Seminar on Query Optimization in Databases

Deep Modh
Indian Institute of Technology, Bombay

Supervisor : Prof S.Sudarshan

April 28, 2017

Content of Presentation

1. Brief Introduction to Parallel and Multi-Objective Query Optimization
2. Monitoring Streams Using Aurora
3. Runtime Optimization of Join Location

Brief Discussion of Common papers

1) Volcano Framework
   a) Logical and physical space exploration
   b) Cost-based optimization
2) SCOPE Optimizer
   a) Parallel query processing integration with optimization
   b) Distribution and Merge operators
3) Join Order Optimizations
   a) Exploring complete rule-sets
   b) Cross-product free joins

Monitoring Streams using Aurora *

* All the content and figures explaining Aurora are taken from Monitoring Streams - A New Class of Data Management Applications Donald Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, Stanley B. Zdonik VLDB 2002: 215-226.

Monitoring Streams using Aurora

Monitoring applications differ substantially from traditional DBMS.

1. Traditional DBMS has Human-Active, DBMS-Passive model. Monitoring applications has DBMS-Active, Human-Passive model
2. None of the traditional DBMS have implementation that scales to a large number of triggers on the other hand most monitoring applications are trigger-oriented

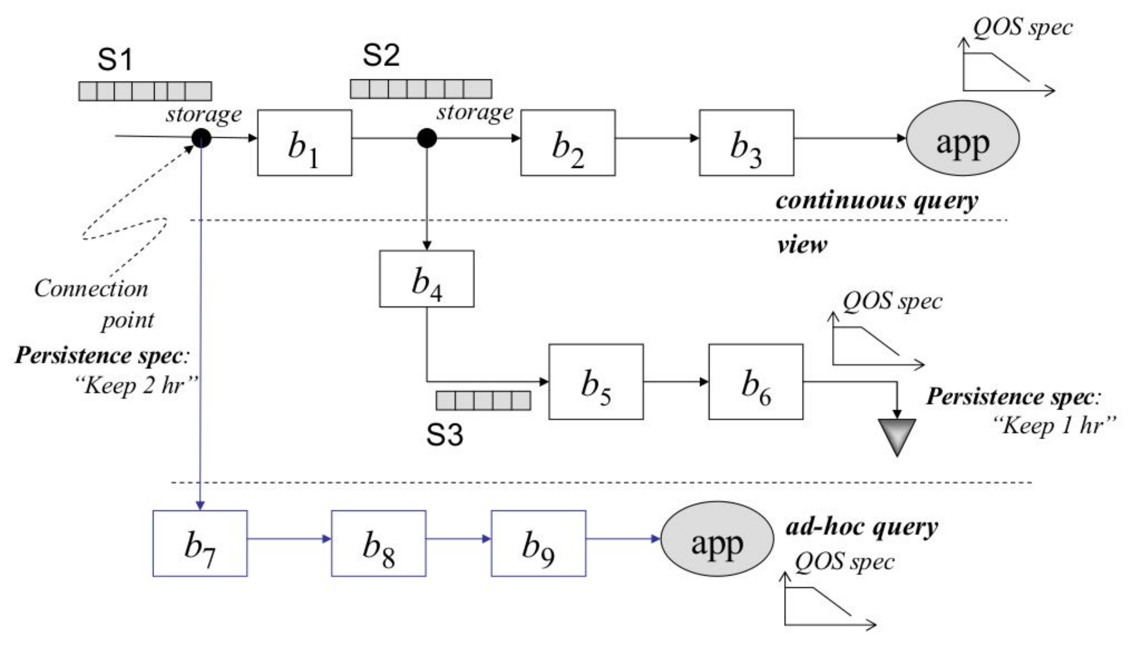
Aurora System Model

Aurora is a data-flow system and uses the boxes and arrows paradigm.

Tuples flow through loop-free directed graph of processing operations (boxes) and are presented to application as output stream.

Aurora consists of two kind of operators; windowed operators like slide, tumble, latch and operators that act on a single tuple (eg. filter)

Aurora supports three modes of operation



Aurora Optimization

Locally optimise portion of network surrounded by connection points.

Inserting Projections

Combining Boxes

Reordering Boxes : bj following bi; cost = $c(b_i) + c(b_j) \times s(b_i)$

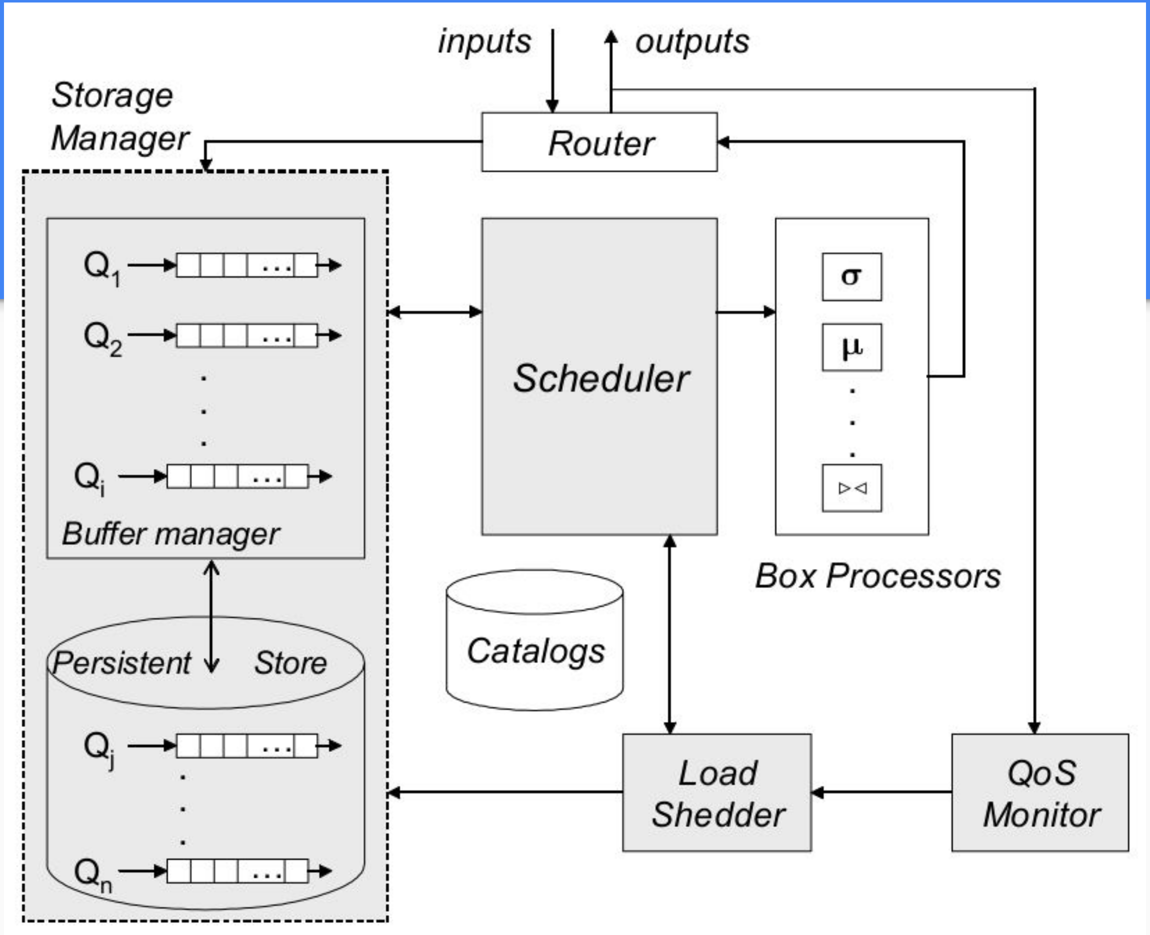
Interchange if $1 - s(b_j) / c(b_j) > 1 - s(b_i) / c(b_i)$

Optimal order

Ad-Hoc Query Optimization

Information is organized as B-tree at connection point

Initial boxes can pull information from the B-tree using indexed lookup if possible. (eg. if box is join)


inputs
outputs
Storage Manager
Router
Scheduler
σ
μ
⋈
Box Processors
Q1
Q2
Qi
Buffer manager
Persistent Store
Qj
Qn
Catalogs
Load Shedder
QoS Monitor

Aurora Run-Time Architecture

Inputs from data sources and outputs from boxes are fed to the router, which forwards them either to external applications or to the storage manager which maintains box queue.

Scheduler picks a box for execution and passes it to the multi-threaded box processor.

QoS monitor, monitors system performance and activates the load shedder when it detects poor system performance.

QoS, Queue Management

Aurora attempts to maximize the perceived QoS provided by application as 2-D normalized QoS vs attribute (eg. delay, tuple drop, value) graphs.

Aurora manages one queue at the output of each box, which is shared by all successor boxes.

To allow Aurora to scale up arbitrarily, each queue is stored in disk storage. Blocks in main memory are evicted based on priority.

Priority assignment

Based on expected utility under the current system state

using feedback mechanism which continuously observes the performance of the system and dynamically reassigns priorities to outputs

Train scheduling : To avoid overhead of inter-box bypass and context switching between the boxes.

Overload

static analysis : throughput * selectivity < input

dynamic analysis : based on QoS of delay

reduces the number of tuples being processed via load shedding

Assumption : application is coded such that it tolerates missing tuples from data source because of communication failures

Load Shedding

Based on QoS graph, tuples which results in minimum decreases in overall QoS is identified

This interval is converted into a filter predicate which is passed upstream until split-point.

If it is difficult to calculate inverse function for upstream box, filter is estimated using trial-and-error which is passed downstream from split point to that box.

Runtime Optimization of Join Location*

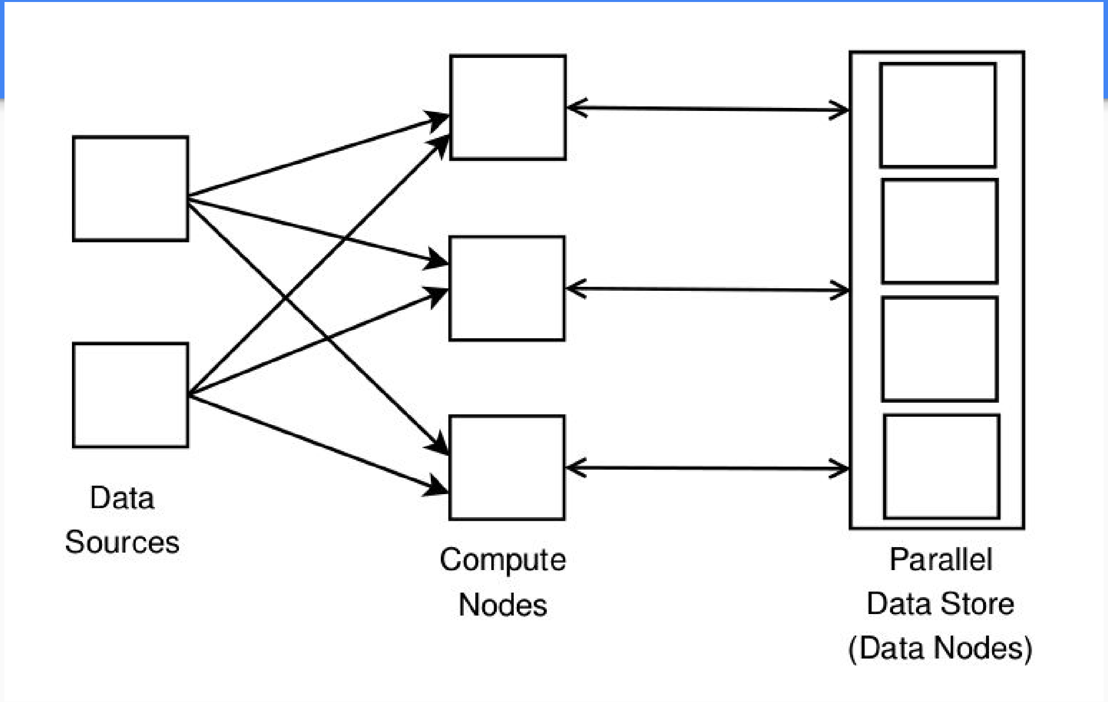
* Content and figures explaining this are taken from Runtime Optimization of Join Location in Parallel Data Management Systems Bikash Chandra, S. Sudarshan VLDB 2017

Runtime Optimization of Join Location *

Parallel systems often need to join a streaming relation with data indexed in a parallel data storage system and compute user defined function (UDF) on the joined tuples.

It can be done using reduce side joins (at data node) or map side join (at compute node)

Here we discuss techniques to make runtime decisions between the two options on a per key basis.


Data
Sources
Compute
Nodes
Parallel
Data Store
(Data Nodes)

Function of the form f (k, p);

where “k” is the key of streaming tuple. “p” is a list of parameters to the function. If p is empty then function can return stored value. (computing the join) and does not compute any UDF.

functions f (k, p) can be invoked in the following ways.

Data Request : compute the function at the compute node.

Compute Request : compute the function at the data node.

Ski-Rental Algorithm

model the decision problem between data request and compute request as classical online ski-rental problem.

Compute requests can be considered as renting and fetching the values locally can be considered as buying.

However our problem is significantly different in many key aspects.

Recurring Costs After Buying

Let the recurring cost after buying is b r ; cost of buying is b; cost of renting is r and m is number of accesses at which we buy.

We should keep renting as long as the cumulative renting cost is less than the cumulative buying cost.

$r * m <= b + br * m$

Caching

Algorithm 1 : skiRentalCaching

Inputs: k = data item key

```
updateBenefit(k)
updateCounter(k)
if  k ∈ mCache then
    v ← mCache.get(k)
    localComputeQueue.add(f,k,p,v)
else if  k ∈ diskcache then
    v ← dCache.get(k)
    localComputeQueue.add(f,k,p,v)
    condCacheInMemory(k,v,itemSize)
else
    if counter(k) ≤ b/(r − b_rM) then
        computeQueue.add(f,k,v,p)
    else
        if condCacheInMemory(k,ϕ,itemSize) then
            dataQueue.add(mCache,f,k,p)
        else if  counter(k) ≤ b/(r − b_rD)  then
            computeQueue.add(f,k,v,p)
        else
            dataQueue.add(dCache,f,k,p)
        end if
    end if
end if
```

Updates to the Data Store

tuples may get updated and bought items can no longer be used.

1. Data node can broadcast to all compute nodes.
2. with each response to a compute request, the data node also sends the timestamp when the item was last updated. The compute node tracks the timestamp of the last compute request for each data item. If the timestamp gets updated between two compute requests, the counter for the data item is reset.

Putting All Together

$tCompute = max(tDiskj, ((s_k + s_p + s_{cv})/netBw_{ij}), tc_j)$
$tFetch = max(tDisk_j, ((s_k + s_v)/netBw_{ij}))$
$tRecMem = tc_i$
$tRecDisk = max(tc_i, tDisk_i)$
Maximum is used because multiple invocations run concurrently and the disk and network access of these overlap with each other.

Disk and CPU costs are measured at run-time. Network bandwidth is measured before execution. To accommodate changes to these values, exponential smoothing is done

Multiple Joins : feeding the result of one join as the input of the next join

Prefetching : Requests to the data store are usually blocking.

Balancing of Computation : use gradient descent to compute the number of tuples to be sent to compute node from data node for which the maximum of the costs is minimum.

Bibliography

These slides are based of work in following papers.

1) Tyler Akidau, Alex Balikov, Kaya Bekiroglu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, Sam Whittle. MillWheel: Fault-Tolerant Stream Processing at Internet Scale. *Procs. VLDB 2013 Pages 1033-1044*

2) Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein,Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. *SIGMOD '03 Pages 668-668*

3) Mohamed A. Soliman, Lyublena Antova, Venkatesh Raghavan, Amr El-Helw, Zhongxian Gu, Entong Shen, George C. Caragea, Carlos Garcia-Alvarado, Foyzur Rahman, Michalis Petropoulos, Florian Waas, Sivaramakrishnan Narayanan, Konstantinos Krikellas, Rhonda Baldwin. Orca: A Modular Query Optimizer Architecture for Big Data *SIGMOD '14 Pages 337-348*

4) Prasan Roy,Chapter 2 -Rule-Based Query Optimization using the Volcano Framework *PhD thesis, IIT Bombay, 2000*

5) Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, Stan Zdonik Monitoring Streams – A New Class of Data Management Applications. *VLDB '02 Pages 215-226*

6) Anil Shanbhag and S. Sudarshan Optimizing Join Enumeration in Transformation-based Query Optimizers *Proceedings of the VLDB Endowment 2014 Pages 1243-1254*

7) Jingren Zhou, Per-Ake Larson, Ronnie Chaiken. Incorporating Partitioning and Parallel Plans into the SCOPE Optimizer *Data Engineering (ICDE), 2010 IEEE 26th International Conference Pages*