ASE 15 Query Tuning with 'plancost'

Diagnosing Query Plan Problems for Better Query Performance

This article examines the ASE command 'set statistics plancost on' and provides some simple guidelines for very effective query tuning.

By Rob Verschoor

It is a well-known fact that ASE 15 has a new query processing engine with many new algorithms for better query performance. However, many ASE users are not aware that ASE 15 also makes it much easier to diagnose query plan problems. This article shows how.

Estimates and Metrics

The central feature of this article is the ASE command **set statistics plancost on**. After executing it, ASE 15 displays the optimizer's estimates for every operator in the query plan along with the corresponding metrics from actual execution of the query. Query plan problems are often the result of incorrect optimizer estimates. With this information, it is now easy to see if the optimizer estimated correctly or not. In contrast, in ASE 12.x this information was simply not available. In ASE 15, it opens up entirely new ways of looking for query plan problems. The best part: this information is easy to use for DBAs and developers. With some simple guidelines, some very effective query tuning capabilities come within reach.

In the 'plancost' output (I'll use that as shorthand for "the query plan diagnostics printed for every query after running **set statistics plancost on**"), each query plan operator looks something like this:

```
TableScan
stores
(VA = 0)
r:522 er:507
l:14 el:13
p:0 ep:2
```

As you may have guessed, this is an example of a table scan on a table named **stores.** In this output, **er:** indicates the number of rows that the ASE query optimizer has estimated this operator will process (507 in this case), and **r:** the actual number of rows during execution (here, 522). **el:**, **l:**, **ep:** and **p:** indicate the estimated and actual logical and physical I/O, respectively. For general understanding of ASE query plans, only **er:/r:** is relevant since the logical and physical I/O estimates are derived from the estimated rowcount. For other operators, **p:** (actual physical I/O) is the relevant property instead, and we'll look at this later in this article.

Finding Discrepancies

Now, when you suspect a query to have a non-optimal query plan, you should look for query plan operators that show wildly different numbers for **er:** and **r:**. We're not looking for a factor 10 or

Rob Verschoor is a Technical Director and Data Management Evangelist at Sybase, located in EMEA. Rob's focus is on Adaptive Server Enterprise, ASE Cluster Edition, Replication Server, and Sybase IQ; he has a true passion for query performance and stressing ASE to its limits. He can be reached at robv@sybase.com

even 100 here, but for larger differences. Also, it is important to remember that, when looking for query plan issues, you should only look at **IndexScan** or **TableScan** operators: discrepancies found here may 'bubble' upwards in the query plan and get amplified or reduced in the process, making them largely useless.

So, let's assume such a big discrepancy between estimated rows(**er:**) and actual rows (**r:**) is indeed observed. The question then must be: why did the query optimizer so badly over- or under-estimate the number of rows to be processed by this query plan operator? When looking at such a case, the first thing to consider is that, for IndexScan and TableScan operators, the optimizer estimates for the **er:** numbers are based on the available statistics for the table being queried. Naturally, a possible cause is that the statistics are outdated or otherwise inaccurate; if so, it is not surprising that the optimizer gets the estimates wrong.

Update Statistics

Perhaps more interesting than the reason behind such a plancost discrepancy is what you can do about it. The answer is simple: you should consider running **update statistics** for tables where the plancost information reveals big discrepancies between the **er:** and **r:** numbers. Then, re-run the query, and check if the **er:** number becomes more in line with the actual execution rowcount (**r:**). If the **er:** number is positively affected by updating the table's statistics, this confirms that the statistics quality was playing a role in the incorrect optimizer estimates. If, on the other hand, **update statistics** makes no difference, you should try running **update index statistics** or generate more histogram steps (**update [index] statistics using** nnn **values**).

When the column(s) for which statistics are created may contain many duplicate values, better statistics might be created by setting the configuration parameter **'histogram tuning factor'** to a larger value than the default. This has the effect of better reflecting the duplicate values in the histograms by means of so-called 'frequency cells' (see the ASE documentation for more details).

If updating the statistics has a positive effect, then hopefully this means the query plan changes along in such a way that the query executes faster too (after all, you're supposed to be performing this type of analysis only for queries where performance is less than you expect or require; if it runs fast enough for your needs anyway, who cares about the optimizer's estimates?)

"The possibility to spot possible bad optimizer estimates, and to verify whether statistics quality affects such estimates, is a big step forward for diagnosing query plan problems in ASE 15: query plan debugging has become a more manageable activity, and less of a black art."

Additional Tuning Tips

Lastly, if **update statistics** has no positive effect at all, just verify that the existing index(es) on the table are the right ones for the query being executed: for example, an index could have a sub-optimal column ordering for the query being executed.

The possibility to spot possible bad optimizer estimates, and to verify whether statistics quality affects such estimates, is a big step forward for diagnosing query plan problems in ASE 15: query plan debugging has become a more manageable activity, and less of a black art. As a DBA, this makes you more effective.

Note that there can still be some remaining cases where, whatever you do, the optimizer's estimates just will not get any closer to the execution-time metrics. That possibility hasn't changed; but with the ASE 15 'plancost' information, you will have to spend much less time and effort to arrive at that conclusion compared to ASE 12.x.

One last remark about the 'plancost' output. Except for small and simple queries, the plancost output quickly becomes too wide to fit on a display screen or a sheet of printing paper. For such cases, enable traceflag 9528 (as well as 3604) instead: this causes the plancost information to be printed in a narrower format similar to the showplan output, fitting much better on a page, as follows (note that **er:** and **r:** are shown as **est:** and **act:**, respectively):

```
| | |TableScan (VA = 0, LSTATE_CLOSED)
| | |stores
| | |  Rows: act:522 est:507
| | |  Lios: act:14 est:13
| | |  Pios: act:0 est:2
```

Conclusion

In conclusion, knowing how to use the 'plancost' output will allow you to be more effective in identifying and troubleshooting query plan issues in ASE 15 than has ever been possible in ASE 12.x. ■