Parallelism in IBM® WebSphere® Information Integrator
Version 8.2

By Simon Harris
DB2 and WebSphere Information Integrator Performance
IBM Silicon Valley Lab
February, 2005
Table of Contents

1  A brief introduction to what’s new in WebSphere II v8.2........................................3
2  Overview................................................................................................................3
   2.1 Wrapper architecture......................................................................................3
   2.2 Parallel execution of federated queries..........................................................4
3  Trusted and fenced wrapper architecture...............................................................4
   3.1 Trusted wrappers............................................................................................5
   3.2 Thread-safe fenced wrappers .........................................................................5
   3.3 Non thread-safe fenced wrappers...................................................................7
   3.4 Wrapper memory considerations....................................................................7
   3.5 Wrapper runtime considerations.....................................................................9
4  Intra-partition parallelism ....................................................................................10
   4.1 Runtime considerations for intra-partition parallelism...................................11
   4.2 Implementation guidelines for intra-partition parallelism................................12
5  Inter-partition parallelism ....................................................................................12
   5.1 Runtime considerations..................................................................................15
   5.2 Implementation guidelines for inter-partition parallelism...............................16
6  Computational partition groups ..........................................................................16
   6.1 Runtime considerations for computational partition groups...........................17
   6.2 Implementation guidelines for computational partition groups.......................17
7  Summary..............................................................................................................18
8  Acknowledgments: ..............................................................................................18
9  Special Notice ....................................................................................................18
1 A brief introduction to what’s new in WebSphere II V8.2

This article discusses the performance-related features introduced as part of IBM WebSphere Information Integrator (WebSphere II) V8.2. The focus is on those aspects that allow WebSphere II to make better use of the resources available in partitioned database and SMP environments. In brief, the article covers:

- **Fenced and trusted wrapper architecture**
  This section explores the WebSphere II process model and explains the difference between trusted and fenced wrappers – the latter being a prerequisite for exploitation of partitioned database resources. In addition, it also discusses the scalability benefits the fenced wrapper can bring to federated servers supporting a high number of concurrent users.

- **Intra-partition parallelism for SMP systems**
  This section discusses when it is appropriate to enable intra-partition parallelism in a federated environment and the benefits you can achieve with this capability.

- **Inter-partition parallelism for partitioned database servers**
  Finally, the article details with how the new features improve resource utilization and query response times for WebSphere II instances that are partitioned.

Other performance-related features introduced in WebSphere II V8.2 but not specifically covered by this article include cache tables, the “Update Statistics” facility, informational constraints on nicknames, and improvements to the snapshot monitor to enable monitoring of remote SQL fragments. Details on each of these can be found in the *IBM DB2® Information Integrator Federated Systems Guide* – SC18-7364-01 (ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/iivfpe81.pdf).

For the purposes of the technical discussions within this paper, it is important to understand that WebSphere II (formerly known as IBM DB2® Information Integrator) continues to be built upon core DB2 Universal Database™ (DB2 UDB) technology. The federated server, which is the core component of IBM’s data federation, continues to be a DB2 database enabled for federation. The re-branding of the DB2 II to WebSphere II has in no way altered the internal workings of the Information Integrator product.

2 Overview

2.1 Wrapper architecture

WebSphere II uses wrappers to communicate with remote data sources. These wrappers are essentially responsible for maintaining remote data source connections, issuing SQL in the native dialect of the remote data source, and retrieval of subsequent result sets. In WebSphere II V8.1, the wrapper code is loaded and executed directly within the DB2 agent process. This ‘trusted’ form of execution is very efficient, but also has a number of issues:

- Bugs within the wrapper code and/or foreign client code may affect the stability of the DB2 engine.
- Severe restrictions are placed on the type of plans the optimizer may generate in a partitioned database environment.
- Every db2agent process working on behalf of an application must load its own private copy of the wrapper library. With a high number of concurrent users, this may consume significant amounts of memory.

WebSphere II V8.2 introduces the ability to execute wrappers in **fenced mode**, where the wrapper is loaded and executed in a process separate from the DB2 engine. Fenced wrappers provide several benefits over trusted wrappers, including:

- The DB2 engine becomes more fault-tolerant because bugs within the wrapper or foreign client can no longer affect the engine.
- Restrictions on the type of plans the optimizer may generate in a partitioned environment are relaxed, resulting in more efficient federated query execution.
2.2 Parallel execution of federated queries

WebSphere II’s exploitation of parallelism comes in two forms that are closely related to DB2 UDB’s intra-partition and inter-partition parallelism.

**Intra-partition parallelism** (also known as SMP parallelism) refers to the process of decomposing a query into multiple sections that are run in parallel by multiple processes on a single database partition. In WebSphere II V8.1, there is no intra-partition parallelism, and all federated queries are executed by a single process. WebSphere II V8.2 makes better use of multi-processor machines by enabling the sections of a federated query that reference local data to execute in parallel on multiple processes. Sections of the federated query that references nicknames still execute serially.

Inter-partition parallelism (also known as MPP parallelism) is only available on partitioned database servers and refers to the process of dividing a single query into multiple parts that run in parallel on the different partitions of the server. In WebSphere II V8.1, federated queries involving local partitioned data and remote nickname data can only execute on the coordinator partition (the partition to which the application connects). That is, both the remote nickname data and local partitioned data are processed serially on the coordinator partition, essentially losing any benefit from the partitioning of the local data. In WebSphere II V8.2, access to nickname data is still serialized at the coordinator partition, but once the data has been fetched, the federated server may dynamically re-distribute the nickname data to each of the local partitions - so the processing between nickname and local partitioned data can occur on all partitions in parallel.

WebSphere II V8.2 also introduces the concept of computational partition groups (CPGs) on partitioned database servers. A computational partition group is a set of database partitions (commonly referred to as a partition group) that can be used to dynamically re-distribute nickname data in order to achieve parallel join processing. CPGs are used when the federated query references only nickname data. The nickname data is serially fetched from the remote source to the coordinator partition and then dynamically re-distributed to the partitions in the CPG for parallel processing.

The net effect of these changes means federated queries can have significantly improved performance over previous releases of WebSphere Information Integrator because multiple processors and/or partitions can work in parallel in order to resolve queries.

3 Trusted and fenced wrapper architecture

Within WebSphere II, connections to federated sources are implemented through wrappers. A wrapper is simply the mechanism by which the federated server interacts with a data source. Routines within a wrapper allow the federated server to manage connections to data sources, issue queries, retrieve data, and so on, and also contain information about the properties and capabilities of that class of data source (for query planning purposes). The wrapper interacts with the data source by loading the client libraries of the foreign data source, and communicates using the vendors' published API. The foreign client libraries for non-DB2 data sources are supplied by the vendor of the remote data source.

Recall that a wrapper may be defined as trusted or fenced, and that a trusted wrapper is executed within the DB2 engine, and a fenced wrapper is loaded and executed by a process external to the DB2 engine (known as a DB2 Fenced Mode Process, or db2fmp). WebSphere II V8.1 only supports trusted wrappers, while a wrapper in WebSphere II V8.2 can be defined as fenced by setting the DB2_FENCED wrapper option to ‘Y’ (default setting for this option is ‘N’ for trusted).
The foreign client libraries loaded by a db2fmp may be thread-safe or non-thread-safe. Thread-safe client libraries allow the db2fmp process to create and manage multiple threads, where each thread corresponds to a remote data source connection. Non-thread-safe clients do not allow the db2fmp process to safely create multiple threads, and consequently, each db2fmp can only support a single remote data source connection. This has scalability implications that are discussed later in this article.

There now follows a more detailed discussion on the architecture, benefits, and drawbacks of fenced and trusted wrappers.

### 3.1 Trusted wrappers

When the wrapper is defined as trusted, the wrapper code is loaded directly into the db2agent process – that is, it is allowed to execute in the same address space as the DB2 UDB engine, as illustrated in Figure 1.

![Figure 1: Trusted wrapper architecture.](image)

Whilst this generally provides the best-performing solution for a serial configuration, problems in the wrapper or foreign client code may impact the stability of the federated server. A program exception in the wrapper or foreign client could result in the db2agent shutting down abnormally, which may force the DB2 instance to end abruptly.

In trusted mode, since the wrapper library is loaded into each db2agent, then the wrapper code cannot be shared by other applications. Therefore, each db2agent must load its own copies of the wrapper libraries it requires to access the remote data sources. This can be inefficient in terms of memory usage.

Plans involving trusted nicknames are also limited in the scope of parallelism they can achieve. See section ‘4, Intra-partition parallelism,’ and section 5, ‘Inter-partition parallelism,’ for more details.

### 3.2 Thread-safe fenced wrappers

WebSphere II V8.2 introduces the ability to execute wrappers in fenced mode. Fenced wrappers are loaded by a process external to, and in a separate address space from, the DB2 engine and are implemented using a similar technique to DB2 fenced routines (stored procedures and UDFs). The fenced wrapper architecture is illustrated in Figure 2.

![Figure 2: Fenced wrapper architecture.](image)

The fact that the wrapper is loaded by an external db2fmp process protects the engine from program defects that may exist in the wrapper or foreign client. (This isolates the instance from any potential implementation problems that may exist in these components.) If a trap occurs in the wrapper or foreign client code, the process or thread (for thread-safe wrappers) running the wrapper will terminate without impacting the instance. The isolation of the wrapper component in this manner is particularly important when using wrappers developed by third parties, or when developing your own wrapper using the wrapper SDK.
In addition, in order to isolate the wrappers from potentially corrupting each other, each db2fmp will only load a single wrapper library. That is, a single db2fmp will support connections to those federated data sources that use a particular library. For example, a db2fmp that has loaded the DRDA wrapper will not load the net8 wrapper.

Whilst this isolation provides greater stability, it comes at a cost. Since the engine and db2fmp are separate processes, they must communicate using IPC resources, which have additional overhead compared to trusted communication (which takes place directly). The impact of this overhead on execution time is discussed in more detail in section 3.5, ‘Wrapper runtime considerations.’

Unlike the trusted wrapper where the db2agent owns the connection to the data source, in a fenced wrapper it is the db2fmp process that owns the connection. Since the connection now lives outside the db2agent, the db2fmp is able to act as a gateway to a specific class of data source and the wrapper code can be shared by multiple applications (with a thread-safe client). When a db2agent requires access to a data source using a fenced wrapper, it will check to see if a db2fmp process has already loaded the required wrapper. If the wrapper has already been loaded, the agent will reuse that wrapper. If the agent finds the wrapper has not previously been loaded, then it will start a new db2fmp process, which will then load the appropriate fenced wrapper.

Figure 3 shows a wrapper library using a thread-safe client loaded into a db2fmp, maintaining federated data source connections for three applications.

As an example, consider application A, which references nicknames defined using fenced wrappers for net8 and DRDA data sources, and application B, which references fenced nicknames for net8 and Informix data sources. This will cause three db2fmp processes to be created, as illustrated in Figure 4.

In order to prevent WebSphere II from maintaining idle connections to the remote data source and consuming resources unnecessarily, a wrapper will release a data source connection under the following conditions:
- When a db2agent terminates and releases its local connection to the federated server
- After DDL operations which alter the definition of federated objects (wrapper/server/user mapping)
- When the data source has not been referenced for a number of transactions
Use of the fenced wrapper also allows the optimizer to generate parallel access plans on a partitioned database server. Unlike a DB2 coordinating agent, the db2fmp process is able to access nicknames and then distribute or broadcast the data to subagents for parallel processing, leading to potentially large performance improvements. See section 5, ‘Inter-Partition Parallelism,’ for a more detailed explanation.

### 3.3 Non-thread-safe fenced wrappers

A db2fmp loading a non-thread-safe foreign client behaves very similarly to a db2fmp loading a thread-safe foreign client, except it can only support a single connection to a remote data source. This means that unlike the thread-safe wrapper, a non-thread-safe wrapper cannot be shared by multiple applications. In this case, each db2agent accessing nicknames of a particular data source type will start a new db2fmp process, which will then load the appropriate wrapper library. Figure 5 illustrates how three separate db2fmp processes must be started. Note that each must load its own copy of the wrapper library (using a non-thread-safe client) in order to support three applications all accessing the same data source.

The impact this has in terms of memory utilization is key, and is discussed in more detail in section 3.4, ‘Wrapper memory considerations.’

![Figure 5: Non-thread-safe fenced wrappers](image)

**Note:** Currently, only Sybase ctlib and Teradata wrapper clients are non-thread safe.

### 3.4 Wrapper memory considerations

One of the benefits of using fenced wrappers over trusted wrappers is the potential to reduce memory consumption. Thread-safe fenced wrappers for a given data source type are loaded only once by a db2fmp and then shared among multiple agents, whereas trusted wrappers are loaded directly into each agent and therefore incur redundancy.

The chart in Figure 6 compares the memory consumption of db2agents plus their fenced or trusted wrapper against a varying number of concurrent federated server connections. The figures above the bars indicate the percentage of memory saved using the fenced wrapper over the trusted wrapper. You can see that for a single connection, a trusted wrapper consumes less memory than a fenced wrapper. However, the fenced wrapper scales much better than the trusted wrapper and typically starts to have a smaller memory footprint at about five concurrent connections. Therefore, if reducing memory consumption is an overriding factor:

- Consider using trusted wrappers if the federated server is memory-constrained, and there will be less than five concurrent connections to each class of back-end data source.
Consider using fenced wrappers if the federated server is memory-constrained and there will be five or more concurrent connections to each class of back-end data source.

These tests used a wrapper library with a thread-safe client. You would expect similar tests on a wrapper library using a non-thread-safe client to show much less scalability because a db2fmp would have to be created to service each connection. Consequently, in the case of a non-thread-safe client, a trusted wrapper likely provides better memory utilization and scalability than a fenced wrapper.

Use of the fenced wrapper may also impact the amount of shared memory available to a DB2 instance created under WebSphere II. The IPC communications used to exchange data between the database engine and the fenced wrapper process require a shared memory segment. Consequently, there is one less shared memory segment available for other users of the database shared memory (such as bufferpool and sort space). This has most significance on 32-bit systems where there is a limit to the amount of shared memory that DB2 can reference. If the DB2 instance (created under WebSphere II to act as the federated server) is already using fenced mode processes for UDFs or fenced stored procedures, then fenced wrappers will use the same shared memory segment (and consequently their use will not impact shared memory).

The following DB2/WebSphere II parameters relate to fenced mode processes and fenced wrapper configuration and may be useful in tuning fenced wrappers:

- **ASLHEAPSZ** database manager configuration parameter defines the size of the communication buffer used between:
  - local db2agents and the associated applications, and
  - local db2agents and db2fmp processes
  The defined number of bytes is allocated from shared memory for each db2fmp process or thread that is active on the system.

- **DB2_FMP_COMM_HEAPSZ** registry variable defines the maximum number of 4K pages from the shared memory segment, which can be used for all fenced routine communication (not just fenced wrappers). Send/Receive communication buffers for all active db2fmp’s of size ASLHEAPSZ are allocated from this memory pool. The default size for fmp communication is approximately 22MB, which is enough space to run approximately 10 fenced routines. You may need to increase this value (by setting DB2_FMP_COMM_HEAPSZ) if you are using a large number of fenced routines/wrappers.

- **KEEPFENCED** database manager configuration parameter indicates whether or not a fenced mode process (db2fmp) is kept in the fenced pool after a fenced mode routine call is complete.

---

**Note:** In WebSphere II V8.2, the health monitor uses fenced stored procedures and is enabled by default. Therefore the use of fenced wrappers will not consume an additional shared memory segment.
**FENCED_POOL** database manager configuration parameter defines the number of fenced mode processes (db2fmp’s) which can be kept in the cache. It has a slightly different use, depending on whether or not the client library loaded by the db2fmp is threadsafe or not:
- For threadsafe db2fmp processes (processes serving threadsafe stored procedures and UDFs), this parameter represents the number of threads cached in each db2fmp process.
- For non-threadsafe db2fmp processes, this parameter represents the number of processes cached.

**Note:** For a federated server, this parameter should be sized to take into account the fact that db2fmps loading fenced wrappers are always pooled and never destroyed. You must also account for other stored procedures and UDFs that run on the federated server.


## 3.5 Wrapper runtime considerations

As you might expect, fenced processes communication between the db2agent and db2fmp is inherently more costly than the direct communication employed by trusted wrappers.

The chart within Figure 7 compares trusted and fenced wrapper execution times when selecting rows having eight columns from a single remote Informix nickname.

The chart indicates the results of the tests and shows that the fenced wrapper adds a 3% elapsed time overhead for this particular query. A 3% increase in execution time may be considered relatively minimal compared to the many benefits associated with using fenced wrappers.

The fenced overhead varies depending upon the query being executed and the plan chosen by the optimizer. In general, plans that involve many small interactions between the db2agent and wrapper (for example, when a nickname is selected as the inner of a nested loop join) have more overhead than those plans that have fewer interactions between the agent and wrapper (such as the bulk fetch shown in Figure 7).
However, the overhead that the two-process fenced wrapper structure introduces can often be outweighed by the fact that fenced wrappers provide the optimizer with a wider choice of plans in a partitioned environment. In such an environment, working on relatively large data volumes, the change in plan will often more than compensate for the fenced process overhead. See section 5, 'Inter-partition parallelism,' for more details.

4 Intra-partition parallelism

Intra-partition parallelism (often referred to as SMP parallelism) occurs when a query is decomposed into multiple parts, which are then executed in parallel by multiple processes called subagents. A coordinator agent 'coordinates' the subagents by giving them tasks to perform and is responsible for collecting the results from each subagent before delivering them to the next operator in the query. Intra-partition parallelism can occur on a non-partitioned database server or within a single partition of a partitioned database server and is enabled by setting the database manager configuration parameter, INTRA_PARALLEL to 'Yes.' This often results in faster execution time of queries, but with increased CPU utilization.

Intra-partition parallelism for federated queries is disabled in WebSphere II V8.1. That is, execution of any query containing reference to a nickname is serialized at the coordinator agent. WebSphere II V8.2 has been enhanced to enable intra-partition parallelism for those parts of a federated query that reference local DB2 data. Sections of the query that reference nicknames are still executed serially by the coordinator agent. Consequently, this new feature will only benefit those federated queries that also reference local data.

**Note:** Intra-partition parallelism is not dependent upon the use of the fenced wrapper; that is, it can take place with both fenced and trusted wrappers.

Within an SMP-enabled federated plan, remote access (the SHIP operator for relational sources and RPD for non-relational sources) becomes a serialization point. This forces all operations in the query plan above the SHIP operator to be executed serially by the coordinating agent. Essentially, the further down to the bottom left of the plan a SHIP operator appears, then the greater the serialization within the plan. To demonstrate this, consider the following SQL statement:

```
SELECT * FROM nn, t WHERE nn.col = t.col
```

where nn represents a nickname, and t represents a local DB2 table.

In WebSphere II V8.2, with intra-partition parallelism enabled, if the local table t is the outer of the join operation, a possible execution plan chosen by the optimizer is shown in Plan 1.

In this plan, the TQ (TableQueue) operator defines the boundary between coordinator and subagents. The subagents will scan the local table in parallel and send the rows to the coordinator agent. The coordinator agent will then send the receiving rows to the nested loop join, which will use the SHIP operator to access the remote data. In this plan, all operators above the TQ occur at the coordinator agent and are therefore serialized.

However, if the optimizer chooses the nickname to be the outer table of the join (as illustrated in Plan 2), then the query will be fully serialized (no TQ operators appear within the plan) because the SHIP operator is at the bottom-left side of the plan.
Therefore, the degree on intra-partition parallelism achieved within a query (and therefore its execution time) is highly dependant upon the position of the SHIP operator within the plan.

Similarly, other federated operations, such as UNION, or INSERT/UPDATE/DELETE, can take advantage of this enhancement by parallelizing the local portions of the plan.

The techniques used to tune and monitor intra-partition parallelism in a federated environment are the same as for regular DB2 UDB databases. The degree of parallelism may be controlled by tuning the following parameters:

- INTRA_PARALLEL database manager configuration parameter
- DFT_DEGREE database parameter, CURRENT DEGREE special register or DEGREE bind option
- MAX_QUERYDEGREE database manager parameter or the RUNTIME DEGREE special register

### 4.1 Runtime considerations for intra-partition parallelism

Tests have shown that on a federated server that has spare CPU cycles, enabling intra-partition parallelism provides a performance improvement for queries that reference moderate to large amounts of local DB2 data. For most other queries, enabling intra-partition parallelism has minimal effect of runtime.

The chart in Figure 8 summarises a test in which eight DSS type queries (all of which benefited from intra-partition parallelism) were executed in three different environments.

The environments tested were:

- No SMP: A federated server with intra-partition parallelism disabled, using trusted wrappers. The federated server was tuned differently from the SMP enabled tests to ensure it made best use of the resources available.
- SMP Fenced: A federated server with intra-partition parallelism enabled, using fenced wrappers.
- SMP Trusted: A federated server with intra-partition parallelism enabled, using trusted wrappers.

Each of the queries referenced a combination of local and remote nickname data.

The 32% increase in CPU cycles necessary to support the 24% reduction in execution time measured during this test demonstrates the need to have spare CPU cycles on the federated server.
server before enabling intra-partition parallelism; otherwise, the CPUs may become overloaded and system throughput will probably decrease.

Our tests also showed the fenced and trusted wrappers to have virtually identical execution times for this query set, illustrating that fenced wrappers can have comparable performance with their trusted counterparts.

4.2 Implementation guidelines for intra-partition parallelism

You can use the following recommendations as guidelines to assess when intra-partition parallelism is likely to be beneficial on a federated server:
- When there are sufficient volumes of local data (either local user tables, or MQTs)
- For decision support-type workloads (not OLTP)
- When there are sufficient spare CPU cycles available on the machine

With intra-partition parallelism enabled, the sort heap threshold is allocated from shared memory, and consequently there is one less shared memory segment available for the DB2 database shared memory set. On 32-bit systems, the number of shared memory segments DB2 can utilize is limited, and therefore the principal consumers of this memory (bufferpools and sort space) may need to be tuned in order to avoid over allocating memory.

Experience has shown that setting the degree of parallelism to four or below will (in general) provide the best overall results in terms of machine utilization and execution time.

5 Inter-partition parallelism

Inter-partition parallelism can only occur on a partitioned database system. It refers to the process of dividing a single query into multiple parts that run in parallel against different sets of the data on different partitions of a partitioned database server.

For transactional consistency, the federated server must maintain a persistent connection to a data source whilst the application referencing that data source remains active. Since DB2 sub-agents are transitional processes that may be created and destroyed at any time, it is not possible to allow these sub-agents to own and manage remote data source connections. Since no other mechanism existed in WebSphere II V8.1 (which supports only trusted wrappers), the remote data source connection was owned by the coordinating agent (since this is the only process that is guaranteed to exist for the duration of the transaction). This condition imposed restrictions on the type of federated plans that can be generated in a partitioned environment. By enforcing nickname operations to be done on the coordinator partition by the coordinator agent, one risks losing many of the benefits of a partitioned database server because operations which are usually parallelized become serialized.

As an example, consider Figure 9 which shows a table $T_1$ whose data is spread across four database partitions ($p0$ – $p3$). A federated query is issued that joins $T_1$ with remote nickname data $N_1$:

In WebSphere II V8.1, because nickname operations can only be performed by the coordinator agent on the coordinator partition ($p0$), the data of partitioned table $T_1$ is fetched from each local partitioned data executing on coordinator partition

![Figure 9: Join between local partitioned data and nickname data executing on coordinator partition](image-url)
individual partition and sent to the coordinator partition. The join operation between table and nickname then takes place serially on the coordinator partition. In this case, the two variations of plans that the optimizer may generate are shown in Plan 3.

In each case, the TableQueue operator must be above the local table $T_1$ and is responsible for sending the data from the data partitions to the coordinator partition where join processing with the nickname will take place. In this case only the fetching of the local partitioned data from the disks is performed in parallel – the actual join and anything above the join in the plan takes place serially on the coordinator partition. These types of plans are referred to as coordinator plans since most of the processing takes place on the coordinator partition.

In WebSphere II V8.2, the restriction that access to the nickname data must be managed by the coordinator agent has been lifted by introducing the concept that a db2fmp process (which is persistent) may own and manage the remote connection. This opens up a number of options to the optimizer, which can now generate plans that distribute the fetched nickname data from the coordinator partition to each of the local partitions where join (and other processing) can take place simultaneously on all partitions. Access to nickname data is still performed serially on the coordinator partition, but it may now be driven by a sub-agent and managed by the fmp.

In the example above, it is now possible to have the following sequence of events, as illustrated in Figure 10:

1. 1.1. Local data is fetched from disk by each individual partition in parallel.
   1.2. Data is fetched from the nickname onto the coordinator partition.

2. The nickname data can then be dynamically re-distributed from the coordinator partition to all other partitions which contain data for table $T_1$. Those partitions then work in parallel to perform the join along with other operations which can take place concurrently.

3. The intermediate result set from each of the partitions is then sent to the coordinator partition for final processing and the final result set is then returned to the application.
The optimizer now has many additional plan variations to consider, two of which are outlined in Plan 4.

With these plans, nickname access still occurs on the coordinator partition, but once the data is available, it can be dynamically redistributed to the partitions containing the local data from T1.

The TableQueue operator TQ(2) is responsible for sending the nickname data to the appropriate partitions so that it may be joined with the local data (operation 2 in Figure 10 above). Since this data comes from a nickname, it has no concept of a partitioning key and the data must be dynamically redistributed at runtime. The nickname data is re-partitioned on the join key in order to ensure it is co-located with the local partitioned data. The TableQueue operator TQ(1) in the above plans returns the intermediate result sets from the partitions to the coordinator (operation 3 in the diagram above).

The plans above assume that the join between the nickname and table is defined on the partitioning key of the local table. When the join is not defined on the partitioning key, then the optimizer may consider the set of plans illustrated in Plan 5:

Plan 5: Possible plans for join between local partitioned data and nickname when join is not on partitioning key

The effect of all these plans is that both nickname and table data will be co-located, ensuring that processing of the join can occur simultaneously on all partitions. In the two left-most plans, the optimizer chooses to use TableQueues to distribute the data from both the nickname and local table across all partitions using the join key as the partitioning key. In the two right-most plans, the optimizer chooses to use a Broadcast TableQueue (BTQ) to send all the rows returned from the nickname to each of the data partitions.
These types of plans are referred to as *TQ over nickname* plans because the TableQueue operator appears above the nickname (SHIP operator) in the plan – the TQs do not have to appear directly above the SHIP operator as there maybe several other intervening operators. These plans can only be generated for nicknames defined using fenced wrappers in WebSphere II V8.2.

The optimizer will usually choose between a *coordinator* style plan or a *TQ over nickname* style plan based on the following factors:

- The estimated number of rows involved in the join and the cost of transmitting these between partitions
- The increased resources available to resolve the query when a parallel plan is selected, such as bufferpool and sort space.
- The estimated number of rows output from the join (reducing or exploding)
- The cost of other operators involved in evaluating the query
- Whether or not the local table is partitioned on the join key

When deciding which plan to choose, the optimizer must offset the cost of dynamically redistributing the data with the potential benefit of parallel processing. If the optimizer thinks that the cost of data re-distribution will be greater than the benefits of parallel processing, then it is likely that a *coordinator* style plan will be selected.

### 5.1 Runtime considerations

In a partitioned environment, the fenced wrapper has consistently demonstrated a significant performance benefit over the trusted wrapper. The benefits in runtime improve as either or both of the following increase:

- Amount of local partitioned data involved in the query
- Number of local partitioned tables involved in the query

The chart in Figure 11 compares performance of the fenced and trusted wrappers in a partitioned environment using the TPCH data and workload.

The two main fact tables in the TPCH schema were locally partitioned on the database server. The remaining six dimension type tables were defined as nicknames at a single remote data source.

The tests highlight the fact that given a moderately complex decision support type environment, then across the complete workload, queries using the fenced wrapper outperformed those using the trusted wrapper. In fact in this case, the total fenced wrapper execution time was over ten times faster than the total trusted wrapper execution time.

In certain circumstances, it is possible that individual queries may execute faster using the trusted wrapper rather than the fenced wrapper. This usually occurs when the amount of local data involved in the query is small and the runtime difference is usually of the order of less
than 15% (see query 19 in Figure 11 above). At these low volumes of data, the optimizer chooses the same coordinator plan for both trusted and fenced wrappers. Since both plans are the same, the fenced wrapper is slightly slower due to the additional overhead it must incur for agent to wrapper communication (as discussed earlier in this paper). As the volume of local partitioned data involved in the join increases, the optimizer chooses to select TQ over nickname plans for the fenced wrapper which dynamically redistribute the data and perform parallel join processing. With the trusted wrapper, the operators in the plan may change as the data volumes increase, but the work must still be performed serially on the coordinator partition. Consequently, the runtime benefits of fenced over trusted increased as the volume of local partitioned data increased.

Fenced wrapper performance showed even more marked improvement over trusted when the query involved joining multiple local partitioned tables with remote nickname data. When a trusted nickname is involved in a query with multiple local partitioned tables (which may be co-located), even the join between the local tables had to occur serially on the coordinator partition. If the join between the local tables is reducing, this means transmitting many more rows between partitions than is actually necessary to satisfy the join criteria. When a fenced nickname is involved, the join of the local tables may occur in-place and in parallel on each of the data partitions. Thus, the number of rows being transferred between partitions is greatly reduced.

### 5.2 Implementation guidelines for inter-partition parallelism

The tests documented in section 5.1 show that in a partitioned database environment with federated enabled, then:

- As the volume of local and remote data involved in the join increases, and/or
- The number of local partitioned tables involved in the join increases,

the fenced wrapper demonstrated improved performance and resource utilization over the trusted wrapper. It is therefore recommended to use the fenced wrapper with partitioned database systems.

Access to nickname data using either trusted or fenced wrappers on a partitioned database system will increase the workload on the coordinator partition. Since the trusted wrapper must perform all processing on the coordinator partition, it is likely to add considerably more workload to this partition than the fenced wrapper. The design and configuration of partitioned systems which access federated data must take this into account in order to avoid potential resource bottlenecks. The most common technique for load balancing the coordinator is to rotate the partition which acts as the coordinator. You can implement this technique by allowing applications to connect to different database partitions, and rotating the partition using a round-robin technique.

### 6 Computational partition groups

Computational partition groups (CPGs) apply only to partitioned database servers and allow queries which reference only fenced nicknames to run in parallel across multiple partitions. CPGs allow the optimizer to consider plans that access the nickname data from the coordinator partition and then dynamically redistribute the data to the partitions in the CPG for parallel processing. A CPG can be any regular partition group specified in the system catalog (except IBMCATGROUP) and can be set using the DB2_COMPPARTITIONGROUP registry variable.

The example in Figure 12 illustrates the use of computational partition groups in the join of two remote nicknames, N1 and N2.

The sequence of operations is:

1. Nickname data is fetched from N1 and N2 to the coordinator partition (p0).
2. The fetched data for both N1 and N2 is dynamically re-distributed from the coordinator partition to all partitions within the CPG. Those partitions then work in parallel to perform the join along with other operations which can take place concurrently.

3. The intermediate result set from each of the partitions is then returned to the coordinator partition for final processing before the final result set is delivered to the application.

In this case the optimizer selected a plan (illustrated in Plan 6) that uses a computational partition group rather than a join on the coordinator partition. In this example, with a CPG, the optimizer only has one possible plan type that involves redistribution (although the join legs may be reversed).

In a federated query that does not reference any local data, a plan demonstrating a computational partition group will have a TQ operator above a SHIP operator. More common would be to see a TQ operator above each SHIP in a join leg. The nickname data is dynamically redistributed based on the join key between the two nicknames in the query.

6.1 **Runtime considerations for computational partition groups**

Computational partition groups are most beneficial when the amount of data retrieved from the nicknames is large and the query is moderately complex.

Once again, in certain circumstances, it is possible for a query using the trusted wrapper to outperform the equivalent query using a fenced wrapper. When the nickname data referenced by the query remains relatively small, both the fenced and trusted wrappers choose coordinator plans and computational partition groups are not selected by the optimizer. In these cases, fenced is once again slower than trusted by an average of approximately 10% to 15%. As the data being retrieved from the nicknames increased, plans utilizing computational partition groups were selected by the optimizer and the fenced wrapper quickly started to outperform the trusted wrapper.

6.2 **Implementation guidelines for computational partition groups**

In a partitioned database environment with queries that only reference federated data, then computational partition groups should be enabled as they will reduce execution time of large
federated queries. The use of CPGs is also likely to help reduce potential bottlenecks on the
coordinator partition since a CPG plan redistributes the workload amongst other database
partitions.

For regular DSS type queries, lab tests have indicated that there is no inherent performance
advantage in defining computational partition groups with additional (or fewer) partitions which
share the same resources. This is because the bottleneck for these types of queries lies with
the fmp as it fetches the data from the remote source, and this process is always serialized. In
theory, if the computation within the query is particularly complex and intensive, then more
partitions within a CPG may be beneficial; but tests thus far have not supported this
hypothesis.

7 Summary
The new features in WebSphere II V8.2 discussed in this paper represent a major step
forward in both the stability of WebSphere II federated servers and their exploitation of
resources in partitioned and SMP environments. With the introduction of the fenced wrapper
architecture, wrapper libraries can now be isolated from the DB2 engine helping to protect the
engine against errors in either the wrapper or foreign client code. By only loading the wrapper
code once for all applications, the fenced wrapper also greatly improves the scalability of
federated servers with a high number of concurrent users.

The article has also demonstrated that non-partitioned federated database servers can
benefit significantly by enabling intra-partition (SMP) parallelism for those queries that
reference a combination of local and remote data.

Most significantly, the enhancements in WebSphere II V8.2 now enable WebSphere II to
better exploit the resources available in partitioned database servers by allowing dynamic re-
partitioning of nickname data to database partitions for parallel processing. The subsequent
increase in parallel processing can have a significant impact in reducing federated query
execution times.

Enjoy!

8 Acknowledgments:
I would like to thank all the people who contributed to the content and reviewed this article. In
particular, thanks go to Eileen Lin, Susanne Englert, Holger Kache, Tian Zhang, Ron Yorita
and Anjali Betawadkar-Norwood.

9 Special Notice
The information contained in this document has not been submitted to any formal
IBM test and is distributed AS IS. The use of this information or the implementation
of any of these techniques is a customer responsibility and depends on the customer’s
ability to evaluate and integrate them into the customer’s operational environment.
While each item may have been reviewed by IBM for accuracy in a specific situation,
there is no guarantee that the same or similar results will be obtained elsewhere.
Anyone attempting to adapt these techniques to their own environment does so at
their own risk.