimization towards parallel query planning and execution in Distributed RDF environments. The critical objective of the proposal is to optimize the query patterns from the query chains initiated to execute parallel in distributed RDF environment, which is unique regard to the earlier contributions related to parallel query planning and execution strategies found in contemporary literature. All of these existing models aimed to notify the query patterns from the given query chain, which are less significant to optimize the parallel process of the query patterns that discovered from multiple query chains submitted in parallel in distributed environment (such as cloud computing) to query the distributed triple stores. In order to this, the Discrete Rank based Query Pattern Optimization (DR-QPO) strategy is proposed. The DR-QPO optimizes the query patterns from multiple query chains initiated in parallel. A novel scale called Discrete Rank Consistence Score (DRDCS) defined, which uses the order of other metrics query pattern occurrence count, search space utilization, and access cost as input. The experiments conducted on the proposed model and other benchmark models found in contemporary literature. The results obtained from the experimental study evincing that the proposed model is significant and robust to optimize the query patterns in order to execute distribute query chains in parallel. The comparative analysis of the results obtained from DR-QPO and other contemporary models performed using ANOVA standards like t-test, Wilcoxon signed rank test.

Keywords: *RDF, Query Optimising, Parallel Planning, SPARQL, DRDCS, Access Cost Search Space Utilization Distributed Query Science.*

1 INTRODUCTION

Semantic web solutions are gaining prominence, and RDF (Resource Description Framework) is one of the flexible data models that are developed for semantic web [1]. RDF is profoundly used in varied range of applications like Semantic Web [2] [3], scientific applications and web 2.0 solutions [4] and databases [5]. Numerous researches were carried out and contemporary solutions were developed with distributed solutions for storage and querying in RDF data.

Some of the contemporary RDF data storage solutions are Babel Peers [6], 3rdf [7], RDF Peers [8], Atlas [9] and Gride Vine [10] which uses P2P overlay networks for storing and querying RDF data using distributed process. To obtain quality search results, RDF triples with same concept, predicament is stored three times in the network for every triple component in an individual manner, in the RDF data stores.

Usually the triples are clustered with same identifier over the same peer, and such action address the issue of constraint search for a specific subject, predicate or an object from a local database. Though SPARQL3, RDF query language has varied salient features like aggregation and optional clause, still the popular dialect adapted is the conjunctive queries like BGP (Basic Graph Pattern) queries that typically feature in many equality joins.

Despite the fact that no single partitioning can assure that all the queries are PWOC, still majority of the queries need processing across varied nodes and hence, data redistribution across nodes are essential. Based on the complexity of the processed query, the impact of evaluating distributed part of the query plan might vary. In deciding the pattern of decomposing and evaluating a RDF query over a parallel context, logical query optimization plays a vital role. In the distributed data management [11], for effective assessment of queries, there is need for maximizing parallelism and taking advantage of distributed processing capacity, which essentially reduces the response time.

In the context of a parallel RDF query evaluation, intra-operator parallelism depends on combined operators which process the chunks of data in parallel. Developing massively –parallel plans can support in improving the inter-operator parallelism, as few combined operators are feasible to any root-to-leaf path in the plan. Such development could be attributed to processing carried out by the joints directly and it doubles up in the response time. Some of the earlier works like binary joints organized in bushy plans [12] with joints (n>2) in the first level of plans and the binary joins in next level [13] [14] [15] or the joins at all the levels [16], but with organized in left deep plans. The aforesaid methods result in high plans and high responses times.

However, Hadoop RDF [17] is an effective model designed for developing bushy plans of joins, but the constraints pertaining to assurance of flat plans is not guaranteed. The other major constraint observed in the aforesaidmodels are about overlay load balancing. As a result of non-uniform distribution of the frequency relate to subject, predicate and object oriented occurrences in triples, it is feasible that the triples may not be stored over the peers of underlying the networks in a uniform manner.

It is imperative from the constraints discussed above, that query optimization is certainly a NP-hard problem and many of the near-optimal query plans proposed are of NP-hard [18]. Runtime increment issues are envisaged for the algorithms upon the rising number of joins and contemporary application scenarios [19]. Also the complexity of systems over which the query is processed also increases because of load factor. To address such issues, contemporary models have put forth new models and contemporary processing operators for introducing varied new parameters for tuning the query processing [20] [21]. Such significant developments can support query optimization much harder than the size of search space increment. Further, many of the developments also influence new cost metrics for considering query plans apart from the execution time. In the other dimension, the challenge is that when multiple plan cost metrics are considered, it could lead to harder-set of query optimization too [22] [23] [24]. In a comprehensive scenario, it can be stated that majority of the emerging solutions for query optimization are resulting in NP hard and also increase the need for parallel query optimization algorithms.

In regard to this, the contribution of the manuscript tends to achieve the following:

- Optimizing the parallel queries targeted to search on distributed RDF store.
- A distributed ranking model should depict to optimize the query processing time and resource utilization
- Optimization process should evince the linear complexity

2 RELATED WORK

Parallel query optimization can be termed as the serial optimization algorithms generating plans, which are executed in parallel [25] for query processing. Though contextually there are varied considerations in realizing the term, in the current scenario a parallel algorithm for generating query plans are discussed.

In [8] the authors have focused on issue of load-balancing by curtailing the storage of very popular URIs and literals, depending on storage capacity of local peers. Though the process might provide the desired outcome, still the challenge is about probable loss of complete result. In [6] the study has targeted to address the problem by developing an overlay tree over DHT position of an overly popular triple component. Such type of load balancing shall be very fragile in the instance of node failure in overlay tree, and could even result in loss of the complete branch of tree.

In [26], the study has worked on huge variation among the peer's data load upon triples being indexed 3 times by fixed hash depth of 1. For improving the data distribution, the proposed model has focused on idea of indexing triples using random hash depth, wherein for every depth there shall be equal number of potential location keys for triple components. It showed that with higher value of hash depth, the better is the triples distribution among peers. But it comes with cost of network communication in the instance of query evaluation, as many peers are queried for evaluation of triple pattern.

In [27], for addressing the hotspots of unfair load balancing, a solution in terms of indexing a triple for every possible combination of its two components like subject + object, predicate + object, subject + predicate and such combination are proposed. The impact of such process was extra storage of triples in the network, despite of achieving fair triple load distribution.

In [9], for improving the query load distribution, study has focused on additionally indexed triples by combination of varied triple components, resulting in 7 replications for every triple in total. It is used in extra storage overhead for distribution of query processing load amidst the peers. But the key element missing was about studying the utilization of overhead for improving the response time for query processing.

In [28], the notion of quad is used for representing the RDF data and a solution with optimized index structure for supporting evaluation of RDF queries in centralized RDF data stores are considered.

The study reflects that only 6 indexes are essential for covering the entire range of 16 access patterns, but every access pattern is considered a quad when there is combination of subject, predicate, object or the context is either defined or a variable. Considering such scope for reducing the number of essential indexes, in section IV-B only 3 indexes are required for new index scheme to ensure coverage of all the 8 possible triple patterns.

The proposed work connects to the earlier works that parallelize classical dynamic program designed with query optimization algorithm [29], [30], [31], [32], [33], [34], and [35]. Many of the prior algorithms were devised for shared-memory structures, which lack scalability over a certain level of parallelism [36]. In the other way, prior algorithms evaluated to consider certain volume, but the algorithm devised in [35] evinced scalability with shared-nothing architecture comprising over 250 workers. Some of the key factors that differentiate prior algorithms to proposed algorithm is the limitation and scalability.

In the earlier algorithms, the query patterns that are independent, cyclic or recursive only planned to execute in parallel as independent threads and there is presumption that all the threads might share common data structures and thus it is easier to access intermediary results generated in the other threads. This practiceleads to huge communication overhead over shared-nothing architectures and moreover, these algorithms least significant towards optimizing query patterns found multiple query chains submitted in distributed environment. In addition, the earlier algorithms have relied on central coordinator for assigning rather fine-tuned optimization of tasks to the work threads. Two of the key disadvantages of such process are: i) it needs considerable communication among the master and workers ii) increased levels of time complexity in managing at central system, which leads to delay in implementing parallelism.

In regard to this, the proposed algorithm does optimize the query patterns that are independent, recursive or cyclic in multiple query chains that effectively workin shared nothing distributed environment with multiple clients and multiple distributed triple stores.

3 DISCRETE RANK BASED QUERY PATTERN OPTIMIZATION

This section explores the methods and materials used discrete rank based query pattern optimization for distributed query planning for RDF stores. The initial step of the proposed model finds the query patterns of size 1 to n and ranks them in ascending order of their occurrence count, such that highest rank is most optimal. Further rank these patterns in descending of respective search space utilization, such that the query pattern with highest rank is most optimal. Afterwards, allocate ranks to the same patterns in descending order of their access cost that similar to the ranking strategy followed for search space utilization, such that the pattern with highest rank is most optimal towards access cost. At this stage, each pattern discovered enlisted with three discrete ranks corresponding to their occurrence count, search space utilization and access cost. Further the process of estimating theDiscrete Rank distribution consistency score (DRDCS) of the query patterns is performed, which considers further toallocate global rank of each of the pattern. The global rank assigned to all of these patterns in descending order of their DRDCS. The detailed exploration of these phases depicted in following sections.

RDF structure based semantic web search usually carried out using SPARQL queries by indicating them in the form of tree structure. Query chosen for the process categorizes in to three distinct formats like left deep, right deep and bidirectional deep tree, which is termed as bushy tree. In the instance of any of the trees retaining subjects, predicates, or the objects as leads and join them as nodes, then the depth first tree is formed first, and the connection amidst the nodes will form a bushy structure, whereas the join connected to subject predicate or object connects to the other join which could conclude left deep tree. The emphasis of proposed model is to optimize the parallel query planning process and reduce the cost of execution. Hence, the query structure chosen for exploring the proposed solution is bidirectional or bushy tree.

The depicted model is explored by considering the query structures defined in renowned query language called SPARQL However this model can adapt to any of the formal query languages available.

3.1 Query Patterns discovery

This section explores the Tabu Search based query patterns and their occurrence count discovery. Initial task of the proposed model is to identify the query patterns of size 1andtabu transactions (no need to search) respective to each discovered pattern of size 1. Further, the search will be conducted recursively for two-size query patterns to max possible size of query patterns. During the search, the insignificant patterns and insignificant transactions respective to significant patterns place in respective tabu lists, which ignores from search space in further iterations.The proposed model is using tabu search in order to limit the process time.The detailed pattern discovery process explored following:

All query elements $E = \{e_1, e_2, - - - - e_m\}$ of size m are a finite set those appear in the set Q of distributed query-chains given. The query-chain set Q of size n represents the all distributed query chains $\{q_1, q_2, q_3, ..., q_n\}$ such that each query-chain formed by the subset of query elements in set E, such that each query-chain $\{q_r \exists q_r \in Q \wedge 1 \le r \le n\}$ is the superset of one or more query elements QE. Each query-chain is notified by a unique query-chain id $\{r \exists 1 \le r \le n\}$, which referred further as qid. Each query element e_j in a given query-chain represents query element e with id $1 \le j \le m$.

Further, the query elements E considers as 1-size patterns set and determines their occurrence count and tabu list of transactions respective to each 1-size pattern as follows.

$qp_1 \leftarrow E$ // clone the set of all query elements as 1-size query patterns set qp_1

$qps \leftarrow qp_1$ // The set qps retains all possible query patterns discovered that initialized by moving all 1-size patterns

```
step 1. $\overset{m}{\underset{i=1}{\forall}} \{p_i \exists p_i \in qp_1\}$ Begin
step 2. $o(p_i) = 0$ // denotes the occurrence count
    of pattern $p_i$ that initialized to 0;
step 3. $o(p_i) = \sum_{j=1}^{n} \{1 \exists p_i \subseteq q_j \wedge q_j \in Q\}$
step 4. $tlq(p_i)$ // is an empty list contains the
    query chains those not having query pattern
    $p_i$
step 5. $\overset{n}{\underset{j=1}{\forall}} \{tlq(p_i) \leftarrow q_j \exists p_i \not\subset q_j \wedge q_j \in Q\}$ //
    moving all query chains those not having
    query pattern $p_i$
step 6. End //of step 1
```

The notations used in the algorithm are as follows:

```
step 1. $s=2$ // represents the size of the query-
    patterns to be discovered in sequence, which
    is initialized by 2. Since the algorithm uses
    the query-patterns of size 1 as input.
step 2. $tlp$ // Tabu list of query-patterns:
step 3. Tabu list of query-chains $tlqc$ :
step 4. Prepare tabu list of Query-chains $tlt$ :
    query-chains that are having one or zero
    items from 1-size patterns list
step 5. Main Loop: While (True) Begin
step 6. Pick $s-1$ size patterns
step 7. $\overset{|qp_{(s-1)}|}{\underset{j=1}{\forall}} \{p_j \exists p_j \in qp_{(s-1)} \wedge p_j \notin tlp\}$ Begin //for each query pattern $p_j$ in $qp_{(s-1)}$
step 8. $\overset{|qp_{(s-1)}|}{\underset{k=1}{\forall}} \{p_k \exists p_k \in qp_{(s-1)} \wedge j \neq k \wedge p_k \notin tlp\}$ Begin //for each query pattern $p_k$ in $qp_{(s-1)}$, such that $j \neq k$
step 9. $p_{jk} = p_j \cup p_k$ // results new query pattern $p_{jk}$ that contains all unique query elements in $p_j$ and $p_k$
step 10. $o(p_{jk}) = \sum_{j=1}^{n}\{1 \exists p_{jk} \subseteq q_j \wedge q_j \in Q\}$ // find the occurrence of pattern $p_{jk}$
step 11. $if(o(p_{jk}) > 0)$ Begin
step 12. $qp_s \leftarrow p_{jk}$ //move pattern $p_{jk}$ to the query pattern set $qp_s$
step 13. $\overset{n}{\underset{j=1}{\forall}}\{tlq(p_{jk}) \leftarrow q_j \exists p_{jk} \not\subset q_j \wedge q_j \in Q\}$ find all tabu query chains related to pattern $p_{jk}$
step 14. End // of step 11
step 15. Else $tlp \leftarrow p_{jk}$ // move pattern $p_{jk}$ to tabu pattern list $tlp$, since the pattern does not exists in an query chain
step 16. End // of step 8
step 17. End // of step 7
step 18. $if(|qp_s| > 0)$ begin //If $qp_s$ is not empty
step 19. Update tabu list of Query-chains: query-chains that are having one or zero patterns from $qp_s$ list
step 20. $qps \leftarrow qp_s$ // moving all patterns from $qp_s$ to $qps$
step 21. $s = s+1$
step 22. End //of step 19
step 23. Else break the main loop in step 5
step 24. End // of step 5
```

The process explored in step 1 to step 24, discovers all possible patterns as set qps and their respective occurrence count. Further, this set qps used as input to allocate discrete ranks to all patterns about occurrence count, max search space required and access cost required respectively. The discrete rank allocation for each pattern explored in following sections.

3.2 Search Space Utilization

To track the optimality of a query formed in to a bushy tree, two metrics "minimal access cost" and "search of minimal combinations" were defined in the earlier contribution [37]. As both the metrics are fundamental for estimating the optimality of a query pattern, the search space is used is proportionate to the varied combinations traversed because of tow query elements connected under a join node pattern. Hence, the metric search of minimal combinations supports to assess the search space utilization. About this observation, the metric "search of minimal combinations" [37] adapted and the result obtained for this metric from each query pattern considered as search space utilization of the respective query pattern. The process of the estimating this metric of a given query pattern is as follows:

Search of minimal combinations could be defined as the cumulative outcome of combinations amidst the subjects and count of objects in a chosen predicate. Estimation of max possible combination are carried out as:

$$mpc(p_k) = \sum_{i=1}^{|S|}\sum_{j=1}^{|O|}\{1 \exists p(s_i \rightarrow o_j) \equiv P\}$$

// in the equation above, the counting of the possible combinations for a subject s_i and object o_j with anticipated predicate P, $p(s_i \rightarrow o_j)$ is concrete predicate of the subject s_i and object o_j. "$mpc(J_k)$" denotes the max possible combinations for a query pattern p_k //.

3.3 Query Pattern Access cost

Other significant factor is the access time which is unique for all query elements in a query pattern which targets the single RDF store. However, if the triple stores that are targeted are differing from the every query element in a chosen query pattern constituted, accordingly the access time might vary for varied query elements. In order to address the constraint, the other significant metric "minimal cost" is considered in the earlier solution [37], which is also adapted in the second model. The metric is very much reliant on the metric "search by minimal combinations”. The process of evaluating the metric explored are:

Minimal access cost is the cumulative of costs for accessing the Distributed RDFs over varied subjects to attain the objects within expected predicate that is assessed using the following equation.

$$ac(p_k) = \sum_{i=1}^{|S|} \sum_{j=1}^{|DRDF|} \{ac(s_i \Rightarrow rdf_j) \otimes vc\}$$

// in the equation above, |S| is the cumulative number of subjects over a given query pattern,

$|DRDF|$ Is the number of RDFs measured under distributed architecture, $ac(s_i \Rightarrow rdf_j)$ shall be access cost to rdf_j under the selected subject s_i and vc is indicating the number of visits. The notation $ac(p_k)$ indicates the minimal access cost of the query pattern p_k.

3.4 Ranking by their occurrence count

Clone qps as set $oqps$ and refine it as if any pattern p_i is subset of other pattern p_j with same occurrence count, then discard p_i from $oqps$. The model adapted to prune the patterns as follow:

```
step 1: $\overset{|oqps|}{\underset{i=1}{\forall}} \{p_i \exists p_i \in oqps\}$ Begin // for each
    pattern $p_i$ that exists in $opqs$
step 2: $\overset{|oqps|}{\underset{j=i+1}{\forall}} \{p_j \exists p_j \in oqps \wedge p_i \neq p_j\}$ Begin // for
    each pattern $p_j$ of set $opqs$, which is not
    equal to pattern $p_i$
step 3: $if\left(p_i \subset p_j \wedge o(p_i) \equiv o(p_i)\right)$ Begin // if
    pattern $p_i$ is subset of $p_j$ and the occurrence
    count of both patterns is identical
step 4: $opqs \setminus p_i$ //prune the pattern $p_i$ from set
    $opqs$
step 5: $i = i - 1$ // decrementing the index of the
    loop in step 1
step 6: Go to step 1;
step 7: End // of step 3
step 8: End // of step 2
step 9: End // of step 1
```

Rank the query patterns in $oqps$ in ascending order of their respective occurrence count, such that patterns having same occurrence count will have same rank and pattern with highest occurrence count entitles highest rank. The process of rank allocation as follows:

```
step 10: $\overset{|oqps|}{\underset{i=1}{\forall}} \{p_i \exists p_i \in oqps\}$ Begin // for each
    pattern $p_i$ that exists in $opqs$
step 11: $\overset{|oqps|}{\underset{j=i+1}{\forall}} \{p_j \exists p_j \in oqps \wedge p_i \neq p_j\}$ Begin // for
    each pattern $p_j$ of set $opqs$, which is not
    equal to pattern $p_i$
step 12: $t = j$
step 13: $\overset{|oqps|}{\underset{k=j+1}{\forall}} \{p_k \exists p_k \in oqps \wedge p_k \neq p_j\}$ Begin //
    for each pattern $p_k$ of set $opqs$, which is not
    equal to pattern $p_j$
step 14: $if\left(o(p_k) < o(p_j)\right)\ t = k$
step 15: End // of step 4
step 16: End //of step 2
step 17: $if\left(o(p_i) > o(p_k)\right)\ p_i$ □ $p_k$ // if
    occurrence count of pattern $p_i$ is greater than
    occurrence count of pattern $p_k$ swap their
    positions in $oqps$
step 18: $idx = 0$ // ranking index initialized by 0
step 19: $\overset{|oqps|}{\underset{i=1}{\forall}} \{p_i \exists p_i \in oqps\}$ Begin //
step 20: $if\left(i \neq 1 \&\& o(p_{i-1}) \equiv o(p_i)\right)\ r_o(p_i) = idx$
step 21: Else Begin
step 22: $idx+ = 1$
step 23: $r_o(p_i) = idx$
step 24: End //of step 12
```

3.5 Ranking by search space utilization

Clone qps as set $sqps$, and refine the set $sqps$, such that if any pattern p_i is subset of other pattern p_j and search space usage of p_i is equal to search space usage of p_j then discard p_i from $sqps$. Further, Rank the query patterns in $sqps$ in descending order of their respective occurrence count, such that patterns having same search space usage will have same rank and pattern with lowestsearch space usage entitles highest rank. The sorting and ranking procedure depicted as follows:

$$\overset{|sqps|}{\underset{i=1}{\forall}}\left\{\overset{|sqps|}{\underset{j=1}{\forall}}\left\{\left(sqps \setminus p_i\right)\exists\left\{\begin{matrix} p_i \subseteq p_j \wedge \\ mpc(p_i) \equiv mpc(p_j) \wedge \\ [p_i, p_j] \in sqps \end{matrix}\right\}\right\}\right\}$$

// The equation is pruning the query patterns if any of the pattern p_i is subset of the pattern p_j and the search space utilization of the both patterns $mpc(p_i)\, and\, mpc(p_j)$ are identical.

Then sort the leftover patterns of $sqps$ in descending order of their search space utilization (the similar process that explored in steps 10 to 17 in section 3.4 should apply on $sqps$ instead of $oqps$ and comparison should do about search space utilization instead of occurrence count. Further, rank them according to their order of presence in the set $sqps$, such that the distinct query patterns having same search space utilization enlists with same rank. The process explored in steps 18 to 24 is compatible to tank the query patterns in set $sqps$, which should do by comparing their search space utilization instead of occurrence count. The ranks allotted to each query pattern p_i in set $sqps$ denoted by $r_s(p_i)$.

3.6 Ranking by access cost

Clone qps as set $aqps$, and refine if any pattern p_i is subset of other pattern p_j and access cost of p_i is more or equal access cost of p_j then discard p_i. Further, rank the query patterns in $aqps$ in descending order of their respective occurrence count, such that patterns having same access cost will have same rank and pattern with lowest access cost entitles highest rank. The process that adapted to prune the query patterns in section 3.3 can merely follow to prune records from set $aqps$. The condition to prune record should compare their access cost instead occurrence count (see step 1 to 9 in section 3.4). Further, the sorting and ranking is similar to steps 10 to 24 in sec 3.3 and the query patterns should sort in descending order of their access cost instead occurrence count.

The ranks assigned to the query patterns in set $aqps$ refersas the notation $r_a(p_i)$, which denotes the rank assigned to query pattern p_i based on its access cost.

3.7 Discrete Rank Distribution Consistency Score (DRDCS)

This section adapts the mean square distance [38] between the distinct ranks assigned to each query pattern to estimate the rank distribution consistency. In order to define the DRDC for each pattern, initially identifies all unique patterns from set $oqps$, set $sqps$ and set $aqps$, which denotes further as set $uqps$. The foremost step to discover DRDCS of the patterns in $uqps$ is, if any of the query pattern p_i from set $uqps$ not exists in any of the sets $oqps, sqps\, and\, aqps$, then identify the super set of that query pattern p_j with highest rank from the respective set and assign the rank of superset query pattern p_j to the corresponding query pattern p_i. Further assess the DRDCS of the each pattern from the respective ranks of occurrence count, search space utilization and access cost those assigned to each query pattern as follows:

step 1: $\overset{|uqps|}{\underset{i=1}{\forall}}\{p_i \exists p_i \in uqps\}$ Begin / for each query pattern p_i in set $uqps$

step 2: $m_r(p_i) = \dfrac{r_o(p_i) + r_s(p_i) + r_a(p_i)}{3}$ // Finding the mean of the all distinct ranks assigned to pattern p_i

step 3: $d_r(p_i) = \dfrac{\left(\begin{matrix}\sqrt{\left(m_r(p_i) - r_o(p_i)\right)^2} + \\ \sqrt{\left(m_r(p_i) - r_o(p_i)\right)^2} + \\ \sqrt{\left(m_r(p_i) - r_o(p_i)\right)^2}\end{matrix}\right)}{3}$ /Finding the mean square distance of the distinct ranks assigned to pattern p_i

step 4: $cs(p_i) = \left(\dfrac{d_r(p_i)}{r_o(p_i)}\right)^{-1}$ // the discrete rank distribution consistence score is estimated as inverse ratio of the mean square distance against the rank assigned for occurrence count.

step 5: End// of step 1

The discrete rank distribution consistence score estimated as the ratio of mean square distance against the rank assigned to occurrence count of the respective query pattern. This is due to the ambiguity of the root mean square distance obtained for the distinct lowest ranks that approximately equal and the distinct highest ranks that also approximately equal. As an example, the rank set (3, 2, 1) of query pattern p_i and the rank set (33, 32, 31) of query pattern p_j gets the equal value 2 a mean square distance, but ratio of these mean square distances against to the rank assigned under occurrence count of the respective query patterns p_i *and* p_j are 0.666666667 and 0.060606061 which are distinct. These ratio values are lowest for highest distinct ranks and highest for lowest distinct ranks. Hence, the inverse ratio s obtained as consistency score. The consistence score observed for respective example given are 1.5 for query pattern p_i with distinct rank set (3, 2, 1)and 16.5 for query pattern p_j with distinct rank set (33, 32, 31) . Here the rank assigned for occurrence count is considered to assess the discrete rank consistence score, which is since the most occurrences indicates the associability of the respective pattern towards the multiple query chains given as input to query processing. If a query pattern is having highest rank in regard to occurrence count, the execution of respective query pattern accomplish the partial results extraction for the multiple number of query chains that represented by the occurrence count of the respective query pattern.

These query patterns scheduled further in the order of their discrete rank distribution consistence score. The scheduling of these patterns is parallel, which in the context of first come first serve scheduling strategy.

4 EXPERIMENTAL SETUP AND PERFORMANCE ANALYSIS

The proposed model DR-QPO and other contemporary models [31] [32] [33] [35] adapted to compare are implemented using java and executed on Intel 5th generation platform. The distributed environment of multiple triple stores and multiple clients is simulated using java RMI, such that clients can submit SPARQL queries in parallel. The performance statistics of the results obtained from DR-QPO and other contemporary models assessed using expression language R [39].

4.1 The Input Query Formation

Each client of the distributed environment creates synthesizes set of SPARQL query chains such that each query chain contains varied number of query patterns that extracts the results from triple stores and submits to query processing module in parallel. The set of noteworthy RDF stores such as FACTBOOK, FOAF, and LUBM [40-42] adapted as distributed triple stores for experiments. The number of query elements in each query chain is in the range of 5 to 25 and total number of query chains submitted in parallel is in between 10 to 60.

4.2 Performance Analysis

The metric "average query processing time" considered for assessing the performance of the DR-QPO and other contemporary models adapted. The "residual memory ratio", which is the percentage of the memory not in use against to total memory allocated for query processing. Along the side of these two metrics, access cost dissemination ratio, which is the ratio of access cost diffused by executing a cyclic or recursive query pattern only once. The existing contemporary models are considering these cyclic recursive patterns exist in a given single query chain, contrast to this, the DR-QPO is considering the recursive query patterns from the multiple query chains submitted in parallel.

Table 1: The Results Obtained For T-Test and Wilcoxon Test Performed On Average Query Processing Time Observed From Dr-Qpo and Other Contemporary Models

Average Query Processing Time				
	T-test		Wilcoxon Test	
	t-score	p-value	z-score	p-value
Trummer et al., [35]	-6.27786	< .00001	-3.0594	0.00111
chen et al.,[33]	-6.475	< .0000	-3.059	0.00111

	81	1	4	
w. zuo et al., [32]	-6.50351	<.00001	-3.0594	0.00111
F.M waas et al., [31]	-8.59106	<.00001	-3.0594	0.00111

Table 2: The results obtained for t-test and Wilcoxon test performed on residual memory observed from DR-QPO and other contemporary models

Residual Memory				
	T-test		Wilcoxon Test	
	t-score	p-value	z-score	p-value
Trummer et al., [35]	-6.40203	<.00001	-3.0594	0.00111
chen et al.,[33]	-9.28173	<.00001	-3.0594	0.00111
w. zuo et al., [32]	-9.35815	<.00001	-3.0594	0.00111
F.M waas et al., [31]	-13.12842	<.00001	-3.0594	0.00111

Table 3: The results obtained for t-test and Wilcoxon test performed on access cost dissemination observed from DR-QPO and other contemporary models

Access Cost Dissemination				
	T-test		Wilcoxon Test	
	t-score	p-value	z-score	p-value
Trummer et al., [35]	4.01221	0.000293	-3.0594	0.00111
chen et al.,[33]	3.77302	0.000524	-3.0594	0.00111
w. zuo et al., [32]	3.74915	0.000555	-3.0594	0.00111
F.M waas et al., [31]	3.80887	0.00048	-3.0594	0.00111

The consistency of the proposed model evinced through t–test and Wilcoxon signed rank test [43] applied on the values obtained for respective metrics of DR-QPO and other contemporary models adapted. The tables 1, 2 and 3 exploredthe t-score, z-score and respective degree of probability observed between the divergent metric values obtained from DR-QPO and other models. The degree of probability observed for t-test and Wilcoxon signed rank test between DR-QPO and other models is almost nullified (almost zero), hence it is obvious to conclude that the proposed model DR-QPO significantly outperformed all other contemporary models considered for comparison.

5 CONCLUSION

Discrete Rank based Query Pattern Optimization (DR-QPO) towards Parallel Query Planning and Execution for RDF Stores proposed here in this manuscript. Unlike the existing parallel query planning and execution models [31-33, 35] found in contemporary literature, the proposed model is optimizing the query patterns planning and execution in parallel for distributed query chains. The contemporary models limited to optimize the cyclic and recursive patterns found in individual query chain. Contrast to this, the DR-QPO optimizing the query patterns those are recursive in multiple query chains submitted in distributed environment. The metrics such as query pattern occurrence count (pattern exists in multiple query chains), search space utilization and access cost are consider to assign distinct ranks to each query pattern. Further, estimates the discrete rank distribution consistence score (DRDCS) of each query pattern, which is critical contribution of the manuscript. Further, query patterns planned to execute according to their DRDC observed. The results obtained from experimental study evincing that the proposed model consistently outperformed the other contemporary models. The performance analysis of the model proposed and other contemporary models done, which is by comparing the results obtained suing ANOVA standards like t-test and Wilcoxon signed rank test. The scope of this manuscript is optimizing the query patterns discovered from distributed query chains given as input to parallel query processing against diversified triple stores. Hence, in this regard the exploration limited to the model devised to optimize the order of query patterns in regard to discrete ranks assigned under divergent factors such as the pattern associability to query chains (occurrence count), search by minimal combinations (search space utilization) and minimal access cost. The future research can contribute an optimal scheduling strategy to schedule the query patterns to execute in parallel.

REFERENCES

[1] P. Hayes, "RDF Semantics," W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-mt/.
[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "DBpedia: A Nucleus for a Web of Open Data," in ISWC, 2007.
[3] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A Core of Semantic Knowledge," in WWW, 2007.
[4] D. Huynh, S. Mazzocchi, and D. R. Karger, "Piggy Bank: Experience the Semantic Web inside your web browser," J. Web Sem., vol. 5, no. 1, 2007.
[5] E. I. Chong, S. Das, G. Eadon, and J. Srinivasan, "An Efficient SQL based RDF Querying Scheme," in VLDB, 2005.
[6] D. Battr´e, F. Heine, A. H¨oing, and O. Kao, "On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT based RDF Stores," in Proceedings of the 2005/2006 international conference on Databases, information systems, and peer-to-peer computing, 2007, pp. 343–354.
[7] L. Ali, T. Janson, and G. Lausen, "3rdf: Storing and Querying RDF Data on Top of the 3nuts Overlay Network," in 10th International Workshop on Web Semantics, Toulouse, France, August 2011, pp. 257–261.
[8] M. Cai and M. Frank, "RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network," in Proceedings of the 13th international conference on World Wide Web, 2004, pp. 650–657.
[9] E. Liarou, S. Idreos, and M. Koubarakis, "Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks," in International Semantic Web Conference, 2006, pp. 399–413.
[10] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. V. Pelt, "GridVine: Building Internet-Scale Semantic Overlay Networks," in the Semantic Web – ISWC 2004, vol. 3298. Springer-Verlag, 2004, pp. 107–121.
[11] M. T. O¨ zsu and P. Valduriez, Distributed and Parallel Database Systems (3rd. ed.). Springer, 2011.
[12] L. Galarraga, K. Hose, and R. Schenkel, "Partout: A Distributed Engine for Efficient RDF Processing," Technical Report: CoRR abs/1212.5636, 2012.
[13] J. Huang, D. J. Abadi, and K. Ren, "Scalable SPARQL Querying of Large RDF Graphs," PVLDB, vol. 4, no. 11, 2011.
[14] K. Lee and L. Liu, "Scaling Queries over Big RDF Graphs with Semantic Hash Partitioning," PVLDB, vol. 6, no. 14, Sep. 2013.
[15] K. Hose and R. Schenkel, "WARP: Workload-Aware Replication and Partitioning for RDF," in DESWEB, 2013.
[16] N. Papailiou, I. Konstantinou, D. Tsoumakos, P. Karras, and N. Koziris, "H2RDF+: High-performance Distributed Joins over Large-scale RDF Graphs," in IEEE BigData, 2013.
[17] M. Husain, J. McGlothlin, M. M. Masud, L. Khan, and B. M. Thuraisingham, "Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing," IEEE TKDE, vol. 23, no. 9, Sep. 2011.
[18] S. Chatterji and S. Evani. On the complexity of approximate query optimization. In PODS, pages 282–292, 2002.
[19] O. Cure and G. Blin.RDF Database Systems: Triples Storage and SPARQL Query Processing. 2014.
[20] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. E. Ioannidis. Schedule Optimization for Data Processing Flows on the Cloud. In SIGMOD, 2011.
[21] S. Agarwal, A. Iyer, and A. Panda. Blink and it's done: interactive queries on very large data. In VLDB, volume 5, pages 1902–1905, 2012.
[22] I. Trummer and C. Koch. Approximation schemes for many-objective query optimization. In SIGMOD, pages 1299–1310, 2014.
[23] I. Trummer and C. Koch. An incremental anytime algorithm for multi-objective queries optimization. In SIGMOD, pages 1941–1953, 2015.

[24] I. Trummer and C. Koch.Multi-objective parametric query optimization. VLDB, 8(3):221–232, 2015.

[25] C. Chekuri, W. Hasan, and R. Motwani. Scheduling Problems in Parallel Query Optimization. In PODS, pages 255–265, 1995.

[26] R. Mietz, S. Groppe, O. Kleine, D. Bimschas, S. Fischer, K. Roomer, and D. Pfisterer, “A p2p semantic query framework for the internet of things,” Praxis der Informationsverarbeitung und Kommunikation, vol. 36, no. 2, pp. 73–79, 2013.

[27] L. Ali, T. Janson, G. Lausen, and C. Schindelhauer, “Effects of Network Structure Improvement on Distributed RDF Querying,” in 6th International Conference on Data Management in Cloud, Grid and P2P Systems (Globe 2013), Prague, Czech Republic, September 2013.

[28] A. Harth and S. Decker, “Optimized index structures for querying rdf from the web,” Web Congress, Latin American, vol. 0, pp. 71–80, 2005.

[29] W.-S.Han, W. Kwak, J. Lee, G. M. Lohman, and V. Markl. Parallelizing query optimization. VLDB, 1(1):188–200, Aug. 2008.

[30] W.-S.Han and J. Lee. Dependency-aware reordering for parallelizing query optimization in multi-core CPUs. In SIGMOD, pages 45–58, 2009.

[31] F. M. Waas and J. M. Hellerstein. Parallelizing extensible query optimizers. In SIGMOD, page 871, New York, New York, USA, 2009.ACM Press.

[32] W. Zuo, Y. Chen, F.He, and K. Chen. Optimization Strategy of Top-Down Join Enumeration on Modern Multi-Core CPUs. Journal of Computers, 6(10):2004–2012, Oct. 2011.

[33] Y. Chen and C. Yin. Graceful Degradation for Top-Down Join Enumeration via similar sub-queries measure on Chip Multi-Processor. Applied Mathematics and Information Sciences, 941(3):935–941, 2012.

[34] M. a. Soliman, M. Petropoulos, F. Waas, S. Narayanan, K. Krikellas, R. Baldwin, L. Antova, V. Raghavan, A. El-Helw, Z. Gu, E. Shen, G. C. Caragea, C. Garcia-Alvarado, and F. Rahman. Orca: A modular query optimizer architecture for big data. In SIGMOD, pages 337–348, 2014.

[35] Trummer, Immanuel, and Christoph Koch."Parallelizing query optimization on shared-nothing architectures." Proceedings of the VLDB Endowment 9.9 (2016): 660-671.

[36] M. Stonebraker. The Case for Shared Nothing. IEEE Database Engineering Bulletin, 9(1):4–9, 1986.

[37] Shailaja, K., P. V. Kumar, and S. Durga Bhavani. "Progressive Genetic Evolutions-Based Join Cost Optimization (PGE-JCO) for Distributed RDF Chain Queries." Proceedings of the First International Conference on Computational Intelligence and Informatics. Springer Singapore, 2017.

[38] Carmines, Edward G., and Richard A. Zeller. Reliability and validity assessment. Vol. 17. Sage publications, 1979.

[39] Ihaka, Ross, and Robert Gentleman. "R: a language for data analysis and graphics." Journal of computational and graphical statistics 5.3 (1996): 299-314.

[40] The Friend of a Friend (FOAF) project. Retrieved from foaf-project.org: http://www.foaf-project.org/ (2000).

[41] Agency, C. I. The CIA World Factbook 2015.New Yark: Skyhorse Publishing Inc (2014).

[42] Guo, Yuanbo, Zhengxiang Pan, and Jeff Heflin. "LUBM: A benchmark for OWL knowledge base systems." Web Semantics: Science, Services and Agents on the World Wide Web 3.2 (2005): 158-182.

[43] Cleophas, Ton J., and Aeilko H. Zwinderman. "Paired Continuous Data (Paired T-Test, Wilcoxon Signed Rank Test)." Clinical Data Analysis on a Pocket Calculator. Springer International Publishing,2016.31-36.