USING RUNTIME DEPENDENCY MINING APPROACH FOR RE-OPTIMIZATION OF DATA STREAM PARTITIONING

Mr. S. S. Shinde
Assistant Professor
Computer Engineering Department
PVPPCOE, Mumbai, India

Abstract : Data is now getting generated with tremendous use of distributed network and sensor network. This data is processed in terms of streams. In distributed knowledge stream process, a program with multiple queries parallelized by partitioning input streams in keeping with the values of specific attributes, or partitioning keys. By applying completely different partitioning keys to different queries needs re-partitioning. It cause further communication and reduce throughput. Re-partitioning may be avoided by detecting dependencies between the partitioning keys applicable to every question. Existing partitioning optimization ways analyze question query syntax at compile-time to observe inter-key dependencies hence avoid re-partitioning. This work extends those compile-time ways by adding Auto-parallelization. Auto-parallelization technique involves locating regions in the application's data flow graph that can be replicated at run-time to apply data partitioning, to achieve scale. This work, propose an elastic auto-parallelization solution that can dynamically adjust the number of channels used to achieve high throughput without unnecessarily wasting resources. A runtime re-optimization step supported the mining of temporal approximate dependencies (TADs) between partitioning keys. A small indefinite amount is outlined during this work as a sort of dependency which will be roughly valid over a moving time window. This addresses the profitability problem associated with auto-parallelization of general-purpose distributed DataStream processing applications.

IndexTerms - Data-Stream Partitioning, Auto-parallelization, Runtime Optimization, TAD

I. INTRODUCTION

Stream computing is the computational paradigm which enables to carry out analytical tasks in efficient and scalable manner. A data streaming program is typically structured as a directed graph, in which vertices in graph represent queries/operators and edges represent streams. Knowledge traverses every question following the order outlined by the streams. In order to scale such data stream partitioning applications, stream processing system is free to decide how the application graph will be mapped to the set of available host's. As the data is coming from networked node, as the data is not stored directly on the disk, stream computing paradigm avoids this traditional store and process model of data management et.al. [1]. Stream data management has become a highly active research area and has inspired the development of several prototype systems like Aurora, Stream, TeleGraphCQ and Niagara. Linear load is used to compare the performance of SDMS to a Relational Database configured to process stream data inputs et.al.[6]. Current database management system architecture assumes a pull-based model of data access, when active party wants data, she submits a query to the passive party and answer is returned. So, stream-based applications push data to a system, that get must evaluate queries in response to detected events et.al.[5].

The partitioning strategy of a program powerfully affects its performance. A method that allocates completely different partitioning keys to two serial queries requires re-partitioning. To avoid this re-partitioning and to minimize this additional communication price, existing optimization ways think about compile-time optimization to either reconcile conflicting needs or perform "look ahead partitioning". In compile-time question, analysis and optimization concept is employed to acknowledge dependencies between the partitioning needs or candidate keys of serial queries hence avoid re-partitioning. Compile-time optimization presents its static nature of computing. Because of their static nature, compile-time optimization ways fail to acknowledge "dynamic" dependencies that are solely valid inside a particular time window. As the world becomes more interconnected and instrumented, there is a bulk flow of data coming from various software and hardware sensors in the form of continuous data streams. Examples can be found in several domains, such as financial markets, telecommunications, manufacturing, and healthcare. In all of these domains there is an increasing need to gather, process, and analyze these data streams to extract insights as well as to detect emerging patterns and outliers.

As a result, those ways miss re-optimization opportunities that would additional cut back the amount of re-partitioning. Samples of such things embody the Linear Road Benchmark (LRB), during which queries is partitioned off either by vehicle or by road section. Recognizing associate approximate dependency between vehicle and road segments, that is merely invalid once a vehicle enters a brand new section, would enable a partitioning optimizer to uniformly partition all queries by section, therefore avoiding a pricey re-partitioning between vehicle-partition able and segment-partition able queries. This case is off-course not restricted to the Linear Road Benchmark, together will imagine similar cases happening in numerous types of geospatial observance or analysis systems.

In this project work, work shows tendency to extend existing compile-time optimization ways by adding a re-optimization step supported runtime dependency mining. For this purpose, outline temporal approximate dependencies (TADs) as dynamic, approximate dependencies between moving entities over a time window. Small indefinite quantity mining permits the invention of methods that use less re-partitioning, leading to reduced communication and probably higher outturn. Streaming systems, it necessitates taking advantage of multiple host machines to achieve scalability. This requirement will become even more prominent with the ever increasing amounts of live data available for processing. The increased affordability of distributed and parallel computing, thanks to advances in cloud computing and multi-core chip design, has made this problem tractable.

II. LITERATURE SURVEY

M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin et.al.[3] described the long-running nature of continuous queries poses new scalability challenges for dataflow processing. CQ systems execute pipelined data flows that may be shared across multiple queries. The scalability of these data flows is limited by their constituent, state full operators - e.g. windowed joins or grouping operators. To scale such operators, a natural solution is to partition them across a shared-nothing platform. But in the CQ context, traditional, static techniques for partitioned parallelism can exhibit detrimental imbalances as workload and runtime conditions evolve. Long-running CQ data flows must continue to function robustly in the face of these imbalances. To address this challenge, introduce a dataflow operator called flux that encapsulates adaptive state partitioning and dataflow routing. Flux is placed between producer-consumer stages in a dataflow pipeline to repartition state full operators while the pipeline is still executing. work present the flux architecture, along with repartitioning policies that can be used for CQ operators under shifting processing and memory loads. Showing that the flux mechanism and these policies can provide several factors improvement in throughput and orders of magnitude improvement in average latency over the static case.

T. Johnson, M. S. Muthukrishnan, V. Shkapenyuk, and O. Spatscheck et.al. [4] Stated Data Stream Management Systems (DSMS) are gaining acceptance for applications that need to process very large volumes of data in real time. The load generated by such applications frequently exceeds by far the computation capabilities of a single centralized server. In particular, a single- server instance of our DSMS, Gigascope, cannot keep up with the processing demands of the new OC-786 networks, which can generate more than 100 million packets per second. This work, explore a mechanism for the distributed processing of very high speed data streams. Existing distributed DSMSs employ two mechanisms for distributing the load across the participating machines: partitioning of the query execution plans and partitioning of the input data stream in a query-independent fashion. However, for a large class of queries, both approaches fail to reduce the load as compared to centralized system, and can even lead to an increase in the load. This work present an alternative approach - query-aware data stream partitioning that allows for more efficient scaling. Presenting methods for analyzing any given query set and choose the optimal partitioning scheme, and shows how to reconcile potentially conflicting requirements that different queries might place on partitioning. Concluding with experiments on a small cluster of processing nodes on high-rate network traffic feed that demonstrates with different query sets that our methods effectively distribute the load across all processing nodes and facilitate efficient scaling whenever more processing nodes becomes available.

N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Cetintemel, M. Cherniack, R. Tibbetts, and S. Zdonik et.al. [5] describes a unification of two different SQL extensions for streams and its associated semantics. Used the data models from Oracle and Stream Base as examples. Oracle uses a time-based execution model while Stream Base uses a tuple- based execution model. Time-based execution provides a way to model simultaneity while tuple-based execution provides a way to react to primitive events as soon as they are seen by the system.

Arasu, S. Babu, and J. Widom, et.al. [6] CQL, a continuous query language, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based declarative language for registering continuous queries against streams and stored relations. Beginning with presenting an abstract semantics that relies only on black- box mappings among streams and relations. From these mappings we define a precise and general interpretation for continuous queries. CQL is an instantiation of our abstract semantics using SQL to map from relations to relations, window specifications derived from SQL-99 to map from streams to relations, and three new operators to map from relations to streams. Most of the CQL language is operational in the STREAM system. Work presents the structure of CQL's query execution plans as well as details of the most important components: operators, inter operator queues, synopses, and sharing of components among multiple operators and queries. Examples throughout the work are drawn from the Linear Road benchmark recently proposed for DSMSs. It also curate a public repository of data stream applications that includes a wide variety of queries expressed in CQL. The relative ease of capturing these applications in CQL is one indicator that the language contains an appropriate set of constructs for data stream processing.

Researchers Burga Gedik, Scott Schneider, Martin Hirzel and Kun-Lung Wu et.al[2] have recently discovered several interesting, self-organized regularities from the World Wide Web, ranging from the structure and growth of the Web to the access patterns in Web surfing. What remains to be a great challenge in Web log mining is how to explain user behavior underlying observed Web usage regularities. This work address the issue of how to characterize the strong regularities in Web surfing in terms of user navigation strategies, and present an information foraging agent based approach to describing user behavior. By experimenting with the agent-based decision models of Web surfing, work aim to explain how some Web design factors as well as user cognitive factors may affect the overall behavioral patterns in Web usage. In order to Further characterize user navigation regularities as well as to understand the effects of user interests, motivation, and content organization on the user behavior. This work present an information foraging agent based model that takes into account the interest profiles, motivation aggregation, and content selection strategies of users and, thereafter, predicts the emerged regularities in user navigation behavior. In summary, our work offers a means for explaining strong Web regularities with respect to user Interest Profiles, Web Content Distribution and coupling, and user navigation strategies. It enables to predict the effects on emergent usage regularities if certain aspects of Web servers or user foraging behaviors are changed. While presenting an interesting and promising research direction, point out that one of the useful extensions for future work would be to show how the quantitative representations or constructs as used in modeling Web contents and user interest profiles.

III. IMPLEMENTATION DETAILS

Problem Statement:

Enhance the compile-time methods by adding a runtime re-optimization step based on the mining of temporal approximate dependencies (TADs) between partitioning keys to achieve scalability.

Existing System:

In case of existing system, It is working to process data streams at compile time. System optimizes the data stream during compilation according to the value of specific attributes or desired partitioning key. As the need of re-partitioning increases because of applying independent partitioning keys, it cause extra communication overhead in turn reduce throughput of the system. Existing system analyze query syntax to detect inter-key dependencies at compile time to avoid re-optimization. But this methodology still maintains system nature static and not works over moving time period.

System Architecture:

As shown in diagram Following diagram shows system architecture.

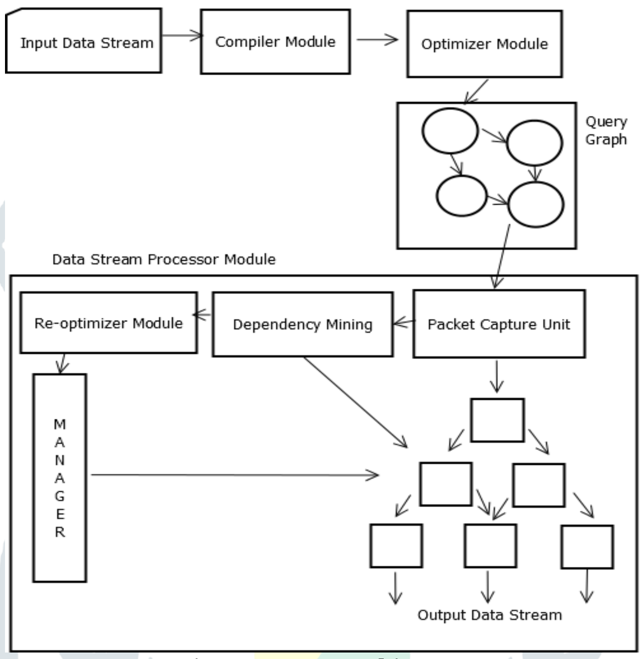

Fig 1: System Architecture

Proposed System:

In proposed system our aim is to extend existing work and implement a data stream processing system which will re-optimize the queries at runtime by checking the dependencies to avoid re-partitioning. This system will able to reduce the communication overhead caused by the static data stream processing system which is optimizing queries at compile time. Objective of this work is to avoid re-partitioning by checking dependencies at runtime over moving time window.

Mathematical Model:

TAD main function

```
        Inputs: m-dimensional vectors X1,X2,…,Xn
        Outputs: A partition of X1,X2,…,Xn into K clusters
                TAD (X1, X2,…,Xn)
                Initialize (X1, X2… Xn);
                while (Num_Clusters>K)
                for (each cluster Ci) //message sending
                Cj=Find_Nearest_Neighbor (Ci);
                Msg_Send (Ci,Cj);
                for (each cluster Ci) //message receiving
                Cj=Msg_Rcv (Ci) // check message
                if (Cj not equal to NULL)
                New Cluster C'= Merge (Ci, Cj); Num_Clusters= Num_Clusters-1;
                Msg_Send (Ci, Cj) //send a message from Ci to Cj
                Ci.TO=j;
                Cj.FROM=i;
                Msg_Rcv (Ci) //check the message box
                if (Ci.FROM=Ci.TO=j) //find mutually
                return Cj;
                else
                return NULL;
```

Generate distance array Ai on each processor i and store the nearest neighbors in the special array B. O(n). Obtain the nearest neighbor on each processor i. O(1). Identify couples by message sending and receiving on each processor i. O(1). For each couple (i, j), do the following: Update distance array Ak on each processor k, where k is any processor other than i and j. O(1), Update B on each processor. O(1), Merge couple (i,j) by dropping processor j (i<j). O(1). If two or more processors remain, go to step 2.

Result Set:

The project going to reduce the optimal partitioning and it has a better performance of compile time and re optimization process. Then it reduced communication and potentially higher throughput. And the performance evaluation of our TAD mining algorithm is required to confirm the feasibility of runtime re-partitioning method.

IV. CONCLUSION

Compile-time optimization method that, partition existing optimization methods offers a runtime again by adding customizable runtime dependencies is presented based on the mining phase. Windows dynamic dependencies, the proverbial estimated multiple dependencies between moving bodies as defined. Mining strategies that reduce communication and potential re-partitioning, resulting in higher throughput, in search of use. The current compile-time compared with the ways, our method to the Division tactics adopted by reduced communication costs.

V. FUTURE SCOPE

This work defer to future work some advanced techniques that choose plans given uncertainty regimes over statistics or choose a set of plans, each of which has an associated validity range specify over statistics, and switch between these plans at runtime depending on the observed statistics. Clearly these techniques are more complex than RoPE, the risks from picking worse plans are larger here, and to the best of our knowledge using these ideas in the context of parallel plans is an open problem.

VI. ACKNOWLEDGE

Special thanks go to authors EmericViel, Haruyasu Ueda. Contributed to this paper for their valuable comments and sharing their knowledge and idea.

REFERENCES

[1] EmericViel, Haruyasu Ueda, Data Stream Partitioning Re-Optimization Based on Runtime Dependency Mining,in ICDE Workshop 2014 978-1-4799-3481-2/14/ 2014 IEEE

[2] Bugra Gedik, scott Schneider, Martin Hirzel, and Kun-Lung Wu, "Elastic Scaling for Data Stream Processing", in IEEE Transactions on parallel and distributed Systems, pp.1447-1463, 2014.

[3] M. A. Shah, J. M. Heller stein, S. Chandrasekaran, and M. J. Franklin, "Flux: an adaptive partitioning operator for continuous query systems", in Proc. ICDE03, pp.25-26 2013.

[4] T. Johnson, M. S. Muthukrishnan, V. Shkapenyuk, and O. Spatscheck, "Query- aware partitioning for monitoring massive network data streams", in Proc. SIG-MOD08, 2008, pp. 1135-1146.

[5] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Cetintemel, M. Cherniack, R. Tibbetts, and S. Zdonik, "Towards a streaming SQL standard", in Proc. VLDB Endowment, vol. 1, pp. 1379-1390, Aug. 2008.

[6] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. B. Zdonik, "Scalable distributed stream processing, in Proc. CIDR03, 2003, pp. 257-268.

[7] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibetts, "Linear road: a stream data management benchmark", in Proc. VLDB04, 2004, pp. 480-491.

[8] D. DeWitt and J. Gray, "Parallel database systems: the future of high performance database systems", in Communications of the ACM, vol. 35, pp. 85-98, Jun. 1992.

[9] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik. "The design of the borealis stream processing engine", in Proc. CIDR05, 2005, pp. 277-289.