AN EFFICIENT ALGORITHM FOR DATABASE QUERY OPTIMIZATION IN CROWDSOURCING SYSTEM

Miss. Pariyarath Jesnaraj [1], Dr. K. V. Metre[2]

[1] *Department of Computer Engineering, MET's IOE, Maharashtra, India*
[2] *Department of Computer Engineering, MET's IOE, Maharashtra, India*

ABSTRACT

Query optimization is important to crowdsourcing system as it provides declarative query interface in relational database management system. In the proposed approach, a technique called crowd optimization in which the user has to submit an SQL query and the system compiles the query, generating the execution plan and evaluating that query giving an optimized plan to the user. In relational database systems, query optimization is providing query interfaces, which are important for crowdsourcing. The system considers cost and latency in query optimization and generates query plans that give a good balance between the cost and latency. Efficient algorithms for optimizing four types of queries are used i.e. selection queries, join queries, complex queries and order-by queries.

Keyword: *Crowdsourcing, query optimization, select query, join query, complex queries, order by query.*

1. INTRODUCTION

Crowdsourcing is the process of getting work, usually through online, from a crowd of people. The word is a combination of the words 'crowd' and 'outsourcing'. The idea is to take work and outsource it to a crowd of workers. Crowdsourcing has created a variety of opportunities for many challenging problems by leveraging human intelligence. Some of the applications such as image tagging, natural language processing, and semantic-based information retrieval are used in crowdsourcing. Crowdsourcing depends on human workers to complete a job, but human workers give errors, which can make the results of crowdsourcing arbitrarily bad. The reason is, first, to obtain rewards, a malicious worker can submit random answers to all questions. This can significantly degrade the quality of the results. Second, for a complex job, the worker may lack the required knowledge for handling it. Human workers in crowdsourcing solve problems based on their experience, knowledge, and perception. It is therefore difficult to find which problems can be better solved by crowdsourcing than solving using traditional machine-based methods. Therefore, a cost sensitive quantitative analysis method is needed. Since data sets can be enormous and monetary cost and latency of data processing with humans can be large, optimizing the use of humans for finding items is an important challenge. In proposed system, a method crowd optimization is designed to hide the complexities as well as relieve the user from the burden of dealing with the crowd data. Query optimization is a function of many relational database management systems. The query optimizer determines the most efficient way to execute a given query by considering the possible query plans. In crowd optimization, the query optimizer generates evaluation plans for submitted query and system selects the best query plan considering both cost and

latency. The cost of query execution depends on I/O overload, CPU, and communication. The best method of execution depends on various conditions including how the query is written, the data size on which query fetches its results, how the data is stored, and accessing structures it uses to fetch the data. Crowd optimization mainly deals with three types of queries such as select, join and complex queries. Some crowdsourcing systems such as CrowdDB, Qurk and Deco provide the SQL interface, which is known, to the database users.

2. RELATED WORK

Meihui Zhang, Ju Fan,et al proposed CrowdOp a cost-based optimization approach for declarative crowdsourcing systems. In this an efficient algorithm within the CrowdOp for optimizing 3 types of queries such as select query, join query and complex selection-join queries is used. It also considers both cost and latency in query optimization objectives and generates query plans that provide a good balance between the cost and latency [1]. V. C. Raykar, S. Yu, et al discussed a probabilistic approach is discussed which is used for supervised learning. In this gives an estimate of the actual hidden labels and also evaluate different experts. Output indicates that the proposed method is superior to the commonly used majority voting baseline. Two key assumptions: (1) feature vector is not depended on performance of each annotator for a given instance and (2) conditional on the truth the experts are independent, that is, they make their errors independently [2]. A. Marcus, E. Wu, et al observed that items are compared for joining and sorting data which is the most common operations in DBMS. Qurk used MTurk platform to runs on top of Crowdsourcing [3].

M. J. Franklin, D. Kossmann, et al proposed different difficult functions such as ranking, matching or aggregating results based on fuzzy criteria are computationally performed by CrowdDB system. CrowdDB takes input from human with the help of crowdsourcing system for providing information that is missing from the database which cannot easily get answered by database systems or search engines. CrowdDB resembles with traditional database system with some big change. Traditional database systems do not take human input for query processing. From an implementation point of view human-oriented query operators are needed to integrate crowdsourced data. Cost as well as performance depends on a many new factors including training fatigue, worker affinity, location and motivation [4]. A. Marcus, D. R. Karger, et al observed several techniques for workers using a crowdsourcing platform like Amazon's Mechanical Turk to estimate the fraction of items in a dataset that satisfy some property or predicate without explicitly iterating through every item in the dataset. When performing a GROUP BY operation with a COUNT or AVG aggregate it is important to support predicate ordering and in query evaluation. A comparison based on sampling item labels, showing workers a collection of items and asking them to estimate how many satisfy some predicate are discussed [5].

A. G. Parameswaran, H. Park, et al proposed Deco's data model, query language for declarative Crowdsourcing system is used. Deco a database system that answers declarative queries posed over stored relational data, the collective knowledge of the crowd, as well as other external data sources. Deco appears to the end user as similar as possible to a conventional database system, while hiding many of the complexities of dealing with humans. In this, Crowdsourcing and Databases Crowdsourcing Algorithms are used that are offered practical and principled approach for accessing crowd data and also integrating it with conventional data [6]. P. Venetis, H. Garcia-Molina, et al observed parameterized families of algorithms to retrieve the maximum item and proposed strategies to tune these algorithms under various human errors and cost models are developed. Also evaluating many metrics, both analytically and via simulations, the trade-off between three quantities i.e. quality, monetary cost, and execution time [7]. J. Wang, T. Kraska, et al describes how machine only approaches often fall short on quality, while brute force people only approaches are too slow and expensive. A hybrid human-machine workflow is thus proposed to address this problem. In hybrid approach, the results indicated that (1) the two-tiered approach generated fewer cluster-based HITs than existing algorithms; (2) Hybrid human-machine workflow significantly reduced the number of HITs compared to human-based techniques, and achieved higher quality than the state-of-the-art machine based techniques; and (3) The cluster-based HITs can provide lower latency than a pair-based approach[8].

H. Park and J. Widom, et al discussed Deco provides a cost-based query optimizer, building on Deco data model, query language and query execution engine is explained. Finding the best query plan to answer a query is the objective of Deco [9]. S. B. Davidson, S. Khanna, et al explained if the elements belong to the same group or cluster and have same type, then the answer to a type question is "yes". In this the answer to a value question orders the two data elements. Here the answers returned by the crowd may be sometimes erroneous even though there is an underlying ground truth. Efficient algorithms for top-k and group-by queries, that guaranteed to achieve good results

with high probability [10]. C.-J. Ho, S. Jabbari, et al proposed a crowdsourcing market is a tool that collects data from very different workers. Worker uses label for classification of common tasks but it may be error prone, at a particular time it can be treated as spam. The solution to this problem can be obtained by collecting labels for each instance from multiple workers. With the help of online primal-dual techniques, classification tasks of task assignment and label assignment for workers can be carried out in heterogeneous way. More accurate predictions are lead by adaptively assigning worker at a lower cost when the available workers are diverse [11].

J. Fan, M. Lu, et al discussed the web consists of featured data in terms of HTML tables. If these HTML tables are integrated, it gives rise to a knowledge repository but semantic correspondences between web table columns need to be checked, it can be carried out with help of conventional schema matching but they does not produce good result or it may be sometime incomplete. Two solutions for web table matching which solves schema matching and semantic correspondences. First, concept-based approach is designed which deals with mapping of each column of web table to best concept, which solves problems for columns which are disjoint, due to incomplete values of columns. Second, a hybrid machine crowdsourcing framework which deals with incomplete column with concept matching tasks. An algorithm is used to produce the best matches for the columns, considering cost and utilizing the crowdsourcing [12]. A. D. Sharma, A. Parameswaran, et al proposed a system named CrowdFind which deals with problem of searching some items which satisfy fixed properties within data set for people. Suppose that a person wants to identify total no of travelling photos from a travel website, since the data for this constraints may be very large, also monetary cost and latency would be large. An optimal algorithm is proposed which has comparison capacity between statistic costs versus actual time to evaluate the query. Multiplicative and additive approximations are used to design the algorithms with specific expected cost and time measures [13].

3. SYSTEM ARCHITECTURE

Fig-1 illustrates the architecture of query processing in Crowd Optimization. The user first fills the query form for the required attributes and conditions. The query generator will automatically generate the SQL query and an initial plan of that query is generated. The query optimizer takes the initial query plan as input and produces an optimized query plan. This optimizer parses the query and produces a best cost and time efficient query plan. The parser is like a traditional relational database system, takes the query from the application as an input. Here the query plan is generated based on the query asked. We generate select, join and order-by queries for the given query. Based on the chosen query type the query plan is generated. This query plan contains the well- matched query to get the required resultant value. For the selection query type, it initially read the query and executes them to fetch the appropriate result. Likewise, the remaining join and order-by queries are executed and viewed on the table. Crowdsourcing executor estimates the crowd of each query execution and stores it on a memory for calculating cost and latency. The execution cost and latency may different based on the query type are used for. Finally, it compares the best query plan for executing the given query and returns the expected result to the user.

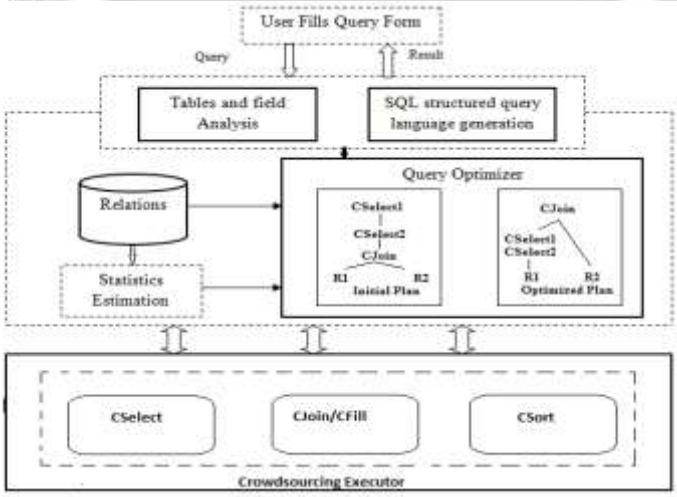

Fig -1: System Architecture

3.1 Algorithms

Algorithm for finding Optimized Query Plan (Q, C, P*)

Input: Q: A query; C: A cost budget;
Output: P*: An optimized query plan

```
1.  if C = Nil then
2.        P*← COSTOPT(Q)
3.  else
4.        Lmin ← COMPUTEMINLATENCY(Q)
5.        Lmax ← COMPUTEMAXLATENCY(Q)
6.  while Lmin ≤ Lmax do
7.        L ← (Lmin +Lmax)/2
8.        P ← LATENCYBOUNDOPT(Q,L)
9.  if P.cost ≤ C then
10.       P*← P
11.       Lmax ← L-1
12.   else Lmin ←L+1
13. Return *P
```

Algorithm for generating Select query plan(C, L, P)

Input: C: Selection conditions; L: Latency constraint
Output: P: A query plan

```
1.  Sort C in increasing order of selectivity ;
2.  /* Initializing Groups G(i; j) */
3.  for (i = 1…|C|)do
4.        G(i,1) ← cost(i, |C|)
5.              for (j = 2 ... L)do   G(i, j) ← +∞
6.  /* Iteratively computing G(i, j)*/
7.                  for (j = 2 ... L)
8.                     G(|C|, j) ← cost(|C|,|C|)
9.                     for (i = 1 ... |C| − 1) do
10.                       G(i,j) ← min^{|C|−1}_{k=i}{cost(i,k) + G(k + 1,j − 1)
11.                       p[i][j] ← k* with the minimum cost
12. Generate query plan P w.r.t G (1, L)from p[.][.]
13. return P;
```

Algorithm for generating Partition tree for Join query plan (T1 ,T2, A^F, C^J, L, N)

Input: T 1; T 2: tuple sets to join; A^F: CFILL attributes; C^J: join conditions; L: latency constraint;
Output: N: A partition tree rooted at node N

```
1.  N ← INITIALIZE (T 1; T 2)
2.  N.cost ← ESTCJOINCOST (N, C^J)
3.  if L = 1 then return N
4.  for each attribute A ϵ A^F do
5.        cost (A) ← ESTCFILLCOST (N, A)
6.  for each value v ϵ Dom (A) do
7.        T'1← ESTCARD (T1, A, v)
8.        T'2← ESTCARD (T2, A, v)
9.        Nc ← GENPARTREE (T'1, T'2, A^F− {A}, C^J, L -1)
```

```
10.       cost (A) ← cost (A) + Nc.cost
11.       SETNODE (A, v, Nc)
12.  A*← the attribute with minimum cost (A)
13.       if cost (A*) < N.cost then
14.          N ← cost cost (A*)
15.          for each value v ϵ Dom (A*) do
16.          Nc ← GETNODE (A*, v)
17.   ADDCHILD (N, Nc)
18.  return N
```

3.2 Mathematical Model

Let S be the crowd optimization system, which can be represented in terms of input, functions and output.

S = {I, F, O}

I: Input: [I1, I2]

I1= Query Type

I2= Number of attributes

F: Functions: [F1, F2, F3, F4, F5, F6, F7, F8]

F1= Defines user who wants to find best query plan

F2= Generate parallel plan

F3= Generate Sequential plan

F4= Perform Selection query optimization

F5= Perform join query optimization

F6= Perform complex query optimization

F7= Perform sort query optimization

F8= Statistic Estimation

O: Output: [O1, O2, O3, O4, O5]

O1= Optimized Query Plan

O2= Query Result

O3= Cost (Time, Performance)

O4= Latency

O5= Selectivity

4. EXPERIMENTAL SETUP

A desktop application for finding best query plan is developed. The system is implemented on Windows 7 platform. Dot Net environment is used with C sharp is used for the front end and SQL Server for the backend. Dot Net Framework 4.0 and Core i3 machine with 2 GB RAM is used for development and testing. Dataset: Auto dataset contains specifications of Vehicle details, images, and their reviews.

5. RESULT AND ANALYSIS

This section examines the effectiveness of the proposed crowd optimization for the select, join, complex queries and order by queries. There are two plans for crowd optimization i.e. parallel plan and sequential plan. In our proposed system we use sequential plan because sequential plan has less execution cost compared to parallel plan. Among several plan after execution the plan, which has the lowest cost, is considered as a best query plan. In graph, X-axis represents cost and latency value and Y-axis represents the number of plans. The chart 2 show the result for a select query optimization where a number of plans is created by the system, based on the attribute given by the user. Chart 3 shows the probability for select query. Probability means a tuple in the relation satisfies a condition in a query.

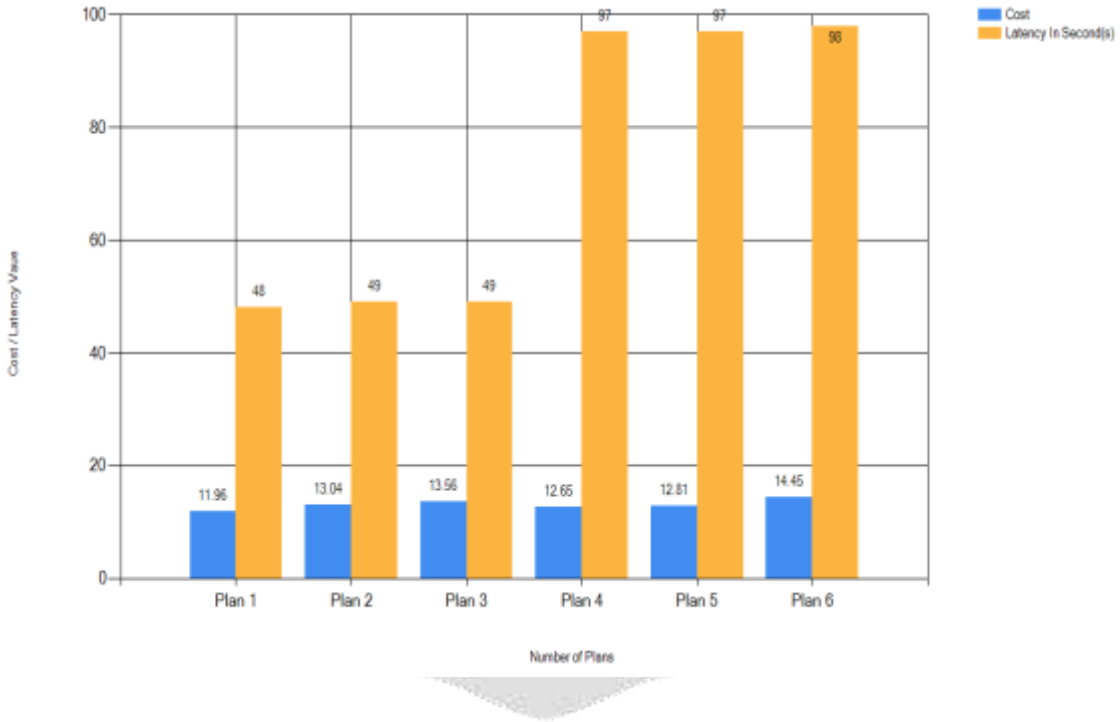

Chart -2: Graph for cost and latency for select query

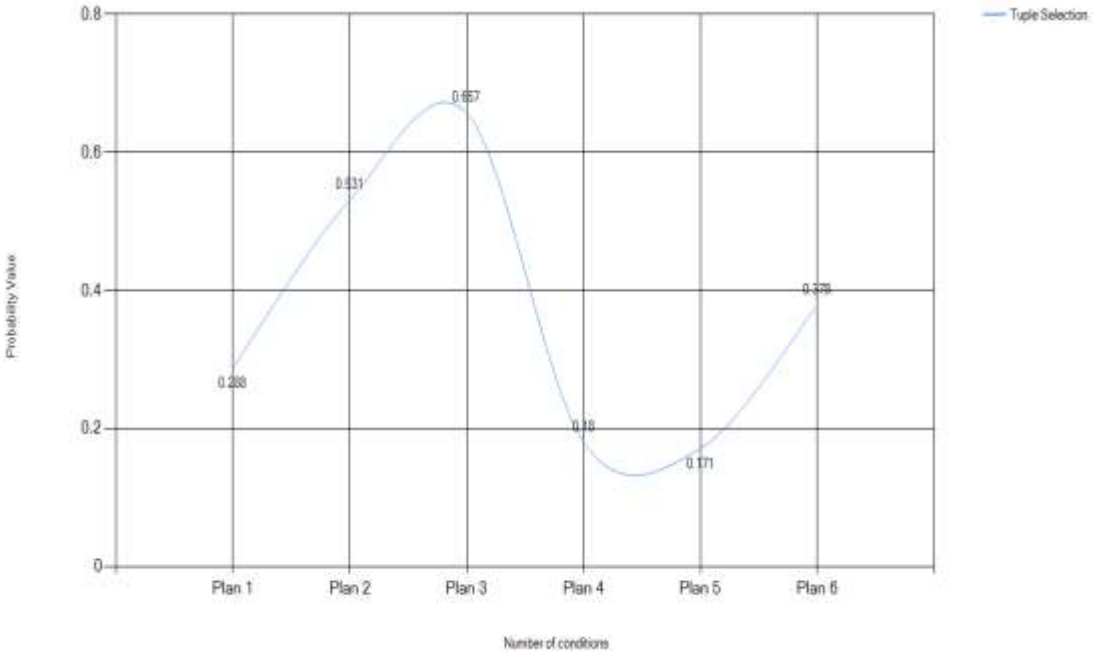

Chart -3: Graph Probability graph for Select query optimization

The Chart 4 shows the result for a Join query optimization where a number of plans is generated by the system, based on the attribute given by the user. In join query the Where clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables and join filter is used to filter only specified attributes. Chart 5 shows the probability for join query optimization.

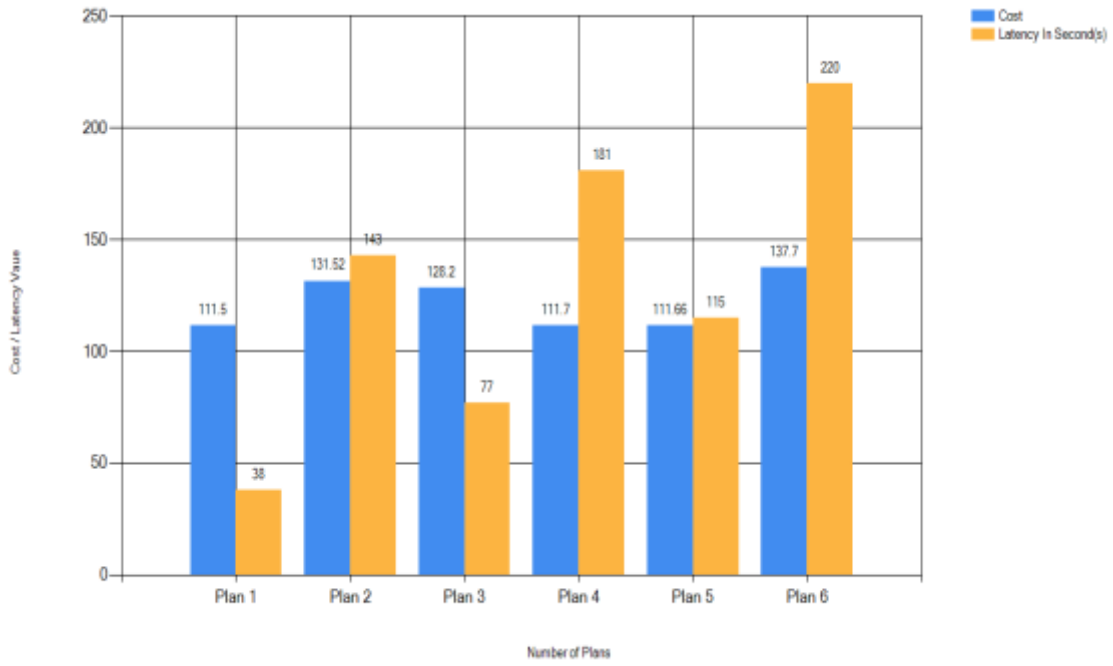

Chart -4: Graph for cost and latency for join query

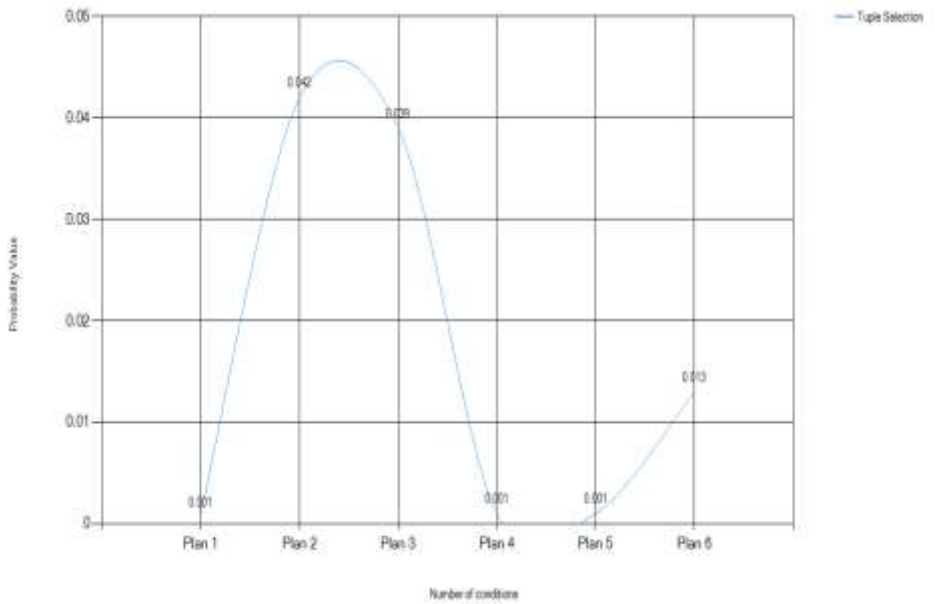

Chart -5: Probability graph for Join query optimization

The chart 6 shows the result for a Join query optimization where a number of plans is created by the system, based on the attribute given by the user. In order by query user can sort the data in ascending and descending order. After the execution of query number of query plan is generated. The plan which has lowest cost is considered as best plan. Chart 7 shows the probability for join query

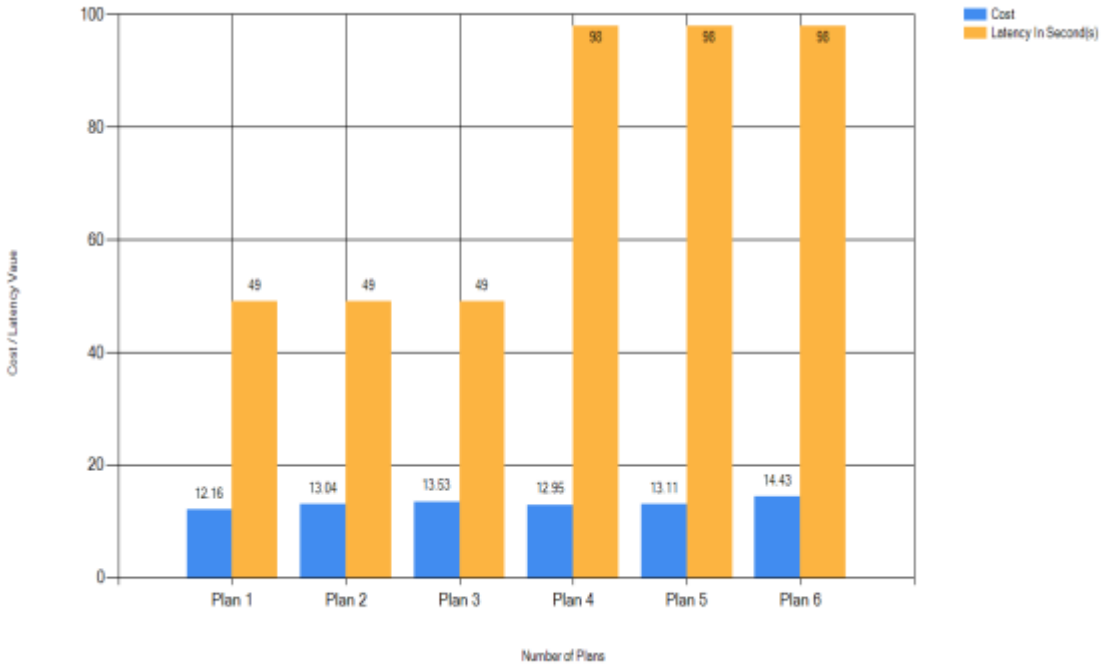

Chart -6: Graph for cost and latency for order by query

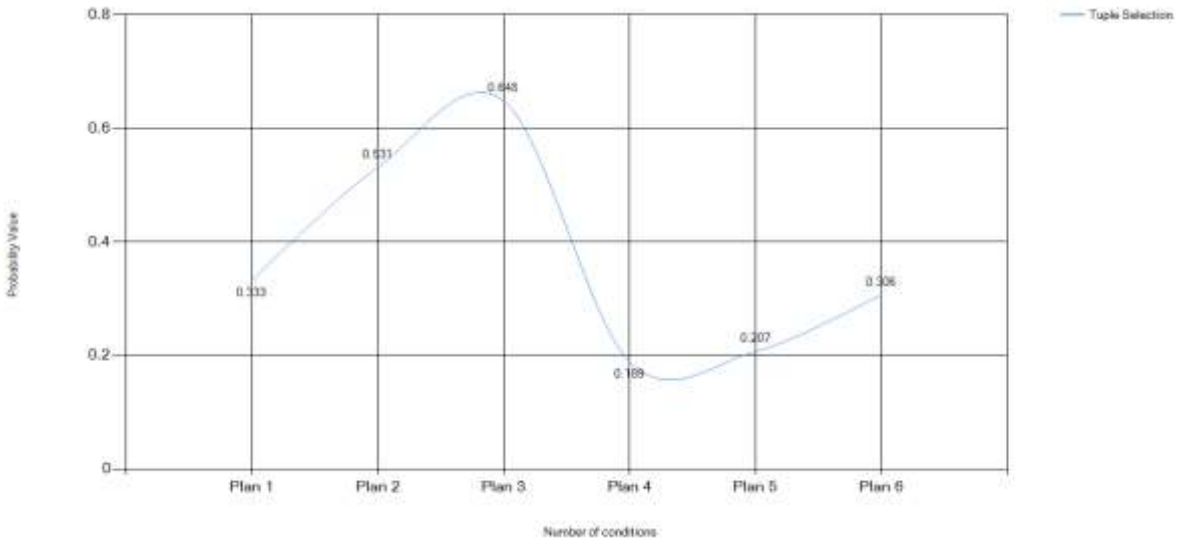

Chart -7: Probability graph for order by query optimization

6. CONCLUSIONS

Crowd Optimization is a cost-based query optimization that considers the cost-latency trade-off and supports multiple crowdsourcing operators. This system can be used for answering queries posed over stored relational data together with data obtained on demand from the crowd. SQL query is being generated by the system which is being optimized on cost constraint depending on which execution plan is created, an output is produced. Crowd Optimization can be extended to support aggregation function like sum, average and count, and group by clause. The technique will be mostly used on grouped data on the query result.

6. REFERENCES

[1]. Ju Fan, Meihui Zhang, Stanley Kok, Meiyu Lu, and Beng Chin Ooi. CrowdOp: Query Optimization for Declarative Crowdsourcing Systems, in knowledge and data engineering, vol. 27, no.8, august 2015.

[2]. V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy, Learning from crowds, J. Mach. Learn. Res., vol. 11, pp.12971322, 2010.

[3]. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, Human powered sorts and joins, Proc. VLDB Endowment, vol. 5, no. 1, pp. 1324, 2011.

[4]. M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, CrowdDB: Answering queries with crowdsourcing, in Proc.ACM SIGMOD Int. Conf. Manage. Data, 2011, pp. 6172.

[5]. Marcus, D. R. Karger, S. Madden, R. Miller, and S. Oh, Counting with the crowd, in Proc. VLDB Endowment, vol. 6, no. 2, pp. 109120, 2012.

[6]. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom, Deco: Declarative crowdsourcing, in Proc. 21st ACM Int. Conf. Inf. Knowl. Manage. 2012, pp. 12031212.

[7]. P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis, Max algorithms in crowdsourcing environments, in Proc. 21st Int. Conf. World Wide Web, 2012, pp. 989998.

[8]. J. Wang, T. Kraska, M. J. Franklin, and J. Feng, Crowder: Crowdsourcing entity resolution, Proc. VLDB Endowment, vol. 5, no. 11, pp. 14831494, 2012.

[9]. H. Park and J. Widom, Query optimization over crowdsourced data, Proc. VLDB Endowment, vol. 6, no. 10, pp. 781792, 2013.
[10]. S. B. Davidson, S. Khanna, T. Milo, and S. Roy, Using the crowd for top- k and group-by queries, in Proc. 16th Int. Conf. Database Theory, 2013, pp. 225236.