E0 261 - Database Management Systems
Handling Data Updates in SpillBound

Sheshadri K. R.
Ph.d. (CDS)
06-02-02-10-12-18-1-16336

Shriram R.
M. Tech. (CDS)
06-02-01-10-51-18-1-15763

December 7, 2018

1 Introduction

SpillBound algorithm [2] is proven to be robust against incorrect selectivity estimates and skewed data distribution. However, the algorithm is not known to be immune to large data updates in relations. We have investigated the robustness of SpillBound in handling data updates through experimentation on different database sizes. We have proposed incremental algorithms which uses the current information about contours and plan bouquet [1] for construction of contours for the updated version of data.

2 SpillBound Evaluation

We have evaluated SpillBound for robustness to data inserts and deletes by using the old contours and plan bouquets and also with the incremental algorithm. The experimental setup, results and analysis are described in the following sections in detail.

2.1 Experimental Setup

We have used Postgres-9.4.1 database with added functionality of selectivity injection and foreign plan costing. TPC-DS *datagen* tool was used to create multiple copies of 1GB database. Primary and Foreign key constraints and Indexes were created in all databases wherever applicable. *CODD* tool was then used to scale the metadata of the 1GB databases to different sizes. We have considered 100GB as the base size (for old contour and plan bouquet) and applied SpillBound on other sizes ranging from 10GB to 150GB.

Note that the SpillBound was not directly implemented inside Postgres but rather written as a Java function that emulates the Algorithm. We have used FPC to identify the maximum selectivity that can be learnt by plans in old bouquet with a given cost budget. This information was used to apply the old plans on different database sizes. Logarithmic selectivity space was used for all the queries. We have only considered 2-D queries in this evaluation.

2.2 Experimental Results

Figures 1(a) and 1(b), provide a measure of Average SO (ASO) and Maximum SO (MSO) for database of 20GB, 50GB and 140GB sizes respectively over different TPC-DS queries. The X-Axis is the TPC-DS queries, the Y-Axis is ASO for Figure 1(a), and MSO for Figure 1(b). The general observed trend is that, the MSO and the ASO obtained by executing the SpillBound algorithm is lower when compared to performing Foreign Plan Costing (FPC) based SpillBound with respect to 100GB database size.

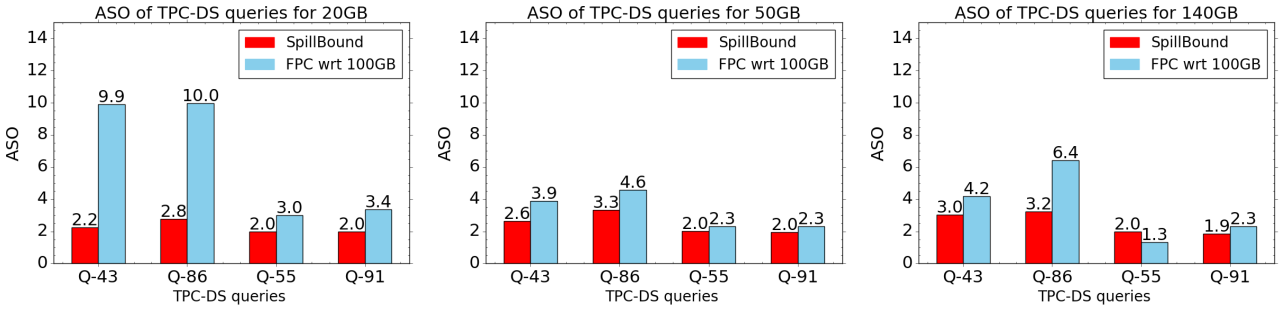

Figure 1(a) : Average SO of 20GB, 50GB and 140GB over different TPC queries

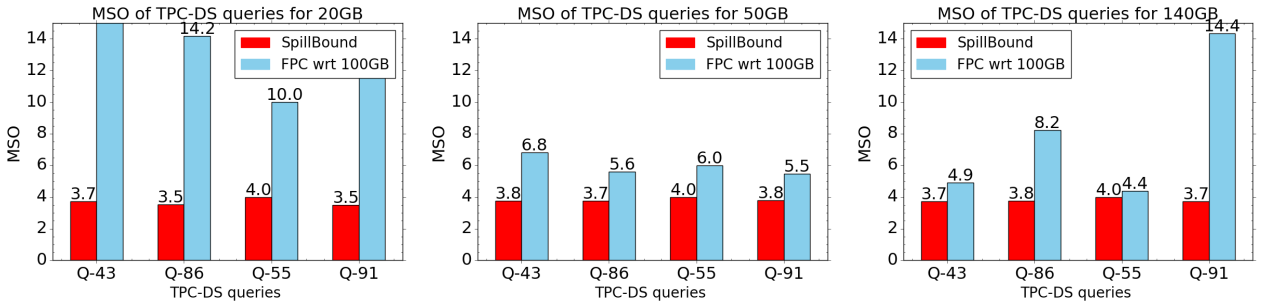

Figure 1(b) : Maximum SO of 20GB, 50GB and 140GB over different TPC queries

Figure 2. below shows the execution of various TPC-DS queries over a fixed reference database size of 100GB. The X-Axis is the databases of different sizes, the Y-Axis is the MSO. As we can see, for the Q-91, the MSO increases as the database shrinks and breaches the value 10 (Theoretical bound for MSO in 2D) at 20GB in the decreased size case, and at 140GB in the increased size case.

For the Q-43, it crosses the MSO value 10 at 30GB in the decreased size case. However, the MSO does not rise for the increased sizes case as it did in case of decreasing database sizes. We noticed that, the plan structure mostly remains the same 100GB onwards, in fact when we examined the plan structure we found that 100GB and 150GB databases both share the same plan structure.

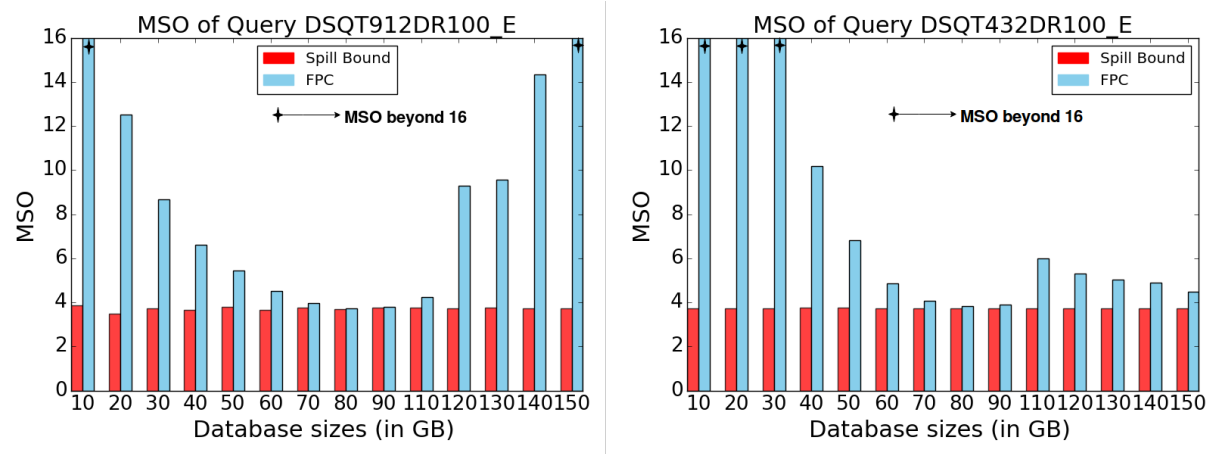

Figure 2: MSO of different database sizes against Q-91 and Q-43

2.3 Analysis

The Figure 3. below shows the plan structure for a database of size 100GB and 30GB, for the TPC-43 query (we can see the MSO of 30GB in the above Figure 2.)

The Bitmap Heap Scan of the store_sales relation in 100GB changes to Index Scan once the database size changes to 30GB. So evidently, when the size reduces we can see that the plan structure changes, therefore, using the 100GB plans for executing the queries of 30GB will shoot up the MSO. Using this information we can observe till what extent can we re-use the plans of 100GB and still have tolerable MSO (< 10 for the case of 2D).

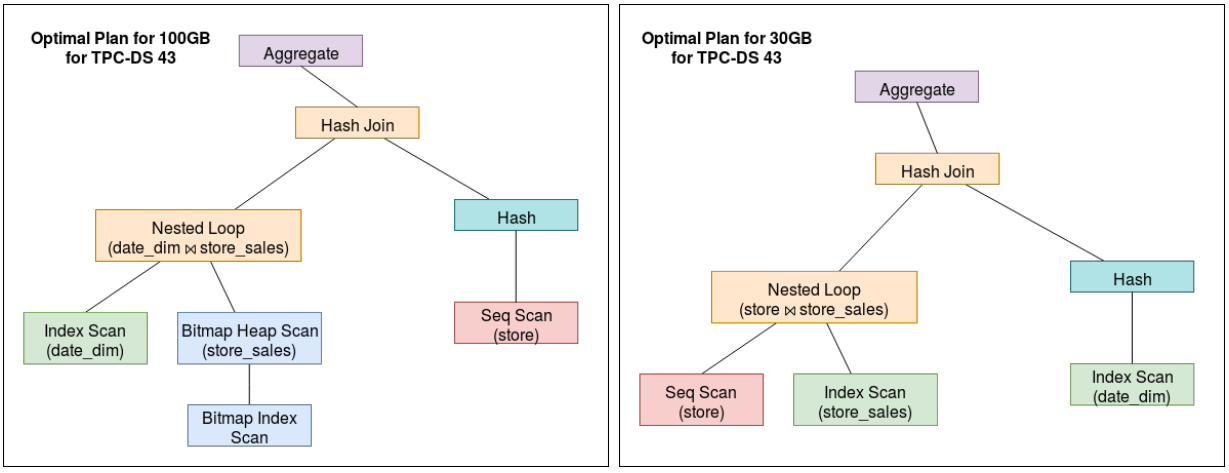

Figure 3: Plan structure TPC-DS Q-43 for 100GB and 30GB

3 Incremental Techniques

3.1 Algorithms

We only describe the incremental algorithms for 2-D scenario. The fundamental idea behind these algorithms is to exploit the geometric properties of the contours. We have analyzed the contour shapes given by NEXUS [3] algorithm and found that most of the contours have highly piecewise linear shape in the logarithmic scale. This fact can be observed in the following figures.

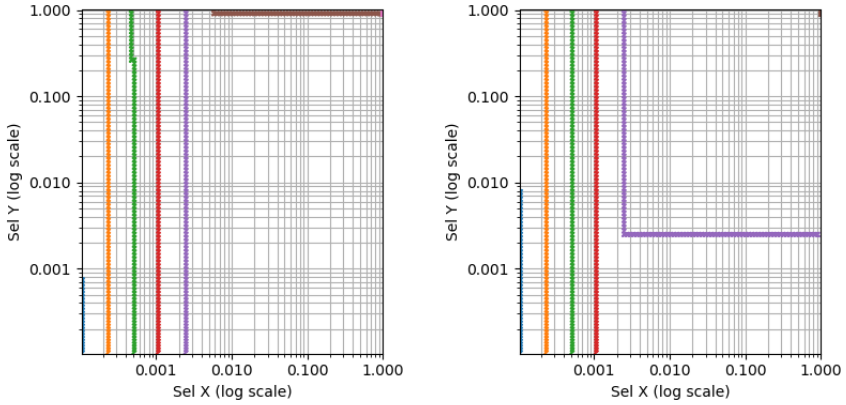

Figure 4: Contour structures for TPC-DS 91 100 GB and 50 GB

Let $C = \{c_1, c_2, ..., c_n\}$ denote the set of original contours for a given database (E.g. 100 GB) and $P_i = \{p_1, p_2, ..., p_{h_i}\}$ denote the set of points in contour c_i. The points are sorted by x-value (ascending) and by y-value (descending) order. Let $C' = \{c'_1, c'_2, ..., c'_m\}$ denote the set of contours in the target database (E.g. 50 GB) and $P'_i = \{p'_1, p'_2, ..., p'_{h'_i}\}$ denote the set of points in contour c'_i sorted in the same way as described earlier.

Algorithm 1 DeltaNxt (C, P)

```
1:  for i : 1 to min(m, n) do
2:      pts ← ()
3:      next_point ← left intercept of P'_i
4:      j ← 0
5:      delta_x ← 1
6:      delta_y ← 1
7:      while 0 <= next_point[x] < res and 0 <= next_point[y] < res do
8:          append(pts, next_point)
9:          if j + 1 < h_i then
10:             delta_x ← P_i[j + 1][x] − P_i[j + 1][x]
11:             delta_y ← P_i[j + 1][y] − P_i[j + 1][y]
12:             j = j + 1
13:         next_point[x] ← next_point[x] + delta_x
14:         next_point[y] ← next_point[y] + delta_y
15:     P'_i ← pts
```

The above algorithm *DeltaNxt* is used to find the set of points in $P'_1, P'_2, ..., P'_{min(m,n)}$ using the information of C and P. The algorithm works by finding the difference in selectivity value between two consecutive points in P_i and uses this information to find the next point in P'_i.

The left intercept of P'_i is obtained using binary search adopted from NEXUS. The cost associated with C' can be obtained trivially by getting the *min_cost* and *max_cost* of ESS in the target database. Note that the contour point sets $P'_{min(m,n)+1}, P'_{min(m,n)+2}, ..., P'_m$ has to be found using NEXUS algorithm. The set of plans associated with each point in C' is obtained by the *Pincer* algorithm described below,

Algorithm 2 Pincer (left, right)

```
1:  if left > right then
2:      return {}
3:  if left == right then
4:      return (left, opt_plan(left))
5:  if opt_plan(left) == opt_plan(right) then
6:      t ← {}; p ← opt_plan(left)
7:      for i : left to right do
8:          append(t, (i, p))
9:      return t
10: else
11:     mid_pt ← (left+right)/2
12:     lp ← Pincer(left, mid_pt); rp ← Pincer(mid_pt + 1, right)
13:     return append(lp, rp)
```

The *Pincer* algorithm takes two contour points (i.e.) *left* and *right* as input and returns the plan associated with each point on the contour between *left* and *right*. The initial input to *Pincer* for a contour c'_i is $P'_i[0]$ and $P'_i[h'_i]$. The algorithm works on the assumption that plans span contiguous set of points on a given contour.

Algorithm 3 LinearNxt (C, P, k)

```
1:  for i : 1 to min(m, n) do
2:      be ← base_end_points; te ← target_end_points; pts ← ()
3:      step_size ← max(2, h_i/(k−1))
4:      samples ← {p_1, p_{1+step_size}, p_{1+2*step_size}, ..., p_{h_i}}
5:      pieces ← {}
6:      for j : 0 to |samples| − 1 do
7:          (x, y) ← (samples[j + 1][x] − samples[j][x], samples[j + 1][y] − samples[j][y])
8:          append(pieces, (0, x)) if x > 0; append(pieces, (1, y)) if y < 0
9:      alpha ← euclidean_dist(te)/euclidean_dist(be)
10:     for j in pieces do
11:         next_list ← scale(j, alpha)
12:         append(pts, next_list.filter(within_space))
13:         break if next_list.contains(beyond_space)
14:     P'_i ← pts
```

We have developed another algorithm *LinearNxt*, shown above, for contour point identification based on piecewise linear approach. This algorithm works by sampling a base contour and then construct linear curve for each consecutive pair of samples. These curves are scaled based on the estimated length of target contours and the points that lie within the ESS space are then added to the target contour.

3.2 Experimental Results

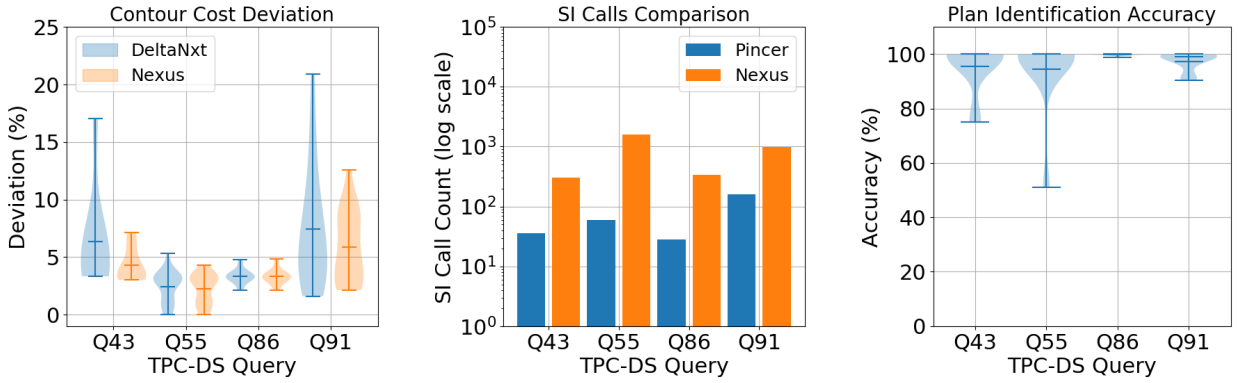

Figure 5: Plots From Experimental Evaluation of DeltaNxt and Pincer

The incremental algorithms were tested on 5 different 2D TPC-DS queries. Base data for *DeltaNxt* was taken from the 100 GB database and was used to identify contour points for 50, 80, 120 and 150 GB databases. Results are plotted in Figure 5. for *DeltaNxt* and *Pincer*.

Figure 5(a) gives a violin plot of relative cost deviation of contour points from the ideal contour cost. It can be observed that the relative cost deviation is comparable between *DeltaNxt* and *NEXUS* algorithms. The number of Selectivity Injection (SI) calls made by *Pincer* is dramatically less than the *NEXUS* algorithm and can be seen in Figure 5(b). *Pincer* takes approximately $\frac{1}{10}$ of the *NEXUS* SI calls in most of the queries. This is a crucial factor of improvement since the SI calls made to the query optimizer are expensive. In terms of accuracy of plan identification, as shown in Figure 5(c), *Pincer* performs well in most of the cases with average accuracy around 90%. There are few outliers as seen in the case of TPC-DS 55. This is due a particular contour having alternating plans across the points.

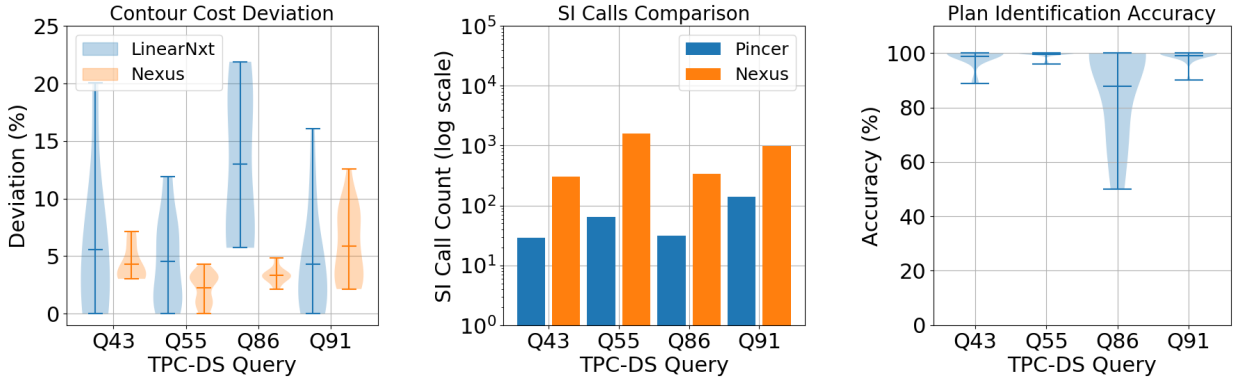

Figure 6: Plots From Experimental Evaluation of LinearNxt and Pincer

Figure 6. shows the experimental results for *LinearNxt* paired with *Pincer* in a similar setup as described above with sample size (k) set to 30. There is a increase in cost deviation compared to the *DeltaNxt* algorithm for some of the queries. This could be due to suboptimal value for the sample size. The SI call counts are comparable with *DeltaNxt* and Plan identification accuracy has improved for some queries.

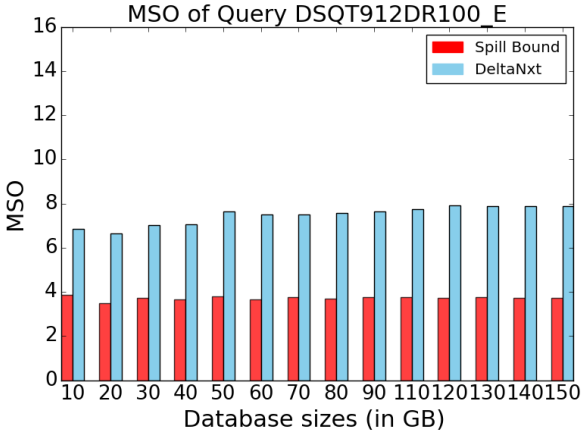

Figure 7: *DeltaNxt* MSO of different database sizes against TPC-91

Figure 7. shows the MSO for a range of database sizes using contour and plan information from *DeltaNxt*. It can be observed that the MSO is within the theoretical limit for all the sizes and comparatively better than old contour reuse strategy for extreme cases of size (E.g. 10 GB, 20 GB, 140GB etc.). MSO from *LinearNxt* has similar values and is not plotted for brevity.

4 Future Work

We would like to evaluate SpillBound for higher dimensions for its robustness. We can also extend the incremental techniques to higher dimensions and work on a theoretical guarantee for the MSO. We wish to explore more techniques based on Machine Learning and other geometry based approaches. We briefly define our ideas here and leave the implementation to future work. In Machine Learning techniques, we can employ supervised learning algorithms to learn the plan of each selectivity point. We can use factors such as cost, selectivity points, query parameters (number of tables, where conditions and so on), database size etc., as the training data and label each of the data point by a distinct plan. In this way, whenever we encounter a new database of a different size, we can map the best existing plan for a range of points in a given contour of new database.

5 Conclusion

We can conclude that the old contours and plans can be reused for a good magnitude of inserts (1.3x) and deletes (0.4x) in the case 2D without violating the theoretical bound for MSO. We have also observed that the incremental techniques can vastly reduce the optimizer call count by trading off some accuracy.

6 Acknowledgements

We thank Mr.Srinivas Karthik for his guidance and patience in explaining the subtelities involved in the project.

References

[1] Anshuman Dutt and Jayant R Haritsa. Plan bouquets: query processing without selectivity estimation. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1039–1050. ACM, 2014.

[2] Srinivas Karthik. Robust query processing. In *Data Engineering Workshops (ICDEW), 2016 IEEE 32nd International Conference on*, pages 226–230. IEEE, 2016.

[3] C Rajmohan. *Turbo-charging Plan Bouquet Identification*. PhD thesis, Indian Institute of Science Bangalore, 2015.