Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 9, September 2013, pg.212 – 219

RESEARCH ARTICLE

Continuous Aggregation Queries Based on Clustering Based Penalty Adaptive Query Planning

T.Keerthana[1], T.selvakannan[2]

[1]Research Scholar, Department of Computer Science, Vivekanandha College, Elayampalayam, Tiruchengode, Tamil Nadu, India

[2]Assistant Professor, Department of Computer Science, Vivekanandha College, Elayampalayam, Tiruchengode, Tamil Nadu, India

[1] keerthanatamilarasan@gmail.com; [2] mrsellu@gmail.com

ABSTRACT: - The passive web pages can transform into active environment by the continuous queries are persistent queries by providing the time varying dynamic query results useful for online decision making. To handle a large number of users with diverse interests a continuous query system must be capable of supporting server push style of Internet-based communication. A network of data aggregators has prior approaches for the scalable handling of push based data dissemination. Their implementation required Greedy Heuristics Algorithm along with pre configured incoherency bounds to manage both multiple aggregators and multiple clients for supporting server push based communications. The sub-optimal solutions are explored by existing heuristic-based approaches can only explore a limited solution space. So we propose to use an adaptive and cost-based approach. In a network of data aggregators, each dedicated and judiciously chosen aggregator serves a set of data items at specific coherencies. By our approach we can decompose a client query into sub-queries and executing sub-queries using aggregators with their individual sub-query incoherency bounds. Our cost model takes into account both the processing cost and the communication cost unlike prior approaches. Clustering based penalty Adaptive Query Planning has better performance in terms of both processing and communication cost.

Keywords: - Clustering algorithm; aggregation; queries; process message

I. INTRODUCTION

The general application such as auctions, personal portfolio valuations for financial decisions, sensors-based monitoring, route planning based on traffic information, etc., make extensive use of dynamic data. The passive web pages can transform into active environment by the continuous queries are persistent queries by providing the time varying dynamic query results useful for online decision making. They need to be able to support millions of queries due to the scale of the Internet. A continuous query system must be capable of supporting a large number of triggers expressed as complex queries against resident data storages in order to handle a large number of users with diverse interests. Many data intensive applications delivered over the Web suffer from performance and scalability issues. There is significant interest in systems that can efficiently deliver the relevant updates automatically. To know value of portfolio for a client; or the AVG of temperatures sensed by a set of sensors. In these continuous query applications, users are likely to tolerate some inaccuracy in the results. To support continuous queries for the users the system is maintained and managed by a multiple resource builders using network aggregators at a time. The exact data values at the corresponding data sources need not be reported as long as the query results Satisfy user specified accuracy requirements. Prior Approaches use Greedy Heuristics Algorithm along with pre configured incoherency bounds to manage both multiple aggregators and multiple clients for supporting server push based communications. Adaptive and cost-based approach implementation involves

- Adaptation Attempt(to check for feasibility)
- Greedy Heuristics
- Simulated Annealing
- Process Message

In these continuous query applications, users are likely to tolerate some inaccuracy in the results. The exact data values at the corresponding data sources need not be reported as long as the query results satisfy user specified accuracy requirements. Data Incoherency: The absolute difference in value of the data item at the data source and the value known to a client of the data. Let vi(t) denote the value of the ith data item at the data source at time t. The value the data item known to the client be ui(t). But the successive incoherence at the client is given by | vi(t) - ui(t) |. As soon as a data incoherency exceeds C the data refresh message is sent to the client for the data item. i.e., | vi(t) - ui(t) |> C.

Network of Data Aggregators (DA):

Using push- or pull-based mechanisms the data refresh from data sources to clients can be done. In the pull-based mechanism data sources send messages to the client only when the client makes a request where as in the push-based mechanism data sources send update messages to clients on their own. For the data transfer between the source and the client we refer push based mechanism. For scalable handling of push based data dissemination, network of data aggregators are proposed as that dissemination tree from sensor nodes to root already exists; and they also install error filters on partial aggregates. Data refreshes occur from data sources to the clients through one or more data aggregators. We assume that each data aggregator maintains its configured incoherency bounds for various data items. In data dissemination a hierarchical network of data aggregators is employed such that each data aggregator serves the data item at some guaranteed incoherency bound. The data dissemination capability point of view, each data aggregator is characterized by a set of (di, ci) pairs. Where di=data item ci=incoherency bound

The configured incoherency bound of a data item at a Data aggregator can be maintained using any of following methods:

- the data source refreshes the data value of the DA whenever DA's incoherency bound is about to get violated. This method has scalability problems.
- Data aggregator(s) with tighter incoherency

bound help the DA to maintain its incoherency bound in a scalable manner. Consider in the network the data aggregators managing the data items x1-x4, various aggregators can be characterized as

A1 :={(x1,0.5),(x3,0.2)}

A2 :={(x1,1.0), (x2,0.1), (x4,0.2)}

Aggregator A1 can serve values of the x1 with an incoherency bound greater than or equal to the 0.5 where as the A2 can disseminate the same data item at a looser incoherency bound of 1.0 or more.

A.Query aggrigation and their execution

To execute in incoherency bounded continuous queryPlan is required. We present a technique for executing multi data aggregation queries. The theme of our scheme is to reduce the number of refresh messages from data aggregator to client. For the better understanding follow the scenario. Scenario 1: Assume the query Q=60X1+190X2+150X3 Where X1, X2, X3 are dataItems for stock with incoherency bound of $75.For the considered scenario the client can get the results as

a. Among data items Client can get data items separately on query incoherency bound is divided.
b. The query a single data aggregator can distribute to all data items to answer.
c. A single query can be divided into number of sub queries and only one data aggregator gives their values.

II. RELATED WORK

These new functionalities go way beyond the simple object finder services that are represented by either simple range or nearest-neighbor queries. In this paper, we focus on one of these new functionalities, in particular, the continuous aggregate k nearest- neighbor queries. While traditional nearest-neighbor queries aim to find the k-nearest objects to only one certain point, the aggregate k-nearest-neighbor queries aim to find the k-nearest objects to a set of multiple points. In that sense, the query answer to the aggregate k-nearest-neighbor query needs to be defined through an aggregate to define how an object is considered close-by to the set of multiple points. In general, the aggregate k-nearest-neighbor query problem can be formulated as follows: Given two sets of data points O and L, find the closest k points from O to all points in L based on a certain aggregate function f. Figure 1 illustrates the use of three different aggregate functions that may be used to define the query answer. In this figure, L1, L2, and, L3 are the set of multiple points L that we need to compute their aggregate k-nearest objects among the set of points O that includes objects O1 to O15. Figure 1a–c represent the cases where the aggregate function f is sum, max, and min, respectively. For simplicity, we consider k = 1. In this case, Fig. 1a considers the sum nearest neighbor query where O14 is the object whose sum of distances to L1, L2, and, L3 is minimal. On the other side, Fig. 1b considers the case of maximum nearest-neighbor query where O9 is the object whose maximum distance to any of
L1, L2, and, L3 is minimal. Finally, Fig. 1c depicts the case of minimum nearest neighbor query where O6 is the object whose minimum distance to any of L1, L2, and L3 is minimal.

The term “aggregate nearest-neighbor query” is coined in the literature in the context of spatial databases [29, 36] to refer to the case of several parties that look for a meeting point while minimizing a certain aggregate function with respect to all. of continuous k-aggregate nearest-neighbor queries (CANN) where we aim to continuously find the closest k moving objects from the set O to a set of stationary points L—the landmarks. The set of moving objects O are continually changing their locations. We aim to provide an incremental algorithm in which we maintain an initial answer with slight overhead rather than reevaluating the query with each movement update. Our problem setting has several real life applications. For example, consider a franchise with L locations that wants to send k e-coupons every few minutes to a set of close-by customers among all customers O who are driving their vehicles. It is of interest to the franchise to carefully select these k customers as the current close-by customers to the L locations. The close-by customers can be represented as either sum, minimum, or maximum distance. As an illustration, consider that L1, L2, and L3 in Fig. 1 are the locations of a certain franchise that aims to send one
E-coupon. The coupon can be sent to that customer with minimum sum, max, or, min distance to the three franchise locations. It is up to the franchise to choose any of the three aggregate functions as any of them will make sense in terms of choosing the best k customers. Thus, we aim to incorporate the three aggregate functions in our approach.

Our proposed algorithms for the continuous k-aggregate nearest-neighbor queries rely on combining the aggregate computation with the sorting and top-k selection in a way that limits the expensive aggregate computations and sorting to only few objects that have the potential to be part of the answer, i.e., top-k. In this context, we propose two algorithms, namely, H-CANN and P-CANN. The first algorithm is a Data clustering

algorithm; it takes into account all landmarks when it decides whether or not to prune an object from being part of the answer. On the other hand, PCANN is a best-effort progressive algorithm that eagerly prunes moving objects, even during the computation of the aggregate distance of the moving objects to the landmarks. P-CANN exploits some interesting geometric properties of the aggregate distance functions to assign a threshold to each landmark. These assignments impose an optimization problem, which is to have an interesting landmark order. We give several heuristics to retrieve the best order, and show through experiments the best policy to use for each aggregate function. P-CANN is a "best-effort" algorithm, it might only produce the k_ aggregate nearest neighbors, where k_ < k. In the Experimental section, we show that, on the average, k_ ≈ 0.95k. None of the proposed algorithms need to recompute the query answer to maintain the query answer continuously. In fact, all the proposed algorithms update the query answer on the fly whenever a location update is made available from the underlying Infrastructure. Parties in the Euclidean space [29] and road networks [36]. Both proposed solutions in [29], 36] focus only on the case of snapshot queries, i.e., they do not maintain the result in case of location updates. In this paper, we overcome the limitations and overhead of re-evaluating previous snapshot-based approaches upon location updates. Indeed, we consider the problem.

III. CLUSTERING BASED PENALTY ADAPTIVE QUERY PLANNING

A. Data clustering algorithm

Providing an integrated access to multiple heterogeneous sources is a challenging issue in global information systems for cooperation and interoperability. In this context, two fundamental problems arise. First, how to determine if the sources contain semantically related information, that is, information related to the same or similar real-world concept(s). Second, how to handle semantic heterogeneity to support integration and uniform query interfaces. Complicating factors with respect to conventional view integration techniques are related to the fact that the sources to be integrated already exist and that semantic heterogeneity occurs on the large-scale, involving terminology, structure, and context of the involved sources, with respect to geographical, organizational, and functional aspects related to information use. Moreover, to meet the requirements of global, Internet-based information systems, it is important that tools developed for supporting these activities are semi-automatic and scalable as much as possible. The goal of this paper is to describe the MOMIS [4, 5] (Mediator environment for Multiple Information Sources) approach to the integration and query of multiple, heterogeneous information sources, containing structured and semi structured data. MOMIS has been conceived as a joint collaboration between University of Milano and Modena in the framework of the INTERDATA national research project, aiming at providing methods and tools for data management in Internet-based information systems. Like other integration projects [1, 10, 14], MOMIS follows a "semantic approach" to information integration based on the conceptual schema, or metadata, of the information sources, and on the following architectural elements: i) a common object-oriented data model, defined according to the ODLI3 language, to describe source schemas for integration purposes. The data model and ODLI3 have been defined in MOMIS as subset of the ODMG-93 ones, following the proposal for a standard mediator language developed by the I3/POB working group [7]. In addition, ODLI3 introduces new constructors to support the semantic integration process [4, 5]; ii) one or more wrappers, to translate schema descriptions into the common ODLI3 representation; iii) a mediator and a query-processing component, based on two pre-existing tools, namely ARTEMIS [8] and ODB-Tools [3] (available on Internet at http://sparc20.dsi.unimo.it/), to provide an I3 architecture for integration and query optimization. In this paper, we focus on capturing and reasoning about semantic aspects of schema descriptions of heterogeneous information sources for supporting integration and query optimization. Both semi structured and structured data sources are taken into account [5]. A Common Thesaurus is constructed, which has the role of a shared ontology for the information sources. The Common Thesaurus is built by analyzing ODLI3 descriptions of the sources, by exploiting the Description Logics OLCD (Object Language with Complements allowing Descriptive cycles) [2, 6], derived from KL-ONE family [17]. The knowledge in the Common Thesaurus is then exploited for the identification of semantically related information in ODLI3 descriptions of different sources and for their integration at the global level. Mapping rules and integrity constraints are defined at the global level to express the relationships holding between the integrated description and the sources descriptions. ODB-Tools, supporting OLCD and description logic inference techniques, allows

the analysis of sources descriptions for generating a consistent Common Thesaurus and provides support for semantic optimization of queries at the global level, based on defined mapping rules and integrity constraints.

B. Penalties Adaptive Query Planning

Motivation Traditional query processing models may not be applicable to most data integration applications for several reasons: the absence of statistics, unpredictable data arrival rates, and redundancy among sources. We discussed the first two reasons in the previous chapter. The third one, redundancy among sources, may cause the query processor to waste time in processing duplicate tuples from duplicate sources. The Tukwila-99 [11] system aims to address these challenges by adopting an adaptive query processing model, and by incorporating run-time information into the decision engine. Architecture Figure 2.1 features the high-level system architecture of the Tukwila-99 [11] system. The query optimizer and the execution engine are interleaved to answer queries. Users pose queries to the system, and they go through a Query Reformulation" phase to be reformulated into queries
over data sources. This is because user queries may be formulated over virtual mediated schemas, and they need to be reformulated over source schemas. The job of the query optimizer is to transform the reformulated query into a physical query execution plan for the execution engine. The optimizer in Tukwila-99 is able to create partial plans if data statistics are incomplete, and it produces rules to define adaptive behavior. The execution engine processes the query plans that are produced by the optimizer. It includes an event handler for dynamically interpreting rules generated by the optimizer and supports a few re-planning techniques when adaptation is triggered. Finally, the query execution engine communicates with data sources through a set of wrappers. Wrappers are responsible for transforming the data from the format used in sources to the format used in the Tukwila-99 system. Main techniques Tukwila-99's adaptive techniques are mostly plan-partitioning based. That is, the re-optimization or re-scheduling of plans can only occur at the end of fragments, e.g., the end of pipelined units or blocking operators, where all the source data must be processed together.
The intermediate results, which are output by the fragments, must be materialized before being processed by the re-optimized portion of the plan. The optimizer must decide how many fragments to complete, balancing the potential performance penalty of materialization versus the potential benefit of being able to adapt the plan if the original plan is poor. Since there is very little information to go by, this is by necessity heuristics-driven. We will later discuss the other two systems' data-partitioning techniques, in which different portions of data can be processed by different plans in parallel, which are not necessarily materialized at intermediate points.

The Tukwila-99 system exploits a rule-based framework that is built into the core of the adaptation decision engine. Generally, the rules in Tukwila-99 are produced by the query optimizer and interpreted and executed by the query execution engine. These rules have the form \WHEN event IF conditions THEN actions". They usually specify when and how to modify the run-time behavior of certain operators and which conditions to check in order to detect opportunities for re-optimization. For example, a rule can be written as follows.

WHEN closed(join)
IF monitored-card(join) > 2 * estimated-card(join)
THEN reoptimize

This rule means that when a join operator finishes execution, if the run-time monitored cardinality of this join operator is twice as large as the estimated cardinality of the join operator, then the system chooses to trigger the re-optimization procedure at this join operator. However, this join operator must be the end of a fragment. This rule may help the system to alleviate the bad effects of inaccurate cardinality estimations. In Tukwila-99, these rules are written in procedural languages such as C/C++ to facilitate manipulation and event handling, although they resemble active rules in deductive databases [19]. Examples of events include opening or closing an operator, failing to connect to sources, running out of memory, and having processed n tuples by an operator. Once an event has triggered a set of rules, the conditions of each rule are evaluated in parallel. A condition can be a comparison of a monitored state, an estimated state or a threshold. After all conditions for a given event have been evaluated, actions are executed. Tukwila-99's actions consist of setting the overow method of a pipelined join, deactivating an operator, rescheduling the remaining operator tree, and re-optimizing

the remaining plan. As discussed before, these actions are all plan-partitioning methods. There are a few restrictions on this rule-based system to avoid common mistakes. For example, all of a rule's actions must be executed before another event is processed, and two rules that might affect each other cannot be executed in parallel.

IV. EXPERIMENT RESULTS

We review two empirical studies based on the three systems we have discussed. In the first set of studies, we examine the differences between adaptive query processing and static query processing under similar workloads and configurations. The second set of studies focuses on the performance comparison among different plan migration techniques given specific parameters. Since different experiments use different workloads and different system configurations to evaluate their performance, we excerpt a few representative studies and analyze each study individually. The experiment of Tukwila-99 is performed on a scaled version of the TPC-D 10MBdataset, and seven queries are computed over four base tables of the dataset with the exception of the Line item table. The optimizer is given correct source cardinalities, but no histograms are available; hence, the optimizer has to compute its intermediate result cardinalities based on estimations of the join selectivity's. All the joins are implemented as pipelined hash joins shows the benefits of adaptive query processing over static query processing on the Execution time. The pipelined strategy executes the query statically. The materialized strategy simply materializes the output at each join. This is even worse than the pipelined strategy in many cases. The materialized and re-planned strategy materializes the intermediate results and re-plan sat the end of each fragment whenever the cardinality of the actual value differs from its estimate by at least a factor of two. Among these three strategies, only the last strategy is an adaptive query processing strategy. From the configure, we can see that the materialized and re-planned strategy is the fastest in executing all the chosen plans, with a total speed-up of 1.42 over the pipelined strategy and 1.69over the materialize strategy.

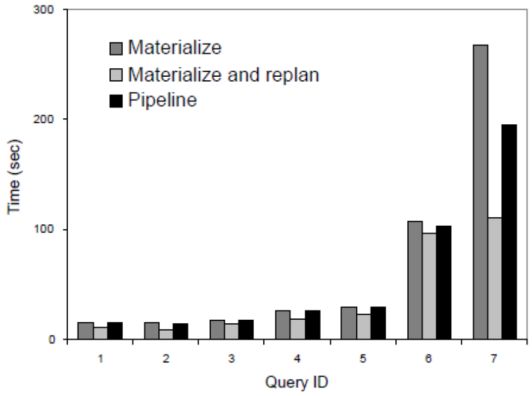
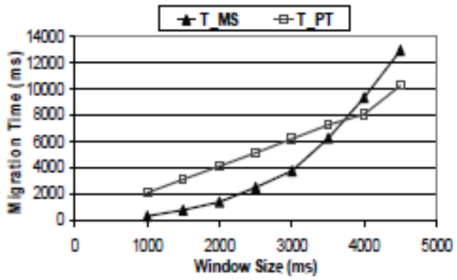


Fig :1 Comparison of static pipelined, materialized, and materialized plus replanted strategy.



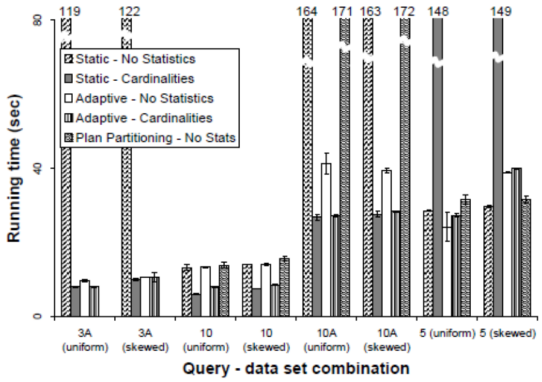
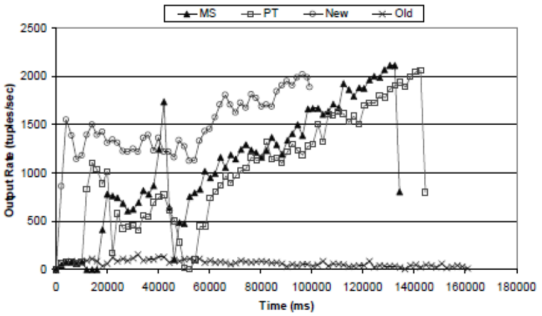


Fig no:2 Comparison of static optimization, adaptive query processing, and plan partitioning



V. CONCLUSION

This paper presents a cost based approach to minimize the number of refreshes required to execute an incoherency bounded continuous query. For optimal execution we divide the query into sub-queries and evaluate each sub-query at a chosen aggregator. Performance results show that by our method the query can be executed using less than one third the messages required for existing schemes. Further we showed that by executing queries such that more dynamic data items are part of a larger sub-query we can improve performance. Our query cost model can also be used for other purposes such as load balancing various aggregators, optimal query execution plan at an aggregator node, etc.

REFERENCES

1] Rajeev Gupta, Kirthi Ramaritham," *QueryPlanning For Continuous Aggregation Queries Over A Network Of data Aggregators*" vol 24, No. 6. June 2012.

[2] S.Shah, K. Ramamritham, and P. Shenoy, "*Maintainin Coherency of Dynamic Data in Cooperating Repositories,*" Proc. 28th Int'l Conf.Very Large Data Bases (VLDB), 2002

[3] Y.Zhou, B. Chin Ooi, and K.-L. Tan, "*Disseminating Streaming Data in a Dynamic Environment: An Adaptive and Cost BasedApproach,*" The Int'l J. Very Large Data Bases, vol.17, pp. 1465-1483, 2008.

[4] S. Agrawal, K. Ramamritham, and S. Shah, "*Construction of aTemporal Coherency Preserving Dynamic Data Dissemination Network,*" Proc. IEEE 25th Int'l Real-Time Systems Symp. (RTSS), 2004.

[5] R. Gupta, A. Puri, and K. Ramamritham, "*Executing Incoherency Bounded Continuous Queries at Web Data Aggregators,*" Proc. 14th Int'l Conf. World Wide Web (WWW), 2005.

[6] C.Olston, J. Jiang, and J. Widom, "*AdaptiveFilter for Continuous Queries over Distributed Data Streams,*" Proc. ACM SIGMOD Int'l Conf. Management of Data, 2003.

[7] S. Madden, M.J. Franklin, J. Hellerstein, and W.Hong, "*TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks,*" Proc. Fifth Symp. Operating Systems Design and Implementation, 2002.

[8] M.R. Garey and D.S. Johnson, "*Computers and Intractability-A Guide to the Theory of NPCompleteness,*" V Klee, Ed. Freeman, New York, 1979.

[10] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic Skyline on Uncertain Data," Proc. Int'l Conf. Very Large Data Bases (VLDB), 2007.

[11] G.M. *Siouris, Missile Guidance and Control Systems.* Springer Publication, 2004.

[12] M.A. Soliman, I.F. Ilyas, and K.C. Chang, "*Top-k Query Processing in Uncertain Databases,*" Proc. Int'l Conf. Data Eng. (ICDE), 2007.

[13] Y. Tao, R. Cheng, X. Xiao, W.K. Ngai, B. Kao, and S. Prabhakar, "*Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions,*" Proc. Int'l Conf. Very Large Data Bases (VLDB), 2005.

