Query Patroller Setup
A customer example

IBM Corporation, 2005

The Problem

In this article, we will present an example of setting up Query Patroller at a customer environment. The customer has a 3TB data warehouse and before configuring Query Patroller, there was a serious problem with query concurrency. The queries are running at the same time ETL and Load jobs are running. Although this is something we will correct in the future, by separating the CPUs that run ETL and the CPUs that run the queries, at this time, all jobs and queries are running on the same SMP AIX server, and this concurrency of users and ETL work is an issue.

Below, we present the way we are setting up the environment, in order to configure Query Patroller based on the queries and the features of this particular customer example.

Configuration

- AIX Regatta p690 server with 16 CPUs, and 64GB of RAM.
- DB2 UDB V8.2 with fixpack 9A and DPF
- DB2 has 16 data partitions and one catalog partition
- Data Stage and Meta Stage products from Ascential for ETL processing
- Query Patroller

Workload Characteristics

The system is running ETL load scripts during the night, starting from 5PM up to 11AM in the morning. In the morning hours, from 8 to 10 AM, there is peak query activity for end users, with ad hoc queries primarily. The number of concurrent queries is usually 4 to 5.

Steps for Query Patroller configuration

In order to setup the Query Patroller we followed the following steps :

1. Gather historical analysis for all queries for 3 days
2. From the historical analysis window, Select all rows and export them to a Delimited file
3. Import the delimited file to Excel spreadsheet and use the ”text to column” function to convert the data into columns in the spreadsheet.
4. Check and sort the cost of all queries and create a distribution, using the number of queries and their cost range
5. Then start to create the query classes based on the findings above. Preferably leave the small queries not intercepted.

Here is the extract from the historical analysis, which is imported into the spreadsheet. The total number of queries is 57113. Below we just copy a small portion of the query extract for the purpose of demonstration. The output spreadsheet is the result of step 3 described above.

Query number			Cost	Priority	Query Start Time	Query Finish Time
161541	EDA	Ran successfully	29.889	500	11/12/2005 10:26	11/12/2005 10:26
161542	EDA	Ran successfully	29.889	500	11/12/2005 10:27	11/12/2005 10:27
161447	EDA	Ran successfully	29.889	500	11/12/2005 9:57	11/12/2005 9:57
161448	EDA	Ran successfully	29.889	500	11/12/2005 9:57	11/12/2005 9:57
161577	EDA	Ran successfully	29.889	500	11/12/2005 10:28	11/12/2005 10:28
161646	EDA	Ran successfully	29.889	500	11/12/2005 10:29	11/12/2005 10:29

I61647	EDA	Ran successfully	29.889	500	I1/12/2005 10:30	I1/12/2005 10:30
I61658	EDA	Ran successfully	29.889	500	I1/12/2005 10:31	I1/12/2005 10:31
I61704	EDA	Ran successfully	29.889	500	I1/12/2005 10:32	I1/12/2005 10:32
I61717	EDA	Ran successfully	29.889	500	I1/12/2005 10:36	I1/12/2005 10:36
I61718	EDA	Ran successfully	29.889	500	I1/12/2005 10:37	I1/12/2005 10:37

After we import the historical analysis to a spreadsheet as above, we start the filtering and classification of queries. In this example, we identify that the first 50343 queries have a cost of less than 30 timerons. These queries are mostly INSERTs to metadata tables of Webfocus which is an end user application, as well as other small operations of INSERT / DELETE / UPDATEs. In general 88% of activity in the system is INSERT / DELETE / UPDATE activity. QP should not intercept these queries and we will exclude them from the Query Patroller interception.

This is an example where with Query Patroller we were able to identify a problematic behavior in the system. Discovering that 88% of system activity is heavy transactional workload, we performed further analysis and we were able to tune the end user application and eliminate the large amount of this workload by changing the application code and setup.

Regardless of the kind of optimization and tuning that has to be done for the INSERT / UPDATE / DELETE operations, we need to eliminate them from out Query Patroller analysis at this point, and focus on just the more expensive activities. Therefore, we create a smaller version of the spreadsheet with only 6770 rows in our spreadsheet with activities of higher cost and response time. The rows we eliminated represent transactions that have a cost of a maximum of 30 timerons.

Based on the analysis we did over a 3-day period, here is the statistical numbers we have with the remaining 6770 transactions: (the numbers below are in timerons)

Number of queries	**Range of Cost in timerons**
106	30-113
2254	100-300
1486	300-2K
2579	2K-10K
165	10K-100K
173	100K-1M
17	1M-10M

At this point we need to further formulate this classification and limit the number of query classes to no more than three or four. If the query classes are more than four, we may end up create additional overhead to the Query Patroller tool which is not necessary.

So we created only 4 classes by combining the classes above that have a similar amount of queries. For example, the number of queries with cost between 100-300 timerons is about 2,2K and the number of queries with cost between 300-2K timerons is about 1,4K which is close. So we combined the two to one single class because these two classes of queries have the same frequency. We did the same for queries with cost between 10-100K timerons which are 165, a number similar to 173 for 100K-1M timeron cost queries, so we combined them both in a single class.

We also decided that all queries below 100 timerons will not be intercepted.

We made the assumption that the maximum number of queries the system should handle concurrently at this point is 10. Normally a system should be able to run 20-30 queries concurrently. In this system we will only try 10 because of the ETL running together with queries in the same SMP server, sharing the same CPU cycles.

Based on this number we aligned the percentage of number of queries to the allocated number of queries allowed to run in this class. Nevertheless, because the smaller running queries finish faster, we did not allocate as many for the first class of smaller queries, as it would proportionally be logical. This is because of the shorter runtime of these queries.

All the assumptions and class configuration always needs monitoring and further tuning maybe required.

Here are the classes we created :

Proposed Query Number of queries	**Classes Percentage of queries in this range**	**Range of Cost in Timerons**	**Max number of queries allowed in QP class**
3740	56%	100-2K	
2579	38%	2K-10K	
338	5%	10K-1M	
17	0.2%	1M-10M	

In our next step, we choose from within Query Patroller historical analysis, five queries from different classes in order to test them run with and without Query Patroller intercepting them.

We looked at the history of queries and we picked some of the most frequently run queries during the workloads. We picked samples from all 4 classes of queries we defined. Here is the mix. This is just a sample mix, and we can do additional samples for further testing.

1 query with cost less than 130K timerons
5 qeuries with cost less than 2K timerons
2 queries with cost between 2K-10K timerons
1 query with cost between 10K-1M timerons
1 query with cost between 1M-10M timerons

We run this list of queries using the db2batch command in order to capture the details of the elapsed time and performance for each. We run them with Query Patroller intercepting and with out Query Patroller intercepting any query. Here are the results.

Query cost in timerons	**Class of query**	**Elapsed time with QP**	**Elapsed time without QP**
130	No intercept	8.154	4.218
225	Class 100-2K	0.002	0.002
833	Class 100-2K	0.723	0.346
1269	Class 100-2K	6.523	4.965
1648	Class 100-2K	0.976	0.078
1783	Class 100-2K	8.537	16.093
2333	Class 2K-10K	0.660	0.319
2649	Class 2K-10K	309.914	318.256
2963	Class 2K-10K	285.594	281.508
1M	Class 10K-1M	115.920	110.960
2M	Class 1M-10M	0.003	0.002

We notice that for most of the large and medium queries Query Patroller provided some improvement.

References

DB2 manuals: http://www-306.ibm.com/software/data/db2/udb/support/manualsv8.html
QP manual: ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/db2dwe81.pdf