International Journal of Engineering and Advance Technology Studies

Vol.1, No.1, pp.25-32, March 2013

Published European Centre for Research Training and Development UK (www.ea-journals.org)

ENRICHING BATCHED STREAM PROCESSING USING BAYESIAN NETWORK FOR WEB SERVICES

Prof D.M Thakore
Professor, BVCOE, Pune, India

Prof N.B Kadu
Assistant Professor, PREC, loni, Ahmednagar, India

Abstract - *The need for secure data transactions has become a necessity of our time. Medical records, financial records, legal information and payment gateway are all in need of secure data transaction process. There have been several methods proposed to perform secure, fast and scalable data transactions in web services. As the web servers deals with the huge amount of query it becomes really difficult to handle all the query to perform in the limited amount of time , and the failure of this can crash the web service or it may cause transaction failures, which can cause a huge financial losses to the organizations. Batched stream processing is a new distributed data processing paradigm that models recurring batch computations on incrementally bulk-appended data streams. The model is inspired by our empirical study on a trace from large-scale production data-processing clusters at the web server end; it allows a set of effective query optimizations that are not possible in a traditional batch processing model. By applying Bayesian Networks concept we stream the query so that similar queries are batched as cluster of queries which we called them as jumbo query. These batched queries are then commit for the transaction so that the complete process runs without any much load on the servers and they can handle heavy amount of transactions without any failures which lead to any fuss.*

Keywords: Bayesian Networks, Web Server, Batched Stream processing, Query Series, Jumbo Query.

INTRODUCTION

At the web servers end where each query specifies computation on a large bulk of data. These systems tend to process queries individually. In reality, we face the challenging problem of executing a large number of complicated queries on a large amount of data every day across thousands of servers. Optimization of query executions is essential for effective resource utilization and high throughput.

Our study further reveals that the redundancy in the web server transactions is due to correlations among queries. The workload exhibits temporal correlations, where it is common to have a series of queries involving the same recurring computations on the same data stream in different time windows. The workload further exhibits spatial correlations, where a data stream is often the target of multiple queries involving different but somewhat overlapping computations. For example, one data stream might store web search logs and is

appended daily with new entries. A query might be issued daily to retrieve the top ten hottest keywords from the search logs over a sliding window of the last 7 days. These daily queries have clear temporal correlations. Another set of daily queries might probe the geographical distribution of search entries in the same 7-day window, thereby exhibiting spatial correlations with the first set of queries.

To expose temporal correlations among queries, we introduce Batched Stream Processing (BSP) to model recurring (batch) computations on incrementally bulk-appended data streams, which is a dominant pattern that we observed in the studied trace. Recurring computations on the same data stream form a query series. An example query series consists of the daily queries for the hottest keywords in the last 7 days, as described earlier. With query series, an execution of an earlier query in a query series could help the execution of later queries in the same query series. First, it could preserve intermediate results that are needed by later queries; for example, for the hottest-keyword query series, it might be beneficial to create a daily keyword count because it will be used by the next 6 days of queries in the same query series. Those intermediate results resemble materialized views [1] in database systems. Second, profiling the execution of an earlier query could help guide optimizations of later queries in the same query series.

Later queries in a query series are driven by bulk updates to input data streams, rather than being triggered by query submission from users. This has significant implications. Queries from multiple query series operating on the same input data stream can now be aligned to execute together when new bulk updates occur. This maximizes opportunities to remove redundant computations or I/O across those queries; such redundancy arises from spatial correlations among those queries. With the Batched Stream Processing model, traditional database optimization techniques, especially those for continuous queries [2] and multiple queries [3], become relevant.
We have studied the temporal stability of data streams to check the feasibility of guiding the optimization of a query based on profiling of the previous executions, especially from those in the same query series. We have found data distributions of newly appended up dates are stable across different days.

In summary, our study of the trace reveals strong temporal and spatial correlations among queries; those correlations lead to significant I/O redundancies and temporal load imbalance. There is a clear indication that queries are mostly driven by new updates. Recurring queries are expected to exhibit similar behavior because of the stability observed on data stream properties: this lays the foundation for optimizing recurring queries based on the profiling of an earlier execution. All these results argue for the batched stream processing model, as well as hinting at the potential benefits of such a model.

We have built a system to support Batched Stream Processing: our System allows users to submit query series and implements a set of global optimizations that take advantage of the notion of query series. Query execution in our system is triggered by arrivals of new bulk updates to streams. A query is decomposed into a number of sub queries, each of which is performed on a new bulk update. Our System aligns sub-queries from different query series into a single *jumbo query* and optimizes the jumbo query to remove redundancies and improve performance.

The rest of the paper is organized as follows: section 2 will introduce about related work done so far. Section 3 will give proposed work, section 4 will explain about results. Section5 will conclude this paper.

PROPOSED METHOD

PART I: BUILDING BAYESIAN NETWORK

A Bayesian network is a graphical structure that allows us to represent and reason about an uncertain domain. The nodes in a Bayesian network represent a set of random variables, X = X1; Xi; Xn, from the domain. A set of directed arcs (or links) connects pairs of nodes, Xi! Xj, representing the direct dependencies between variables.
Assuming discrete variables, the strength of the relationship between variables is quantified by conditional probability distributions associated with each node. The only constraint on the arcs allowed in a BN is that there must not be any directed cycles: There are a number of steps that a knowledge engineer1 must undertake when building a Bayesian network. At this stage we will present these steps as a sequence;

Throughout the remainder of this section we will use the following simple Query diagnosis problem.Example problem: A Database DELTE Query at web server may occur before it may be insert or update from other client panel, In the same way a select query may also occurs before its been Insert or update from other client panel. When this kind of scenario occurs in maintaining huge amount data, a big transaction failures may can lead data loosing or any other threat to the system. So in short all the basic database queries like Insert, Update, Delete and Select are largely depend on arrival of insert query at the server's end. So its Probability is more. So Bayesian Network is very good option to judge the probability and handle all the queries according to priority and executes the query in bulk to reduce the web servers load.

Nodes and values
First, the knowledge engineer must identify the variables of interest. This involves answering the question: what are the nodes to represent and what values can they take, or what state can they be in? For now we will consider only nodes that take discrete values. The values should be both mutually exclusive and exhaustive, which means that the variable must take on exactly one of these values at a time. Common types of discrete nodes include:

- Boolean nodes, which represent propositions, taking the binary values true (T) and false (F). In a domain Database query represents nodes of our system.
- Ordered values. For example, a node Insert represents its dependency on the other query as {low, medium, high}.
- Integral values. For example, a node called commit represent integral value.

Below table shows preliminary choices of nodes

Node name	Type	Values
Insert	Boolean	{high}
Select	Boolean	{high,medium,low}
Update	Boolean	{high,medium,low}
Delete	Boolean	{high,medium,low}

Table1: Node Priority List

Bayesian Structure

The structure, or topology, of the network should capture qualitative relationships between variables. In particular, two nodes should be connected directly if one affects or causes the other, with the arc indicating the direction of the effect. it can be see in the below fig

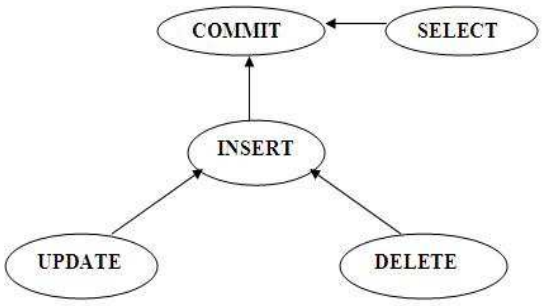

Figure 1: A BN for Query Solving problem

3.1.3 Reasoning with Bayesian networks

Now that we know how a domain and its uncertainty may be represented in a Bayesian network, we will look at how to use the Bayesian network to reason about the domain. In particular, when we observe the value of some variable, we would like to condition upon the new information. The process of conditioning (also called probability propagation or inference or belief updating) is performed via a "flow of information" through the network. Note that this information flow is not limited to the directions of the arcs. In our probabilistic system, this becomes the task of computing the posterior probability distribution for a set of query nodes, given values for some evidence (or observation) nodes.

3.1.4 Types of reasoning

Bayesian networks provide full representations of probability distributions over their variables. That implies that they can be conditioned upon any subset of their variables, supporting any direction of reasoning.

For example,

- **Diagnostic reasoning**, i.e., reasoning from one query to another query then updates his belief. Note that this reasoning occurs in the opposite direction to the network arcs.(D-Delete , C-Commit)

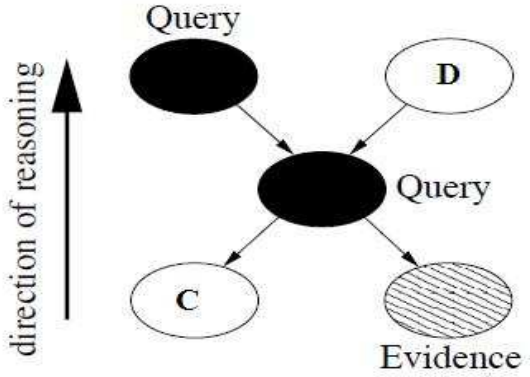

FIG 2 : DIAGNOSTIC REASONING

- **Predictive reasoning**, reasoning from new information about causes to new beliefs about effects, following the directions of the network arcs.

(P- Probability)

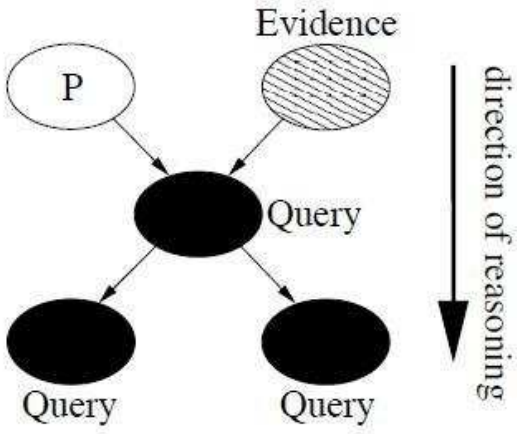

FIGURE 3: PREDICTIVE REASONING

Conditional probabilities

Once the topology of the BN is specified, the next step is to quantify the relationships between connected nodes – this is done by specifying a conditional probability distribution for each node. As we are only considering discrete variables at this stage, this takes the form of a conditional probability table (CPT).

First, for each node we need to look at all the possible combinations of values of those parent nodes. Each such combination is called an instantiation of the parent set. For each distinct instantiation of parent node values, we need to specify the probability that the child will take each of its values.

Bayesian Execution Plan

A central Bayesian Network was first obtained using the whole query feature occurred in the past by studying web log file at server end. We then split these features into two sets corresponding to the respective scenario in the web server. and then by applying Bayesian Network theorem we will query priority based on that we are implementing our batched Stream model that we discussed in the below section.

Batched Stream Processing

Our study on the production trace indicates that a significant portion of queries follow the Batched Stream Processing model, where source input data are modeled as periodically appended *streams* with recurring queries triggered upon bulk appends to these streams. Each single bulk update creates a *segment* of the data stream; different segments are differentiated with their timestamps that indicate their arrival times. Recurring computations form a query series, where each query instance in a query series is triggered when new segment(s) are appended. In the batched stream processing model, users can submit query series that explicitly convey temporary correlations among individual queries; while in a traditional batch processing system these queries would have to be submitted separately. This seemingly simple notion of query series enables a set of new optimizations that are not available or hard to implement in current batch-processing systems.

With query series, execution of an earlier query in a query
series is aware of future query executions in the same query series, thereby offering opportunities for optimizations. First, an execution of an earlier query could piggyback statistical properties of input data streams or intermediate data; such statistical information could guide effective optimizations of later queries. As we have already seen in the empirical study, important statistical properties such as the data distributions of stream and filter selectivity tend to be stable as a data stream grows over time. Previous executions are also effective in estimating the cost of custom functions, which are an important part of data processing queries. Such estimation
Would have been difficult otherwise.

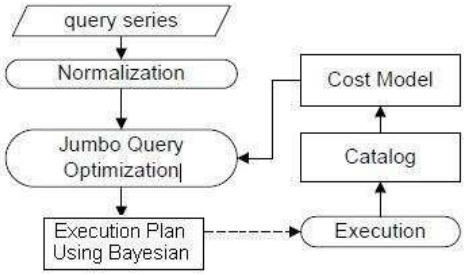

Figure 4: BSP MODEL USING BAYESIAN

More importantly, with query series, query execution is now mostly driven by bulk updates to input streams rather than by submissions from users. Queries in different query series that operate on the same input stream can now be aligned and optimized together as one aggregated query. This helps remove redundancies, which are spatial correlations across query series. Given the power-law distribution on data stream accesses that we observe, a significant number of query series would access the most popular data streams and offer opportunities for optimizations when aligned and combined. To increase chances of sharing across query series, a query might be further decomposed into a series of smaller queries, each on a subset of input stream segments, followed by a final step of aggregating the results of the smaller queries to obtain the final result. Query decomposition ensures that all queries on the same stream process the data on aligned segment windows, even if some queries originally process data over multiple segment windows.

We have developed Batched query processing engine that supports the BSP model and enables new optimizations. Our System allows users to submit a query series by specifying the period and the number of recurrences of the computations. We use the following terms to define the computation units in an execution:

- *S-query*. An S-query is a single query occurrence of a query series; it can access one or more segments on one or more streams.
- *SS-query*. Intuitively, an SS-query is a sub-computation of an S-query that can be executed when a new segment arrives. We associate with each SS-query a timestamp indicating its planned execution time. It is usually equal to the maximum timestamp of the segments it accesses: arrival of the segment with the maximum timestamp triggers execution of the SS-query. An S query can be decomposed into one or more SS-queries in a normalization process.
- *Jumbo-query*. A jumbo-query is a set of SS-queries with the same timestamp; that is, a jumbo query includes all SS-queries that can be executed together, thereby leveraging any common I/O and computations among these SS-queries.

Figure 4 shows how query series are processed in OUR proposed system .When a query series is submitted, BSP normalizes it into a sequence of SS-queries and combines them with their corresponding jumbo-queries. This allows BSP to align query series based on the segments they involve.

RESULTS

As with the stream processing model [4], computation in the BSP model is triggered by new updates to data streams,but without resource and timing constraints normally associated with stream processing; as with the batch processing model, each query in a query series is a batch job, but computations are recurring, as it is triggered by a (bulk) update to data streams.

Batch processing: There is a large body of research on query optimizations for batch processing in traditional (parallel) databases [5]. Shared-nothing database systems like Gamma [6] and Bubba [7] focus mainly on parallelizing a single query. As for multiple query optimizations, materialized views are an effective mechanism in exploiting the result of common sub expressions within a single query or among multiple queries.

CONCLUSION

Bayesian Network: Bayes' theorem allows us to update the probabilities of variables whose state has not been observed given some set of new observations. Bayesian networks automate this process, allowing reasoning to proceed in any direction across the network of variables. They do this by combining qualitative information about direct dependencies (perhaps causal relations) in arcs and quantitative information about the strengths of those dependencies in conditional probability distributions. Computational speed gains in updating accrue when the network is sparse, allowing d-separation to take advantage of conditional independencies in the domain

Stream processing. Stream processing systems such as STREAM [8] and NiagaraCQ [9] usually process real-time and continuous data streams. Due to resource and time constraints, stream data are usually not stored persistently.
Continuous queries run on a stream for a period of time, and return new results as new data arrives. Query processing algorithms for incremental computation [10] and for identifying common sub-queries among continuous queries are proposed to process streams efficiently.

References

[1] P. Agrawal, D. Kifer, and C. Olston. Scheduling shared scans of large data files. *Proc. VLDB Endow.* 1(1):958–969, 2008.
[2] D. Terry, D. Goldberg, D. Nichols, and B.Oki.Continuous queries over append-only databases. In
ACM SIGMOD, 1992.
[3] T. K. Sellis. Multiple-query optimization. *ACM Trans.*
Database Syst., 13(1):23–52, 1988.
[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and
J. Widom. Models and issues in data stream systems.
In *ACM PODS*, 2002.

[5] D. DeWitt and J. Gray. Parallel database systems:
the future of high performance database systems.
Commun. ACM, 35(6):85–98, 1992.
[6] D. J. Dewitt, S. Ghandeharizadeh, D. A. Schneider,
A. Bricker, H. I. Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Trans. On Knowl. and Data Eng.*, 2(1):44–62, 1990.
[7] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping Bubba, a highly parallel database system. *IEEE Trans. on Knowl. and Data Eng.*, 2(1):4–24, 1990.
[8] The Stanford STREAM Group. STREAM: The Stanford stream data manager. *IEEE Data Engineering Bulletin, 26(1)*, 2003.
[9] J. Chen, D. J. Dewitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *ACM SIGMOD*, 2000.
[10] D. Terry, D. Goldberg, D. Nichols, and B. Oki.
Continuous queries over append-only databases. In
ACM SIGMOD, 1992.

******

(deventhakur@yahoo.com, kamleshkadu@rediff.com)