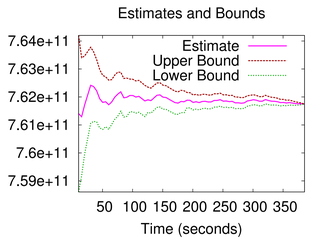
PF-OLA: A High-Performance Framework for Parallel On-Line Aggregation

Chengjie Qin, Florin Rusu
Email: *cqin3@ucmerced.edu frusu@ucmerced.edu*

OLA – On-Line Aggregation

Requirements

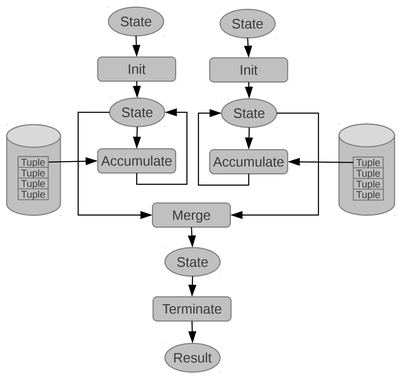
1. *Provide estimates to the final result.*
2. *The width of the confidence bounds should decrease fast enough.*
3. *The estimation model works in a parallel environment.*
4. *The estimation does not incur huge time overhead.*



User-Defined Aggregates(UDA)

UDA Basic Interface

Our on-line aggregation framework is implemented upon UDAs. UDAs represent a mechanism to extend the functionality of a database system with application-specific aggregate operators, e.g., data mining. A UDA is typically implemented as a class with a standard interface defining four methods: `Init`, `Accumulate`, `Merge`, and `Terminate`. These methods operate on the `state` of the aggregate which is also part of the class.While the interface is standard, the user has complete freedom when defining the `state` and implementing the methods. The figure below shows how the four methods work with each other.



UDA Extension

There are three extensions that need to be added to the basic UDA to make it fit for usage in parallel on-line aggregation. They are communication extension, estimation model extension, and user side estimate method extension.

1. The first extension handles the communication problem. We apply the same principles at the core of UDA and extend the UDA interface with methods to `Serialize/Deserialize` the UDA state.
2. The second extension is specifically targeted at estimation modeling for on-line aggregation. Two methods that need to be added: `EstimatorTerminate` and `EstimatorMerge`. `EstimatorTerminate` computes a local estimator at each node. `EstimatorMerge` puts together in a single UDA the estimators computed by `EstimatorTerminate` at each node.
3. The third extension we add to the UDA interface is the `Estimate` method. It is invoked by the user application on the UDA returned by the framework as a result of an estimation request. In the case of on-line aggregation, `Estimate` computes an estimator for the aggregate result and corresponding confidence bounds.

Method	Usage

`Init ()`
`Accumulate (Item` *d*`)`
`Merge (UDA` *input*$_1$`,`
`UDA` *input*$_2$`, UDA` *output*`)`
`Terminate ()`
	Basic interface

`Serialize ()` `Deserialize ()`	Transfer UDA state across processes

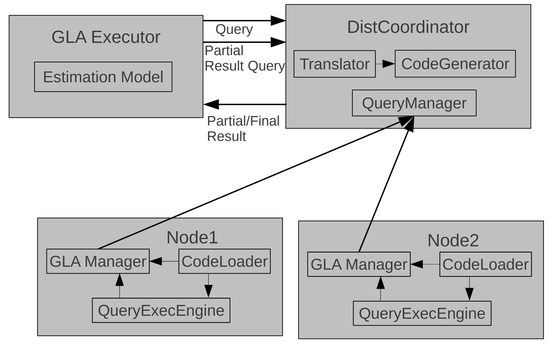
`EstimatorTerminate ()`
`EstimatorMerge (UDA` *input*$_1$`,`
`UDA` *input*$_2$`, UDA` *output*`)`
	Partial aggregate computation

`Estimate (`*estimator*`,` *lower*`,` *upper*`,` *confidence*`)`	On-line estimation

The table above summarizes the extended UDA interface we propose for parallel on-line aggregation.

GLADE – GLA Distributed Engine

GLADE is a parallel processing system optimized for the execution of Generalized Linear Aggregates (GLA). The basic components of GLADE are shown below.



Estimation

LEMMA 1. X is an unbiased estimator for the aggregation problem, i.e., $E[X] = \sum_{d \in D, cond(d)} \text{func}(d)$, where $E[X]$ is the expectation of X. The variance of X is equal to:

$$\text{Var}[X] = \frac{|D| - |S|}{(|D|-1)|S|} \left[|D| \sum_{d \in D, cond(d)} func^2(d) - \left(\sum_{d \in D, cond(d)} func(d) \right)^2 \right] \quad (1)$$

LEMMA 2. The estimator

$$Est_{Var(X)} = \frac{|D|(|D|-|S|)}{|S|^2(|S|-1)} \left[|S| \sum_{s \in S, cond(s)} func^2(s) - \left(\sum_{s \in S, cond(s)} func(s) \right)^2 \right] \quad (2)$$

is an unbiased estimator for the variance in Equation 1.

- `Generic Sampling Estimator`
  Consider the dataset D to have a single partition sorted in random order. The number of items in D (size of D) is $|D|$. While sequentially scanning D, any subset $S \subseteq D$ represents a random sample of size $|S|$ taken without replacement from D. We define an estimator for the aggregate as follows:
  $$X = \frac{|D|}{|S|} \sum_{s \in S, cond(s)} func(s) \quad (3)$$

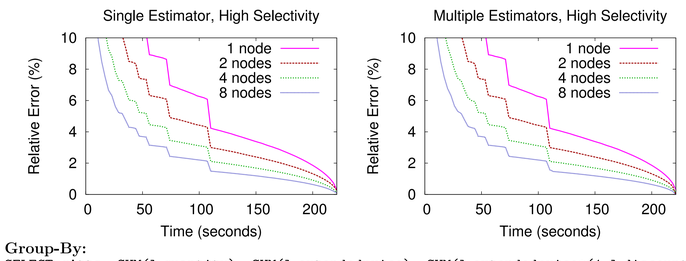
- `Single Estimator Sampling`
  In the PF-OLA framework, the dataset D is partitioned across N processing nodes, i.e., $D = D_1 \cup D_2 \cup \cdots \cup D_N$. A sample S_i, $1 \leq i \leq N$, is taken independently at each node. These samples are then put together in a sample $S = S_1 \cup S_2 \cup \cdots \cup S_N$ over the entire dataset D.

- `Multiple Estimators Sampling`
  For multiple estimators, the aggregate $\sum_{d \in D, cond(d)} func(d)$ can be decomposed as $\sum_{i=1}^{N} \sum_{d \in D_i, cond(d)} func(d)$, with each node computing the sum over the local data partition in the first stage followed by summing-up the local results to get the overall result in the second stage. An estimator $X_i = \frac{|D_i|}{|S_i|} \sum_{s \in S_i, cond(s)} func(s)$ is defined for each partition based on the generic sampling estimator in Equation (3).

Experiments

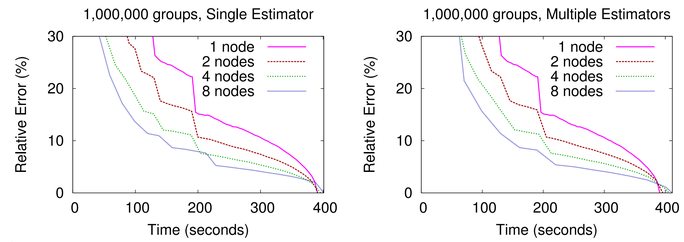
System. We executed all our experiments on a 9-node shared nothing cluster. Each node has 2 AMD Opteron 8-core processors for a total of 16 cores running at 2GHz, 16GB of memory, 4 1TB hard-drives, and runs Ubuntu 10.10 64-bit.
Data. The dataset used in our experiments is TPC-H scale **8,000** (**8*TB***).We generated 8 1TB independent TPC-H scale 1 instances, one on every worker node.
Methodology. Relative confidence bounds width ($\frac{high - low}{estimate}$) is measured as the ratio between the difference of the confidence interval extremes and the estimate. The bounds are computed at 95% confidence level and are depicted as percentage from the estimate.
Aggregation:

```
SELECT SUM(l_extendedprice*l_discount) FROM lineitem
WHERE l_shipdate between [date1,date2] AND l_discount between [0.02,0.03] AND l_quantity=1
```



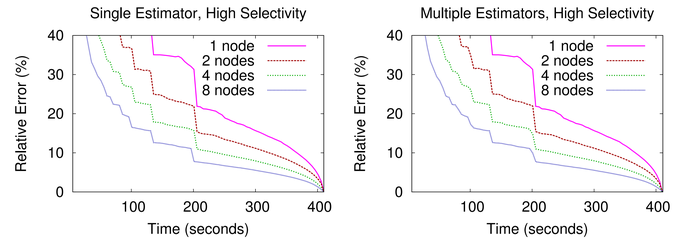
Group-By:

```
SELECT gAtts, SUM(l_quantity), SUM(l_extendedprice), SUM(l_extendedprice*(1-l_discount)),
 SUM(l_extendedprice*(1-l_discount)*(1+l_tax)) FROM lineitem
WHERE l_shipdate between ['1998-09-01','1998-12-01'] GROUP BY gAtts
```



Join:

```
SELECT n_name, SUM(l_quantity), SUM(l_extendedprice), SUM(l_extendedprice*(1-l_discount)),
 SUM(l_extendedprice*(1-l_discount)*(1+l_tax)) FROM lineitem, supplier, nation
WHERE l_shipdate between [date1,date2] AND l_discount between [0.02,0.03] AND
 l_quantity = 1 AND l_suppkey = s_suppkey AND s_nationkey = n_nationkey GROUP BY n_name
```



Query	Execution Time (seconds)		
	`No estimation`	`Single estimator`	`Multiple estimators`
`Aggregate`	222	222	222
`Group`$_{small}$	344	345	344
`Group`$_{large}$	404	407	407
`Join`	409	411	411

The table above contains the execution time for the 8 node configuration. It confirms that on-line aggregation can be enabled for any query at virtually no overhead. It is the asynchronous overlapping of execution and estimation in the GLADE parallel multi-query processing system that makes this possible.