ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2015

Effective and Efficient Re-optimization Method for Data-Stream Partitioning at Runtime

Shinde Sumit Sudhakar

P.G. Student, Dept. of Computer Engineering, Dhole Patil College of Engineering, Wagholi, Pune, India

ABSTRACT: Stream process has gathered interest for its ability to investigate giant amounts of information in time period. The rising quantity of information generated by connected devices and enterprise sensors makes it particularly vital for today's analytic systems to scale expeditiously.The long-running nature of continuous queries poses new scalability challenges for dataflow processing.With tremendous use of distributed network and sensor network data is now getting generated in huge. This generated data is processed in terms of streams. In distributed knowledge stream process, partitioning input streams in keeping with the values of specific attributes, or partitioning keys a program with multiple queries is parallelized. Applying completely different partitioning keys to different queries needs re-partitioning. It cause further communication and reduce throughput. By adding Auto-parallelization technique this work is extended for compile-time to run-time. This 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. The partitioning strategy of a program powerfully affects its performance. A method that allocates completely different partitioning keys to serial queries would require re-partitioning the intermediate stream between the queries, inflicting additional communication. To avoid re-partitioning and 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 is employed to acknowledge dependencies between the partitioning needs, or candidate keys, of serial queries and avoid re-partitioning. Because of their static nature, compile-time optimization ways fail to acknowledge "dynamic" dependencies that are solely valid inside a particular time window.

KEYWORDS*:* Data-Stream Partitioning, Auto-parallelization, Runtime Optimization, TAD

I. INTRODUCTION

A data streaming program is typicallystructured as a directed graph, in which vertices in graph represent queries/operators and edges represent streams.To carry out analytical tasks in efficient and scalable manner Stream computing is the computational paradigm which is used. Knowledge traverses everyquestion following the order outlined by the streams.Stream process has gathered interest for its ability to investigate giant amounts of information in time period. The rising quantity of information generated by connected devices and enterprise sensors makes it particularly vital for today's analytic systems to scale expeditiously. 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 not stored directly on the disk coming from networked node, 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. To compare the performance of SDMS to a Relational Database configured to process stream data inputs Linear load method is used 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

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2015

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].

Data Stream Management Systems (DSMS) are gaining acceptance for applications that need to process very large volumes of data in real time.Detecting dependencies between the partitioning keys applicable to every question Re-partitioning may be avoided. Existing work for partitioning optimization analyse question query syntax at compile-time hence observe inter-key dependencies to avoid re-partitioning.The partitioning strategy of a program powerfully affects its performance.Compile-time optimization presents its static nature of computing. Because of their static nature, compile-time optimizationways fail to acknowledge “dynamic” dependencies that aresolely valid insidea particular time window.A method that allocates completely different partitioning keys to twoserial queries requires re-partitioning. To avoid this re-partitioning and to minimize this additional communication price, existing optimizationwaysthink about compile-time optimization to either reconcile conflicting needs or perform “look ahead partitioning”. In compile-time question, analysis and optimization concept is employedto acknowledge dependencies between the partitioning needs or candidate keys of serial queries hence avoid re-partitioning. 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 analyse these data streams to extract insights as well as to detect emerging patterns and outliers.

Recognizing associate approximate dependency between vehicle and road segments, thatis merelyinvalidonce a vehicle enters a brand newsection, 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, togetherwill imagine similar cases happening in numeroustypes of geospatial observance or analysis systems.As a result, those ways miss re-optimization opportunities that wouldadditionalcut backthe amount of re-partitioning. Samples of such thingsembody the Linear Road Benchmark (LRB), during which queries ispartitioned off either by vehicle or by road section.

In this project work, work shows tendency to extend existing compile-time optimizationways 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. Streaming systems, it necessitates taking advantage of host machine to achieve scalability. This requirement will become even more prominent with the ever increasing amounts of live data available for processing. Small indefinite quantity mining permitsthe invention of methods that use less re-partitioning, leading to reduced communication and probably higher outturn.. 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. Related work

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. Flux is placed between producer-consumer stages in a dataflow pipeline to repartition state full operators while the pipeline is still executing. Work presents the flux architecture, along with repartitioning policies that can be used for CQ operators under shifting processing and memory loads. CQ systems execute pipelined data flows that may be shared across multiple queries. 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.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. 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.

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. 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. The load generated by such applications frequently exceeds by far the computation capabilities of a single centralized server. 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. This work present an alternative approach - query-aware data stream partitioning that allows for more efficient scaling. Presenting methods for analysing 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. 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.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. 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. 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.

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. Most of the CQL language is operational in the STREAM system. 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.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. 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 BurgaGedik, 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. 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 behaviour. 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 behavioural patterns in Web usage.What remains to be a great challenge in Web log mining is how to explain user behaviour 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 behaviour. 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 behaviour.

III. Implementation Details

A. *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 analyse 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.

B. *Problem Statement:*

Adding a runtime re-optimization method to enhance the compile-time methods based on the mining of temporal approximate dependencies (TADs) between partitioning keys to achieve scalability.

C. *Proposed System:*

In proposed system our aim is to implement a data stream processing system which will re-optimize the queries at runtime by checking the dependencies to avoid re-partitioning by extending existing compile-time methods. 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 check dependencies at runtime over moving time window by avoiding re-partitioning.

D. *System Architecture:*

Following diagram shows system architecture.

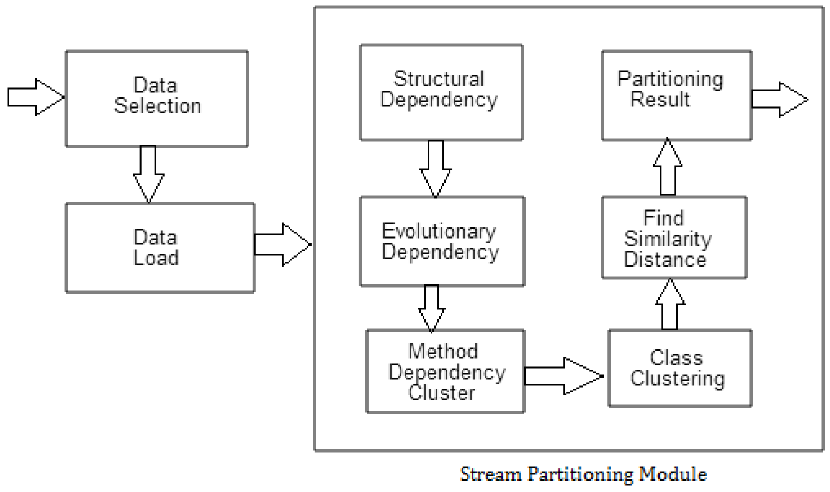

Fig 1: System Architecture

E. *Mathematical Model:*

- Existing System

  Incremental Detection Algorithm:

  Input: An extension of traditional FDs: (R: X $\rightarrow$ Y, Tp)

Process: $X \rightarrow Y$: embedded traditional FD on R
Tp: a pattern tableau
– attributes: $X \cup Y$

– each tuple in Tp consists of constants and unnamed variable

Output:

A, B, C $\Rightarrow$ D, E (support, confidence)

Where support= count(A,B,C,D,E)/database_size

Confidence= count(A,B,C,D,E)/count (A,B,C)

Find all rules which satisfy minsupp

Find all itemsets which satisfy minsupp

- Proposed System

```
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
returnCj;
else
return NULL;
```

F. *Result Set :*

The work is going to reduce the optimal partitioning for better performance of compile time and re optimization process. Performance evaluation of our TAD mining algorithm is required to confirm the feasibility of runtime re-partitioning method.

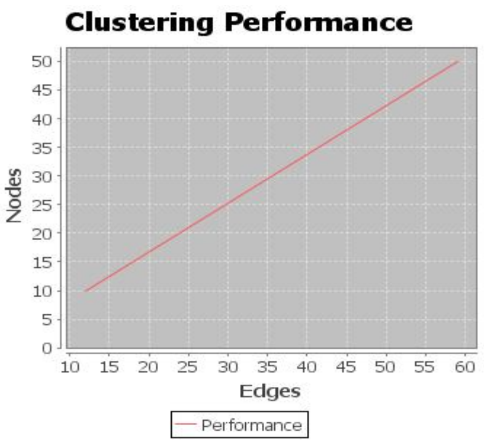

Fig. 2 Clustering Performance

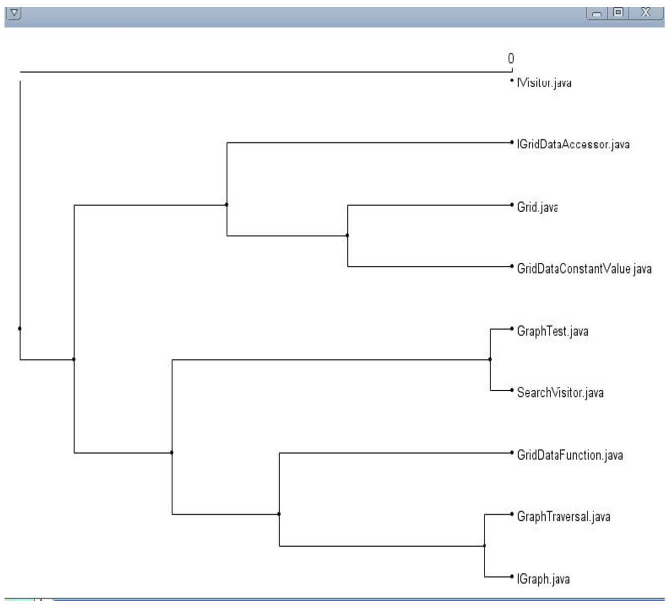

Fig. 3 Partitioning Result

IV. Conclusion

Adding customizable runtime dependencies, the Compile-time optimization method partition existing optimization methods and offers a runtime framework based on the mining phase. Windows dynamic dependencies between data stream query is defined. Used mining strategies reduce communication and potential re-partitioning, results in higher throughput.

V. Future Work

This work is at very early stage, hence implementation of the exception handling and related runtime strategy switching is a challenge. A real life evaluation need to be carried out for throughput and scalability. Performance evaluation of TAD dependency mining algorithm is required to conform the feasibility of runtime re-optimization method.

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. StreamBase. (2013) StreamSQL Guide. [Online]. Available: http:// www.stream base.com /developers /docs/latest/streamsql
7. 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.
8. A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina,M. Stonebraker, and R. Tibetts, "Linear road: a stream data managementbenchmark", in Proc. VLDB04, 2004, pp. 480-491.
9. 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.
10. 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, andS. B. Zdonik. "The design of the borealis stream processing engine", in Proc.CIDR05, 2005, pp. 277-289.

BIOGRAPHY

Mr. Shinde Sumit Sudhakaris a Post Graduate Student in the Department of Computer Engineering, Dhole Patil College of Engineering, Wagholi, Pune. He received Bachelor of Engineering (BE) degree in 2012 from SGBAU, Amravati University, MS, India. His research interests are Data Mining, Web Technology etc.