DB2 Query Parallelism : Staging and Implementation

Yun Wang
IBM Santa Teresa Laboratory
San Jose, CA 95161
wang@stlvm14.vnet.ibm.com

Abstract

Intraquery parallelism has been recognized as a key requirement for DB2 major customers to allow data intensive queries with acceptable response time. DB2 delivers intraquery parallelism in three stages since DB2 Version 3 Release 1 in 1993. DB2 intraquery parallelism requires extensions on many DB2 components, such as optimizer, query executor, buffer manager, ...etc. We briefly discuss the major issues in DB2 intraquery parallelism and the mechanisms used in our implementation.

1 Introduction

The intraquery parallelism has been planned and implemented in three stages.

- The first stage is *Query I/O Parallelism* which allows a single query to have a parallel execution plan. A parallel execution plan may spawn multiple pseudo subtasks, each subtask can access a table, join tables, compute aggregates and issue synchronous as well as asynchronous I/O requests. All the subtasks are scheduled and executed within a single MVS task. A single query can use up to 100% CPU time of a single processor with many I/O requests performed at the same time.
- The second stage is *CPU Parallelism.* which spawns multiple MVS tasks for a parallel query execution. All the processors within a CPC, *Centralized Processing Complex*, can be used for a single query.
- The third stage is *Scalable Query Parallelism* which spawns multiple MVS tasks over all the members in a *DB2 data sharing group*. Since a DB2 data sharing group can include many DB2 subsystems on many different CPCs, the number of processors and I/O channels for a single query are not limited by a single CPC boundary.

Stage 1 and stage 2 allow intraquery parallelism over a DB2 shared memory architecture while stage 3 extends to a DB2 shared disks architecture. Many issues in intraquery parallelism needs to be solved in the context of existing DB2 architecture so that intraquery parallelism will work with the existing DB2 data and applications. Most of the major design issues for DB2 query parallelism will be briefly discussed in the following sections, they include

- Parallel Query Execution Model
- Post Optimizer for Query Parallelism
- Resource Negotiation
- Synchronization
- Resource Control and Monitoring

A few performance measurements will be included to show the improvements and overheads of Query I/O Parallelism against sequential query execution.

2 Parallel Query Execution Model

Parallel query execution can be achieved through operation pipelining and data partitioning. DB2 decides to implement the data partitioning first due to the considerations,

- DB2 supports partitioned table since Version 1, which allows a DB2 user to distribute his large

**Proceedings of the 21st VLDB Conference
Zurich, Swizerland, 1995**

table up to 64 different disks. Parallel query execution through data partitioning will work well for this case.

- Parallel query execution through data partitioning works well with simple query which accesses large amount of data, this has been noticed as very common situations among DB2 customers.

DB2 query processing includes two phases, a query compilation phase and a query execution phase. The query compilation phase takes the user query, goes through parsing, catalog lookup, semantic check, access path selection and ends with a runtime structure which will be interpreted in the query execution phase.

In addition to extending DB2 query optimizer at query compilation phase, DB2 adds a query parallelism decision construct in its query runtime structure and let the query execution phase to make the final decision whether the query should be executed in parallel mode. The appropriate degree of parallelism bases on

- the input parameters in the query from the user, for example, a query like 'select * from t1 where c1 > :hvar' requires different degree of parallelism or even simply sequential execution depending on the value of the input parameter :hvar, such value is not known at the query compilation time and it is only known at the query execution time

- the available system resources at the moment, over committed query parallelism will simply cause resource contentions and system overheads to degrade the query response time and system throughputs

Based on the decision made at query execution phase, DB2 runtime interpreter can either run the query plan in a sequential mode or run the query plan in a parallel mode with preferred degree of parallelism. To run a query plan in parallel mode,

- query subplans will be modified to cover the appropriate data partitioning, each subplan will be given to a subtask to execute

- a data pipe will be created so that each subtask can send its result back to the consumer for subsequent processing

In case the parallel query plan is executed in a sequential mode, the original query subplan will be executed by the original task and neither the data pipe nor the subplans will be created.

2.1 Data Pipe

Data pipe is a new DB2 query processing construct which provides the function to route data from multiple producers to a single consumer or multiple consumers. The input from a producer task can be either a record stream or a workfile. The output to a producer task can be either a record stream or a workfile. The possible combinations include

- record input, record output, no order. In this case, DB2 data pipe has a flow control mechanism which will suspend the producers when consumer takes data slower than the producers. Those suspended producers will be resumed once the low water mark threshold is reached.

- record input, record output, natural order. For example, 'select * from employee where age = 65 order by empno' where the table is partitioned on empno and accessed by an index on empno. DB2 data pipe understands the relative order among all the producers of a data pipe. The results of the first one will be flow back with no buffering while the subsequent ones will be accumulated in data pipe buffers and workfiles when data pipe buffers are full.

- record input, record output, key order. For example, 'select sum(balance), avg(balance) from accounts group by branch'. In this case, the data aggregation will be done in two stages. The first stage will be done by the producers of data pipe and the second stage will be done by the consumer of the data pipe. The AVG function will be handled as a COUNT and SUM by the first stage aggregation in each producer of the data pipe.

- workfile input, record output, key order. For example, 'select * from employee where age = 65 order by deptno'. Each producer of the data pipe decides to do a local sort and DB2 data pipe will merge the results according to the needed key order.

- workfile input, workfile output, key order. Sort Merge join in parallel is such an example. DB2 data pipe can accept a set of key ranges to repartition the input data into multiple output workfiles. Each workfile will be assigned to a different subplan in the subsequent operation.

The interface of data pipe is record oriented through smart buffer allocations, which doesn't require the producer of a data pipe to assemble fields together nor the consumer of a data pipe to deassemble a received record, so that the CPU overhead will be kept to its minimum. DB2 query compiler will assign contiguous

buffers for all the fields in a data pipe record. Such buffer allocation scheme also works when a parallel query plan is executed in sequential mode with no data pipe involved. There will be no data move operations between the place where data are produced or retrieved and the place where data are consumed because both places point to the same buffer.

3 Post Optimizer for Parallelism

DB2 query optimizer has the cost function in terms of resource consumption such as the CPU time and I/O time. The parallel query plan will be determined by a post optimization phase after the sequential query optimization. This post optimizer examines the sequential query plan to see how to parallelize a sequential plan segment and estimates the overhead as well as the response time reduction if this plan segment is executed in parallel. If the overhead is below a threshold and the response time reduction is above a threshold, a parallel query plan will be constructed.

3.1 Parallel Group in a Parallel Query Plan

A DB2 sequential query plan consists of a chain of operations. Each operation can be a table access, a join, or a sort and is represented by a miniplan data structure. DB2 post optimizer examines the sequential query plan in its miniplan chain form to determine the maximal plan segment which can be executed under a single data partitioning scheme. Such a sequential query segment can include

- single table access
- linear join of multiple tables
- aggregate, group by, order by on single table or join of tables

The parallel query plan for a sequential plan segment is called a *parallel group*. which includes the constructs for the query execution phase to

- make the decision on the degree of parallelism
- negotiate for system resources
- establish the data pipe
- replicate the subplans and spawn subtasks

A sequential query segment for parallelism typically starts with a table access miniplan and continues to cascade join miniplans until there is a sort miniplan. A sort miniplan appears either for the SQL group by, order by processing or reorders the current result set for subsequent processing.

DB2 post optimizer may transform a sequential plan into a parallel query plan with multiple parallel groups.

3.2 Decision for Degree of Parallelism

When there is no input parameter referenced in a parallel group, the decision for the best degree of parallelism will be made at query compilation phase. Otherwise, the decision will be made at query execution phase. The decision making includes two steps,

- step 1, the best possible response time will be estimated on,

  - the CPU elapsed time, which will be estimated based on the CPU time estimation, number of processors available, the utilization factor of the systems

  - the I/O elapsed time, which will be estimated based on the access pattern and partition size, when a table is accessed by the query plan in physical order the prefetch I/O time will be used on partition level, otherwise random I/O time will be used on table level

  The best response time is the maximum among the above.

- step 2, data ranges for subplans will be determined by the above estimation of the best response time, once the data ranges are determined the degree of parallelism comes out naturally

An extremely simple example for the following query will be used to show the steps,

```
SELECT COUNT(*)
FROM   ACCT, TRANSACT
WHERE  ACCT.ACC# = TRANSACT.ACC#
  AND  ACCT.BALANCE > 5000
  AND  TRANSACT.AMT > 1000;
```

Table ACCT is a partitioned table on ACC# with 2 partitions. Partition 1 has key range from 1 to 12000 and I/O time 80 seconds. Partition 2 has key range from 12001 to 20000 and I/O time 30 seconds.

Table TRANSACT is also a partitioned table on ACC# with 3 partitions. Partition 1 has key range from 1 to 5000 and I/O time 50 seconds. Partition 2 has key range from 5001 to 15000 and I/O time 60 seconds. Partition 3 has key range from 15001 to 20000 and I/O time 30 seconds.

The query plan is an index access on ACCT table on ACC#, then join the TRANSACT table through the index on ACC#. The estimated CPU time for the query is 20 seconds.

In this case, Best Possible Elapsed Time = max(20, 80, 30, 50, 60, 30) = 80 seconds. Now, what will be the key range for the first subplan ? The 80 seconds I/O time on table ACCT takes key range 1 to 12000. The 80 seconds I/O time on table TRANSACT takes key

range 1 to 10000, due to 50 seconds on partition 1 plus 30 seconds for half of the partition 2. So, the key range for the first subplan is 1 to 10000. From the previous key range, 80 seconds I/O time on table ACCT takes key range 10001 to 20000. The 80 seconds I/O time on table TRANSACT takes key range 10001 to 20000. So, the key range for the second subplan is 10001 to 20000, and the degree of parallelism is 2.

4 Resource Negotiation

A synchronous I/O for a data page may take up to 20 ms while a prefetched data page only takes 2 ms. If the buffer pool is over committed and the prefetched pages are stolen before they are referenced, the same page has to be retrieved again via a high cost synchronous I/O operation. Under this case, a parallel execution of a query can cause more I/O contention and even much slower than a sequential execution. More than that, the overall system throughput can be degraded dramatically. At the query execution phase, each parallel group will negotiate with the DB2 buffer manager to understand what is the appropriate degree of parallelism. The planned best degree will be adjusted by the result of this resource negotiation, a parallel query group may even be reduced to a sequential execution when the system is overloaded.

Working with the resource negotiation, the buffer manager can be asked to increase or decrease the amount of buffers available for parallel query execution through the DB2 *alter bufferpool command.* It is possible to reduce the amount of buffers for the parallel query execution in the prime time of system load, so that parallel query execution can be totally shut down to allow maximal throughput for online transactions. The parallel query executions can be restarted at off shift by increasing the amount of buffers for parallel query execution.

In a scalable query parallelism environment, it is possible to dynamically configure the workload for parallel query executions among the DB2 members in a DB2 data sharing group by adjusting different buffer pool thresholds on each DB2 member system. Buffer pool can be configured to specify how much will be used for parallel query executions of its local DB2 subsystem. In addition, it can also be configured to specify how much will be used for parallel queries from other DB2 data sharing members. DB2 allows each member to preserve the needed amount of resources for its local processing and configure the global resources which can be shared among all the members.

5 Synchronization

5.1 Lock Manager

DB2 uses locks to serialize critical database operations, such as data set open within a DB2 member. Locks for serialization of critical section should work the same way as before. DB2 also uses locks for data concurrency and data consistency, such as the database lock, table lock, page lock, row lock, and so on.

Within a transaction, data can be inserted or updated. When DB2 spawns multiple tasks to execute a query in parallel, each subtask will acquire and release locks as normal. Each subtask should be able to see the inserted and updated data which are protected by locks from other transactions to access. A simple solution which maps all the subtasks into one requester will not work. One example is that the deadlock detection will catch a false deadlock while it isn't, subtask 1 holds lock x, subtask 2 requests lock y, another transaction holds lock y and waits on lock x.

DB2 extends the lock manager with compatible class to catch not only the relationships between lock requesters and lock holders but also the relationships among the subtasks in the same transaction. Also, the locks from a subtask should be able to transferred to its parent task when the subtask completes. This procedure guarantees the isolation level of repeatable read.

5.2 Intertask Communication

One DB2 query may include multiple parallel groups and multiple parallel groups in a single query can be executed concurrently. There may exist multiple DB2 queries active at the same time from a DB2 application program,

User may close a query, which should clean up all the subtasks for all the parallel groups in this query. User may cancel an application, which should clean up all the queries in all the programs in this application. User may commit or rollback the transaction, which should clean up all the queries in all the programs in this application. SQL processing error, such as overflow, divide by zero, should end the query itself. DB2 severe error may occur, DB2 will attempt its recovery procedures. When recovery procedures cannot recover, all the queries in all the programs in the transaction will be stopped, cleaned and the transaction be rollbacked Under the parallel query execution, each possible event has to be propagated and synchronized in between all the related tasks.

When DB2 executes a parallel query plan, a subtask can pass its result to the data consumer via a data pipe mechanism. DB2 data pipe mechanism supports flow control, so a subtask will suspend its execution

when a data consumer doesn't consume the data faster enough. Later, the data consumer will resume the suspended subtask once the remaining data drops below a certain threshold. Of course, even when a suntask is suspended, the subtask needs to respond with special events, such as the clean up or cancel request. Also, the consumer of the DB2 data pipe may be suspended to wait for data from the producers of the data pipe, it has to respond those special events, such as a subtask ends abnormally. A messaging mechanism has been adopted to maintain a posting area. When an event happens or a message arrives, the posting area will be updated and the needed follow-up actions will be triggered. This implementation provides a very flexible mechanism to serve the very complex synchronization needs in parallel query executions.

6 Miscellaneous Issues

6.1 Parallelism Enabling and Control

DB2 provides a bind option to enable the query parallelism for static SQL and a SQL special register for dynamic SQL. So, users have the full control whether they want DB2 to exploit the query parallelism which will reduce the query response time by fully utilizing the system resources for a single query. For a dynamic SQL, the query parallelism can be turned on and off in between each SQL query.

In addition to that, DB2 *Resource Limit Facility* allows one user to specify how aggressive he wants DB2 to exploit the query parallelism for his queries. Query I/O parallelism adds minimal CPU overhead comparing with the sequential execution, while scalable query parallelism adds more CPU overhead in its execution. DB2 user can limit DB2 to exploit only I/O parallelism, single CPC parallelism, or scalable parallelism by using the DB2 Resource Limit Facility.

6.2 Resource Monitoring and Control

DB2 supports lock escalation threshold to avoid lock flooding. When a query is executed in parallel, DB2 will maintain a single system image and support a global lock escalation threshold the same as that in the sequential execution mode. Users and DBAs don't have to adjust the lock resource due to parallel query execution. The same strategy applies to lock limit threshold so that parallel query execution will not monopolize the lock manager's resource. On the CPU side, DB2 supports CPU limit threshold to govern overrun queries. By the same requirement, DB2 will monitor and limit a parallel query, as a whole, not to exceed its CPU limit threshold.

6.3 Performance Monitoring

DB2 supports performance monitoring through a couple of mechanisms, each one of them will be extended to monitor parallel query execution.

DB2 *explain* facility reports how a query is optimized to execute in a PLAN table. The PLAN table has been extended to describe the parallel groups in a query. New information for a parallel group includes what operations are executed in a parallel group and what is the estimated best degree of parallelism at query compilation phase.

DB2 provides several performance records on how a parallel group is executed. The degree of parallelism at the query execution phase, the exact ranges of the data partitioning scheme will be reported when a parallel group is started. The elapsed time of the parallel group as well as the elapsed time and the CPU time for each subtask will be reported when a parallel group finishes.

DB2 reports its I/O prefetch quantity adjustments due to the parallel query workloads on each buffer pool. DB2 also reports the results of buffer pool negotiation. Information will be provided to tell the user how to tune the buffer pool size to increase the success rate in buffer pool negotiations.

7 Performance Measurements

DB2 *Query I/O Parallelism* in Version 3.1 has been released to DB2 customers in 1993. DB2 Query I/O Parallelism has been measured on a IBM 3090-300J with 512 MB central storage and 1024 MB expanded storage against a DB2 internal query workload adopted from a DB2 major customer. Three queries have been picked to show the measurements in their parallel executions as well as sequential executions.

- Query Q1 was a simple SELECT COUNT on a non-indexed column.

```
SELECT COUNT(*)
FROM   T6
WHERE  COLX = 19840918;
```

  Table T6 was a partitioned table with 8 partitions and a total of 78K pages. Partition sizes range from 8788 to 10630 pages. The access method used was a tablespcae scan, there was one parallel group and the planned degree was 8.

  DB2 accounting trace showed 85% elapsed time reduction from 143.57 sec to 20.93 sec, while the CPU time increases from 7.64 sec to 7.95 sec of extra 4%.

- Query Q2 was a join of two tables, followed by GROUP BY and ORDER BY.

```
SELECT T1.C2, T5.C54, T5.C25, COUNT(*)
FROM   T5, T1
WHERE  T5.C9 = 'FINALDSP'
  AND  T1.C1 = T5.C18
GROUP BY T1.C2, T5.C54, T5.C25
ORDER BY T1.C2, T5.C54, T5.C25;
```

Table T5 was a partitioned table with 6 partitions and a total of 78K pages. Table T1 was a non-partitioned table with 4 pages. The access method used was a nested loop join of T5 and T1. Table T5 was accessing via sequential scan. Qualifying rows joined with table T1 via a one-column matching index. There was one parallel group which includes T1 access and T2 join. The planned degree of parallelism was 5. Then the result was sorted for the GROUP BY processing in sequential mode.

DB2 accounting trace showed 73% elapsed time reduction from 142.18 sec to 39.06 sec, while the CPU time increases from 29.02 sec to 31.10 sec of extra 7%.

- Query Q3 was a join of five tables, followed by ORDER BY.

```
SELECT T1.C2, T6.C4, T4.C1, T4.C14,
       T6.C1, T3.C5
FROM   T1, T6, T4, T2, T3
WHERE  T6.C8 = 'C'   AND  T6.C6 = 'INFOR'
  AND  T6.C1 = T2.C1 AND  T2.C2 = T4.C1
  AND  T1.C2 = T6.C4 AND  T3.C1 = T2.C2
  AND  T3.C2 = T2.C3 AND  T3.C5 <> 'EN'
  AND  T3.C5 <> 'FR'
ORDER BY T1.C2, T6.C4, T4.C2;
```

Table T1 was a non-partitioned table with 4 pages. Table T2 was a non-partitioned table with 5000 pages. Table T3 was a partitioned table with 7 partitions and a total of 96K pages, partition size is between 12467 pages and 14186 pages. Table T4 was a partitioned table with 7 partitions and a total of 56K pages, partition size is between 7471 pages and 8249 pages. Table T6 was a partitioned table with 8 partitions and a total of 78K pages, partition size is between 8788 pages and 10630 pages. The access method used was a combination of nested loop joins and hybrid join. There were two parallel groups. The first parallel group included a sequential scan on T3 and nested loop join on T2, the degree of parallelism was 2. The results of the first parallel parallel group would be sorted and repartitioned into 3 workfiles. The second parallel group included a sequential scan on each workfile, hybrid join on T6, nested loop join on T1, and nested loop join on T4. The planned degree of parallelism was 3 in the second parallel group. Then, the results were sorted for the ORDER BY.

DB2 accounting trace showed 32% elapsed time reduction from 215.19 sec to 146.76 sec, while the CPU time increases from 75.93 sec to 75.84 sec of extra 0.1%.

8 Conclusion

DB2 Query Parallelism implementation is based on the DB2 shared disk architecture. However, it takes full advantages of the shared memory when the parallel subtasks are executed within the same Centralized Processing Complex.

The major goal for DB2 Query Parallelism is to improve the query response time by full utilization of the available system resources. The implementation allows a parallel query plan to decide its degree of parallelism and distribution of work dynamically at the query execution phase. DB2 system resources will be monitored, negociated and controlled for the best utilization to improve the query response time as well as the system throughput.

Not all the DB2 operations have been supported by parallelism in its first round implementation. SQL insert, delete, update and outer join are not parallelized yet. Feedbacks on performance and functional requirements will be gathered for continuous enhancements on DB2 intraquery parallelism.

9 References

- C. Mohan, H. Pirahesh, W.G. Tang, Y. Wang, "Parallelism in relational database management systems", IBM System Journal Vol 33, No. 2 (1994)
- A. Tsang, A. Shibamiya, "DB2 Version 3 Query I/O Parallelism Performance Study", IBM Technical Report TR03.545, STL, San Jose, March 1994
- IBM DATABASE 2 Version 3 Application Programming and SQL Guide, SC26-4889-00
- IBM DATABASE 2 Version 3 Command and Utility Reference, SC26-4891-00
- IBM DATABASE 2 Version 3 Administration Guide, Volume 3, SC26-4888-00