IBM Db2 Mirror for i: Performance Considerations

May 2020

IBM Corporation

Acknowledgements

We would like to thank the many people who made invaluable contributions to this document. Contributions included authoring, insights, ideas, reviews, critiques and reference documents.

Our special thanks to key contributors from IBM i Development:

Susan Bestgen – IBM i Performance
Scott Forstie – IBM i Development
Shawn Mohr – IBM i Development
Tim Rowe – IBM i Development
Dan Toft – IBM i Performance
Kris Whitney – IBM i Development

Our special thanks to key contributors from IBM Power Systems Performance:

Paul Nelsestuen – IBM i Performance
Rick Peterson – IBM i Performance

Our special thanks to key contributors from IBM Lab Services Power Systems Delivery Practice:

Eric Barsness – IBM i Performance

Table of Contents

1 Introduction....6
1.1 Purpose....6
1.2 Document Responsibilities....6
2 What is Db2 Mirror for i?....7
2.1 Background....7
2.2 Db2 Mirror Performance Overview....7
2.3 Architecture....8
2.4 RDMA over Converged Ethernet (RoCE)....8
2.5 Mirrored Database Performance Overhead....9
3 Db2 Mirror for i Hardware and I/O Considerations....10
3.1 Introduction....10
3.2 Partition Configuration....10
3.3 I/O Subsystem....10
4 IBM i Configuration Considerations....11
4.1 Introduction....11
4.2 Prestart Job Configuration....11
4.3 Journal Configuration....11
5 Db2 Mirror for i Configuration Considerations....13
5.1 Introduction....13
5.2 Considering What to Mirror....13
5.3 IFS Implementation and Performance Characteristics....13
5.4 Tracking and Resync Considerations....14
5.4.1 Performance Impact of Tracking....14
5.4.2 Resynchronization Considerations....14
6 Db2 Mirror for i Application Considerations....15
6.1 Introduction....15
6.2 Reduce Full Opens....15
6.3 Open Files for Update Only When Necessary....15
6.4 Commitment Control and Locking....15
6.5 Batch Considerations....15
7 Db2 Mirror for i Case Studies....16
7.1 Case Study #1....16
7.1.1 Workload Description....16
7.1.2 Configuration....16
7.1.3 Baseline....16
7.1.4 First Scenario....16
7.1.5 Second Scenario....16
7.1.6 Third Scenario....17
7.2 Case Study #2....20
7.2.1 Workload Description....20
7.2.2 Configuration....20
7.2.3 Baseline....20
7.2.4 First Scenario....20

7.2.5 Second Scenario 20
7.2.6 Third Scenario 20
7.2.7 Summary 21
7.3 Case Study #3 21
7.3.1 Workload Description 21
7.3.2 Configurations 21
7.3.3 Baseline 22
7.3.4 Active/Passive Scenarios 22
7.3.5 Active/Active Scenarios 22
7.3.6 Summary 23
7.4 Client Scenarios 24
7.4.1 Interactive Workload 24
7.4.2 End of Day Batch – Example #1 24
7.4.3 End of Day Batch – Example #2 24
8 Conclusion 25
9 References 26

Figures

Figure 2-1 Db2 Mirror Architecture....8
Figure 7-1 Case Study #1 Results Table....18
Figure 7-2 Case Study #1 CPU and Throughput for 32,000 users across scenarios....18
Figure 7-3 Case Study #1 CPU and Throughput for 48,000 users across scenarios....19
Figure 7-4 Case Study #1 Average Response Times across scenarios....19
Figure 7-5 Case Study #2 Results Table....21
Figure 7-6 Case Study #2 Summary Charts across scenarios....21
Figure 7-7 Case Study #3 Results Table....23
Figure 7-8 Case Study #3 Summary Charts across scenarios....23

1 Introduction

1.1 Purpose

The purpose of this document is to:

- Describe the Db2 Mirror implementation with enough detail that you can understand the performance considerations and case studies that follow.
- Help you prepare for and tune a Db2 Mirror deployment using hardware, configuration, and application performance considerations.
- Provide some performance expectations using case studies.

1.2 Document Responsibilities

The IBM i Development organization is responsible for editing and maintaining the IBM Db2 Mirror for i: Performance Considerations document in collaboration with IBM Power Systems Lab Services. Any contributions or suggestions for additions or edits should be forwarded to Eric Barsness, ericbar@us.ibm.com.

2 What is Db2 Mirror for i?

2.1 Background

IBM Db2 Mirror for i is a major database enhancement for IBM i 7.4. Db2 Mirror is a new IBM i Licensed Program Product that enables near-continuous availability via an IBM i exclusive Db2 active-active two-system configuration.

Db2 Mirror, at its core, is a data-centric solution for continuous availability. Db2 Mirror includes synchronous replication of database files across a tightly coupled active-active configuration. Significant advances in networking technology are a key enabling element of Db2 Mirror. Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) provides far more than an acronym within an acronym. RoCE allows us to connect two IBM i partitions to establish something referred to as a node pair.

Db2 Mirror solves a very important requirement to ensure database objects and data within are identical (in sync) and available across the node pair. Data can be accessed and changed from either node. If there is a planned or unplanned outage, Db2 Mirror resynchronizes the data when the outage ends.

Db2 Mirror can be used to address business requirements such as:

- Avoiding downtime related to hardware or software upgrades
- Avoiding downtime related to maintenance
- Achieving an active-passive solution without having stale data in the mix
- Deploying a true active-active solution

2.2 Db2 Mirror Performance Overview

An important consideration of Db2 Mirror is the impact it can have on the performance of an application. While read-only workloads should not be affected, modify or write (insert, update, and delete) activity on objects that are mirrored will include synchronous replication. This is due to the nature of Db2 Mirror and the requirement to keep objects completely in sync with each other on both nodes while Db2 Mirror is active. Modifications done on one node must also be done synchronously on the other node. The application cannot proceed until the work is complete on both nodes.

While the impact will vary with I/O intensity and the ratio of reads versus writes, the overhead of mirroring database writes could be 2X or higher than non-mirrored equivalent writes for a hypothetical 100% modify workload. The overhead is driven by performing the writes on both nodes and associated high-speed network traffic which coordinates the nodes. Most workloads are a mix of reads and writes and have other application activity and waits which means the overhead impact from Db2 Mirror will be much less than this worst-case scenario. In many cases application optimization and/or faster I/O subsystems can further reduce the impact of enabling Db2 Mirror.

2.3 Architecture

The Db2 Mirror architecture consists of two nodes that are paired together to create a synchronous environment. The nodes are independent, and both can access and update the data that is synchronously replicated in both directions. Db2 Mirror supports replication of data in SYSBAS and in independent auxiliary storage pools (IASPs). Applications can use either SQL or traditional record level access (RLA) to work with replicated data.

For example, Figure 2-1 shows separate instances of the same application running on each node using a synchronously replicated database file. The database file can exist either in SYSBAS or within an IASP. When Row A is changed on Node 1, it is synchronously written to the file on both Node 1 and Node 2. When Row B is changed on Node 2, it is synchronously written to the file on both Node 2 and Node 1.

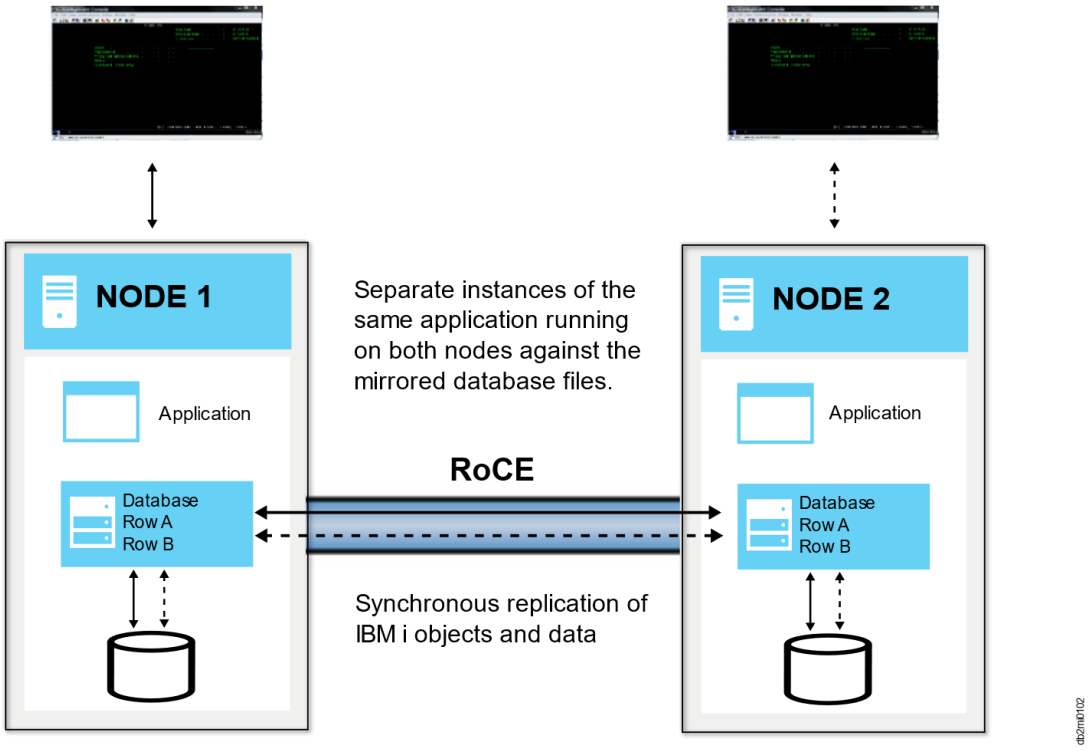


Figure 2-1 Db2 Mirror Architecture

2.4 RDMA over Converged Ethernet (RoCE)

The two nodes in a mirrored pair require a high-speed network connection to perform synchronous data operations. The high-speed connection interface being used is Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE). Adapters that support RoCE must be configured for both nodes.

The ports on the RoCE adapters can be cabled directly together or connected through a switch. IP addresses are configured on the line description associated with each port of the RoCE adapters.

A pair of IP addresses, one from each node, is used to identify each physical RDMA link between the two nodes.

2.5 Mirrored Database Performance Overhead

The synchronous replication design for handling database objects ensures that data is always correctly mirrored on both nodes. Overhead is incurred by RDMA communication flows to acquire and later release necessary row locks while accomplishing mirrored actions on each node.

Individual data modifications (insert, update, delete) effectively occur serially. A change on the local node may execute quickly, but the equivalent change must also be enacted via RDMA on the remote node. The remote action is followed by feedback to the local node (again via RDMA) to indicate whether the remote change was successful or failed. Failure would necessitate undoing the local change.

3 Db2 Mirror for i Hardware and I/O Considerations

3.1 Introduction

This section describes system level and I/O configuration topics that can affect the performance of Db2 Mirror.

3.2 Partition Configuration

Dedicated processors are recommended to get the best performance for a partition. For more information on the technical reasons refer to the IBM i on Power Performance FAQ.

3.3 I/O Subsystem

The performance of I/O operations is critical for Db2 Mirror since the source side must wait for the target side to complete any mirrored database operation including any required synchronous I/Os. Therefore, having a fast I/O subsystem is critical. Technology such as solid-state drives (SSDs) and flash storage as well as large write caches will help improve the performance of Db2 Mirror.

For example, writes to slower performing disks can degrade application response times and lead to increased Db2 Mirror CPU overhead. A sample case study which compares Db2 Mirror performance metrics when using solid state drives versus hard disk drives for an online transactional processing (OLTP) workload scenario is discussed in section 7.1 later in this document.

Similar concerns exist during both planned and unplanned outages. Slower performing I/O will extend the period that a node pair is out of sync and lengthen the amount of time required to complete resynchronization processing following an outage.

4 IBM i Configuration Considerations

4.1 Introduction

Each node in an IBM i Db2 Mirror configuration will be subject to the general performance considerations outlined in the IBM i on Power Performance FAQ. This document will in some cases reference pertinent sections of that Performance FAQ while also outlining specific aspects of Db2 Mirror which may impact performance. Ensuring each node is following the best practices outlined in the Performance FAQ will be important to the performance of your Db2 Mirror implementations.

The rest of this section describes IBM i operating system specific areas of configuration related to performance.

4.2 Prestart Job Configuration

A QDBMSRVR job is created on the target node and is associated with the job on the source node when the first database replication-eligible action is taken within the job. By default, the QDBMSRVR jobs run in the QUSRWRK subsystem. For nodes leveraging Db2 Mirror replication, the recommended prestart job settings for QDBMSRVR jobs are:

INLJOBS(50) THRESHOLD(20) ADLJOBS(40)

For example:

*CHGPJE SBSD(QUSRWRK) PGM(QDBMSRVR) STRJOBS(*YES) INLJOBS(50) THRESHOLD(20) ADLJOBS(40) MAXJOBS(*NOMAX) MAXUSE(*NOMAX)*

This will keep a pool of QDBMSRVR jobs available when source node jobs make the initial connection to the target node (during the first database replication-eligible action) and will optimally replenish the pool for workloads that have a spike in Db2 Mirror connections.

To route the QDBMSRVR jobs to a different subsystem, see https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/db2mi/db2mmonitorjobs.htm

Routing to a different subsystem allows you to route the replication work in the QDBMSRVR jobs to a different memory pool and isolate all replication from other work on the target node.

It is also possible to route the QDBMSRVR jobs to different subsystems and potentially memory pools on the target node by User Profile to further isolate the target replication workload from other replication workload and/or another workload running on the target node.

4.3 Journal Configuration

IBM i journals can have their *attributes* replicated between source and target nodes by adding them to the Replication Criteria List (RCL), but updates to journal receivers and their entries are

never replicated. Each node manages its journal receivers and deposits independently. Thus, when mirroring is active and objects are journaled, each transaction will be journaled by both nodes. Ensuring optimal journal performance on both nodes for journaled objects is critical to overall performance of your Db2 Mirror implementation.

Managing journal performance individually on both nodes will be key to overall performance. Many helpful techniques can be found in the Striving for Optimal Journal Performance Redbook, and also in the Journal management and system performance section of the IBM i Knowledge Center. Particularly, consider omitting the *Open/Close* and *Before* journal entries, as they are not generally useful.

5 Db2 Mirror for i Configuration Considerations

5.1 Introduction

This section describes topics and concepts related to configuring Db2 Mirror which can have an impact on the performance experience for your implementation.

5.2 Considering What to Mirror

Mirroring changes to replicated objects consumes system resources and will add overhead to the applications driving those changes. It is important to carefully consider and control what objects will be replicated for your Db2 Mirror node pair. As the number of objects replicated as their associated storage consumption increases, the possible impact that mirroring can have on performance increases accordingly.

The definition of the list of objects to be replicated is determined by three criteria:

1. The default inclusion state, which is defined only at initial set up.
2. System rules.
3. Replication Criteria List (RCL), which can be user defined for eligible objects.

More information on configuring the replication rules for your installation can be found in section 2.4 of the IBM Db2 Mirror for i Getting Started Redbook.

5.3 IFS Implementation and Performance Characteristics

Integrated File System (IFS) objects are made accessible on both Db2 Mirror nodes by using a different technology than the replication eligible IBM i object types. To be accessible from the secondary node, the IFS objects must be contained within an Independent Auxiliary Storage Pool (IASP), and the IASP must be part of a cluster resource group (CRG).

Simultaneous IFS access from both nodes is implemented by using an IFS client/server technology that uses a mutable file system architecture.

For IFS IASPs, the performance might differ depending upon from which node a file system operation is initiated. Users on the server node where the IASP is varied on might experience faster response times than users on the client node.

For more details on the Db2 Mirror implementation for IFS IASPs see the IBM i Db2 Mirror Getting Started Redbook.

5.4 Tracking and Resync Considerations

5.4.1 Performance Impact of Tracking

When mirroring is suspended, the primary node tracks change operations to mirrored objects. Tracking overhead is very minimal. Newly generated or modified mirrored objects (now out of sync) are added to the Object Tracking List (OTL) and all row level modifications of tracked database files are noted as they occur.

5.4.2 Resynchronization Considerations

As part of resuming replication, the blocked node must be brought up to date with any changes that occurred on the tracking node. Resynchronization is the process by which tracked changes are applied to the other node.

The processing involved to accomplish resynchronization depends upon the amount of tracked information that must be propagated to the blocked node to bring all mirrored objects back into sync. Keep in mind that this resynchronization work is additional processing overhead while the tracked node handles on-going business needs. Several ways to reduce the elapsed time required to complete resynchronization are the following:

- Consider isolating replication activity to a different link than the one(s) used for normal database traffic when configuring Network Redundancy Groups. This prevents replication traffic from negatively affecting network latency on the link used for normal database traffic and avoids the possibility that the combined traffic could saturate a single link.
- The user can specify a parallel degree to improve the performance of resynchronization if Db2 Symmetric Multiprocessing (SMP) is installed. However, since the primary node is still running applications, you should consider the performance impact to those applications. Parallel degree for resynchronization is set using the QSYS2.CHANGE_MIRROR procedure.
- Consider dynamically allocating spare CPU and memory resources to the tracked node during resynchronization.

Additional resynchronization best practices recommended to improve performance and reduce the complexity and problems during resynchronization are documented in more detail in the IBM i Db2 Mirror manual.

6 Db2 Mirror for i Application Considerations

6.1 Introduction

This section highlights topics for consideration within your application environment that can have an impact on performance when implementing Db2 Mirror for IBM i.

6.2 Reduce Full Opens

High rates of Database file full opens can lead to poor performance and unnecessary resource consumption on any IBM i partition — even without Db2 Mirror. Applications with excessive rates of full opens where the file is open for update will suffer additional overhead and reduced throughput with Db2 Mirror. Reducing overall database file full open rates is a long-standing best practice for IBM i.

Section 8.16 of the IBM i on Power Performance FAQ has general performance guidance on this topic and how to analyze your application for full opens.

6.3 Open Files for Update Only When Necessary

Opening files for update will cause file opens and any subsequent record locks to be mirrored to the target node. Read operations can remain local and avoid mirror overhead within a program. Therefore, if your processing is only reading rows of the file, but the open indicates updateability, there will be additional open as well as row processing work for mirroring that could be avoided.

6.4 Commitment Control and Locking

Changes to database objects grouped under commitment control will be mirrored with the same isolation level on the target node. Appropriate locks will be acquired on both nodes during the transaction. This includes read for update locks. Using the lowest level of commitment control required for your application's transactional integrity can help ensure optimal performance in general and for Db2 Mirror.

6.5 Batch Considerations

Long running, write-intensive batch applications will experience longer run times when Db2 Mirror is active. Splitting the processing into multiple jobs or threads that each process a subset of the data in parallel can greatly reduce the run time of time sensitive sections of batch processing.

If there are situations during batch processing where new or temporary files are created and the contents of the files are not needed on both nodes while the batch process is running, it is possible to exclude these files from replication while the batch process is running. Add the appropriate files for replication after batch processing has completed and they will then be available on both nodes while avoiding mirror overhead within the batch run.

7 Db2 Mirror for i Case Studies

7.1 Case Study #1

7.1.1 Workload Description

This scenario is an OLTP COBOL workload with embedded native database functions running under commitment control with the workload running on one side of the node pair (Active/Passive). Five concurrent transactions perform database functions configured to approximately 60% read and 40% write (insert/update/delete). Multiple job sets service the configurable number of virtual users to achieve a desired activity threshold.

7.1.2 Configuration

Two pairs of nodes were tested:

Pair 1: The source node and target node were both configured with 2 cores and memory assigned at 0.5MB per user. A Storwize V7000 configured with solid state drives (SSD) was used as storage for both nodes.

Pair 2: The source node and target node were both configured with 2 cores and memory assigned at 0.5MB per user. A Storwize V7000 configured with hard disk drives (HDD) was used as storage for both nodes.

7.1.3 Baseline

Pair 1 (SSD): We simulated 32,000 users at a total system CPU utilization of 45.6.

Pair 2 (HDD): We simulated 32,000 users at a total system CPU utilization of 56.1.

7.1.4 First Scenario

Pair 1 (SSD): We simulated these same 32,000 users after configuring Db2 Mirror, resulting in a CPU utilization of 81.4 on the source node and 56.5 on the target node. The users were able to complete comparable throughput, but response time went up.

Pair 2 (HDD): We simulated these same 32,000 users after configuring Db2 Mirror, resulting in a CPU utilization of 85.2 on the source node and 64.3 on the target node. The users were able to complete comparable throughput, but response time went up.

7.1.5 Second Scenario

We kept all other parameters constant but increased the CPU to 4 cores on all nodes and repeated the Db2 Mirror tests, with the following results (NET: doubling the cores brought CPU utilizations back in line with baselines):

Pair 1 (SSD): We simulated these same 32,000 users, resulting in a CPU utilization of 46.7 on the source node and 29.9 on the target node. The users were able to complete comparable throughput with response times generally like the 2 core Db2 Mirror results from the first scenario.

Pair 2 (HDD): We simulated these same 32,000 users, resulting in a CPU utilization of 52.9 on the source node and 32.9 on the target node. The users were able to complete comparable throughput with response times generally like the 2 core Db2 Mirror results from the first scenario.

7.1.6 Third Scenario

The final test kept all nodes configured at 2 cores but simulated 48,000 users to examine Db2 Mirror characteristics at higher CPU and/or disk utilizations (NET: I/O speed can become a factor at high utilizations):

Pair 1 (SSD):

- Baseline: We simulated 48,000 users at a total system CPU utilization of 64.8.
- Db2 Mirror: We simulated 48,000 users at system CPU utilization of 100.0 on the source node and 77.2 on the target node. The users were able to complete comparable throughput, but response times were significantly higher than the first scenario of 32,000 users.

Pair 2 (HDD):

- Baseline: We simulated 48,000 users at a total system CPU utilization of 79.1.
- Db2 Mirror: We simulated 48,000 users at system CPU utilization of 72.1 on the source node and 54.7 on the target node. Both throughput and response times were significantly impacted as disk could not keep up with transaction rates.

The results summary table for all test scenarios is found in Figure 7-1. Charts of comparable results include Figure 7-2 for 32,000 users across scenarios, Figure 7-3 for 48,000 users across scenarios, and Figure 7-4 for average response times across all user loads and associated scenarios.

32,000 users	Source CPU%	Target CPU%	Throughput	Average Response Time (Sec)
Pair 1 Baseline	45.6		38990	0.004
Pair 2 Baseline	56.1		38974	0.012
Pair 1 Scenario #1	81.4	56.5	38994	0.011
Pair 2 Scenario #1	85.2	64.3	38686	0.452
Pair 1 Scenario #2	46.7	29.9	39000	0.007
Pair 2 Scenario #2	52.9	32.9	39074	0.455

48,000 users	Source CPU%	Target CPU%	Throughput	Average Response Time (Sec)
Pair 1 Baseline	64.8		58485	0.006
Pair 2 Baseline	79.1		58454	0.021
Pair 1 Scenario #3	100	77.2	58485	0.525
Pair 2 Scenario #3	72.1	54.7	30433	2.476

Figure 7-1 Case Study #1 Results Table

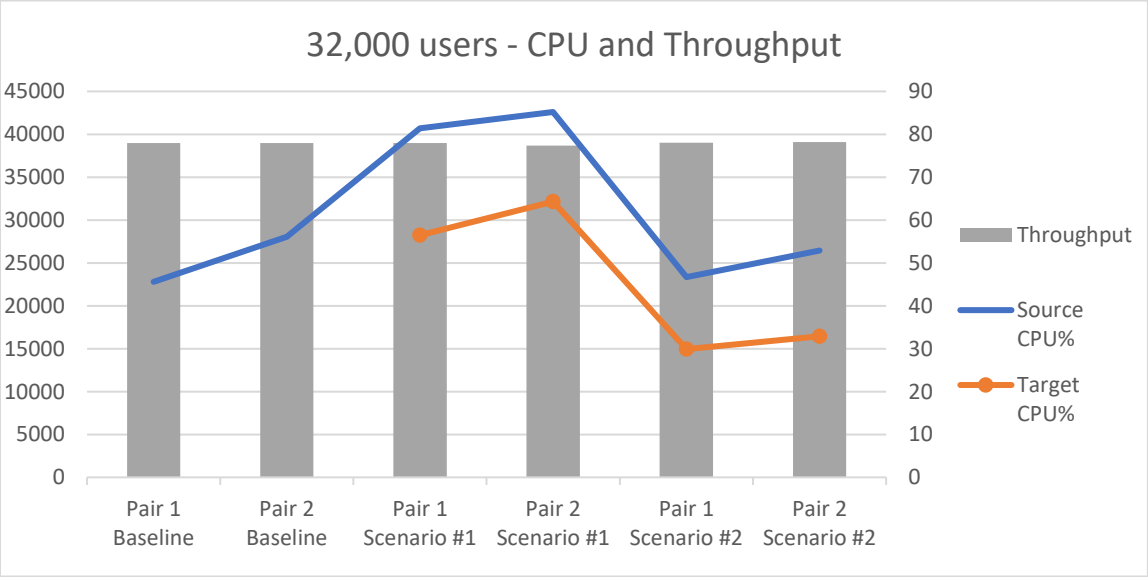


Figure 7-2 Case Study #1 CPU and Throughput for 32,000 users across scenarios

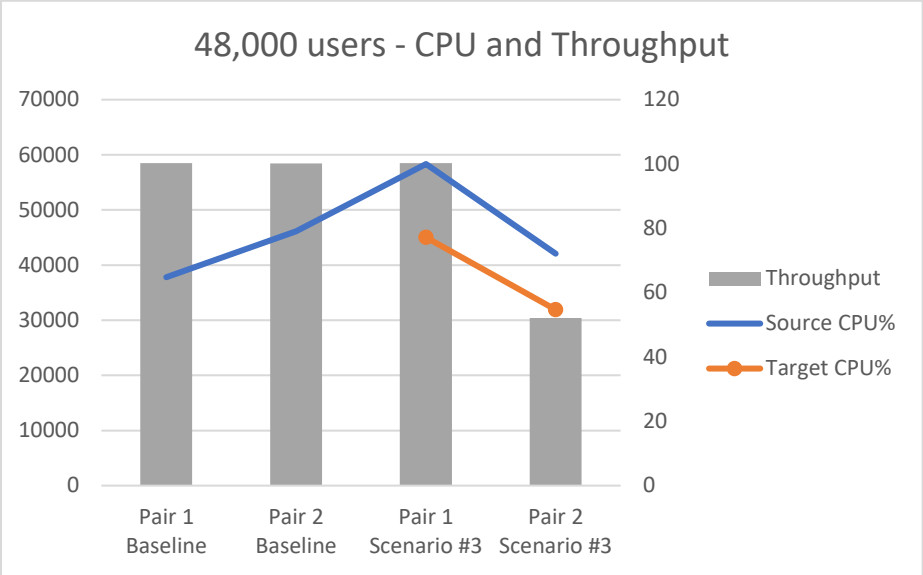
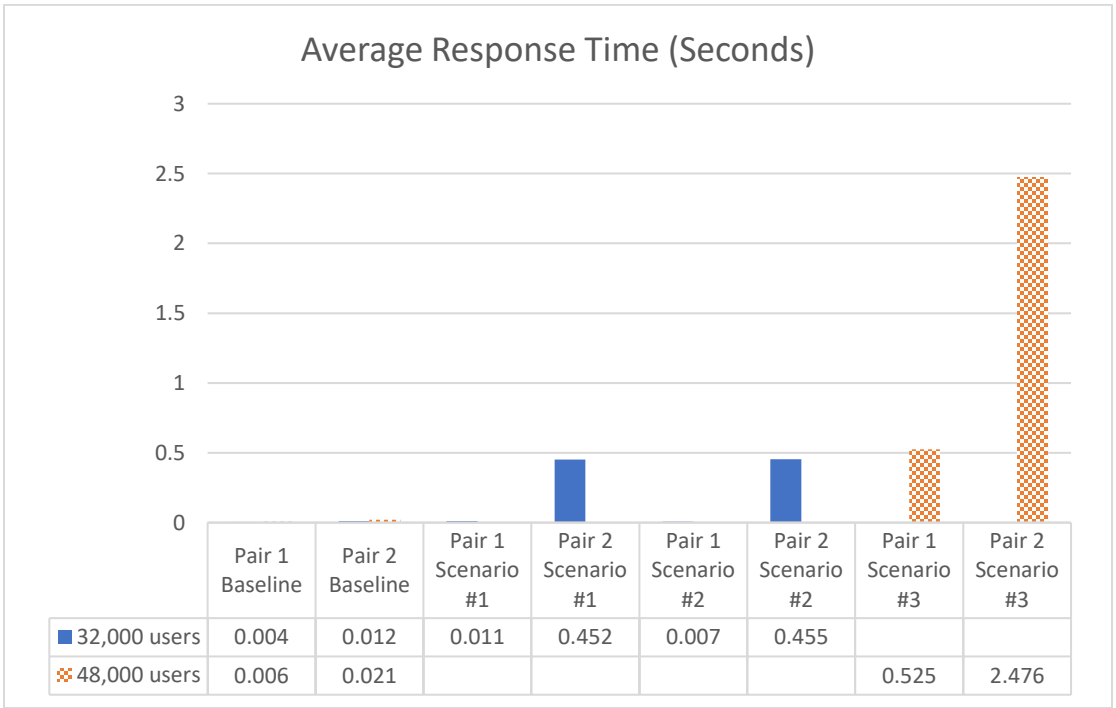


Figure 7-3 Case Study #1 CPU and Throughput for 48,000 users across scenarios



	Pair 1 Baseline	Pair 2 Baseline	Pair 1 Scenario #1	Pair 2 Scenario #1	Pair 1 Scenario #2	Pair 2 Scenario #2	Pair 1 Scenario #3	Pair 2 Scenario #3
32,000 users	0.004	0.012	0.011	0.452	0.007	0.455		
48,000 users	0.006	0.021					0.525	2.476

Figure 7-4 Case Study #1 Average Response Times across scenarios

7.1.7 Summary

This case study shows that for our OLTP COBOL workload the overhead incurred by Db2 Mirror replication in terms of application response time and CPU usage increase could be compensated by a faster disk I/O subsystem and by doubling CPU resources. With CPU resources held constant on the SSD-Flash I/O pair, the workload was able to fully utilize the available CPU on the source node but response time was significantly impacted. Holding the CPU resources constant on the HDD pair resulted in substantial impact to throughput and response times.

7.2 Case Study #2

7.2.1 Workload Description

This is an SQL based OLTP workload consisting of 23 different transactions implemented via SQL stored procedures which execute multiple SQL statements per transaction under commitment control. The workload runs actively on one side of the node pair (Active/Passive). Ten of the transactions result in database changes (Insert/Update/Deletes) as well as reads, while the other 13 are read-only transactions.

The workload is configurable by the number of jobs sets executing and a specific ratio of transactions in order to achieve a consistent transaction rate and mix for each test.

7.2.2 Configuration

The source node had 22 cores with 964 GB of memory in the application pool.

The target node had 11 cores also with 964 GB of memory, but in the *BASE pool where the QDBMSRVR replication jobs run by default.

7.2.3 Baseline

We executed 82 job sets resulting in ~29,900 transactions/second (TPS).

7.2.4 First Scenario

We executed these same 82 job sets after configuring Db2 Mirror, resulting in a decrease in throughput to ~23,770 TPS. The CPU utilization was approximately the same, but response time went up.

7.2.5 Second Scenario

We increased the job sets to 104 in order to achieve approximately the same transaction rate as the baseline test, while keeping the number of cores on the source node the same at 22. This test saw a transaction rate of ~29,870 TPS, which was like the baseline. CPU increased from the baseline as well as response time.

7.2.6 Third Scenario

For this test we increased the number of cores in the source node and adjusted the number of job sets so that the transaction rate and CPU utilization were like the baseline. By using 92 jobs sets and 25 cores on the source node, we saw a transaction rate of 29,820 TPS with a response time closer to the baseline.

The results summary table for all test scenarios is found in Figure 7-5. Charts of comparable results are in Figure 7-6.

	Source CPU%	Target CPU%	TPS	Response Time (ms)	Cores Src/Tgt	Job Sets
Baseline	74.4	N/A	29,900	3.24	22 / NA	82
Scenario #1	74.3	20.9	23,770	4.24	22 / 11	82
Scenario #2	83.9	26.6	29,870	4.34	22 / 11	104
Scenario #2	75.5	26.0	29,820	3.88	25 / 11	92

Figure 7-5 Case Study #2 Results Table

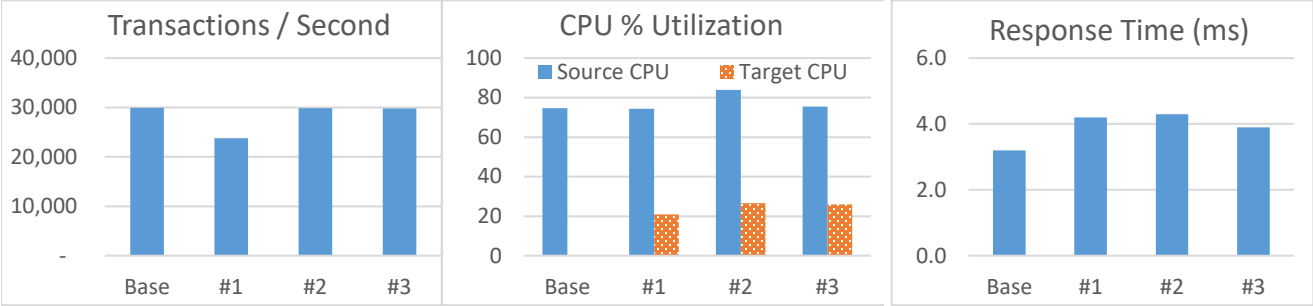


Figure 7-6 Case Study #2 Summary Charts across scenarios

7.2.7 Summary

What we found with this case study is ~14% additional cores were required on the source node in order to maintain about the same overall system CPU utilization and throughput. This workload also required additional jobs to maintain the same throughput due to increased response time. Workloads that have a set of client jobs processing work that are always busy and not typically waiting for work will likely require additional jobs in order to maintain the same throughput. For a workload that uses connection pooling, this could mean an increased value for maximum jobs in the pool.

7.3 Case Study #3

7.3.1 Workload Description

This is the same SQL based OLTP workload as Case study #2. In this study, we ran the workload without Db2 Mirror, with Db2 Mirror in Active/Passive mode and also in Active/Active mode.

7.3.2 Configurations

When running without Db2 Mirror active, the source node had 32 cores with 487 GB of memory in the application pool.

When running in Active-Passive mode, the target node varied from 6-8 cores also with 487 GB of memory but in the *BASE pool where the replication jobs run by default.

When running Active-Active mode, the source and target nodes varied from 21-22 cores memory ranged from 487-584 GB.

We executed a varying number of job sets for each test attempting to achieve ~44,500 transactions/second for each test and keeping CPU utilization around 79%.

7.3.3 Baseline

This test achieved 44,500 TPS (Transactions per Second) with CPU running at 78.9% of the 32 cores with an application average response time of 3.6 milliseconds.

7.3.4 Active/Passive Scenarios

The first version ran with 38 cores on the source node and 8 cores on the target node achieving 44,600 TPS at an average response time of 4.7 milliseconds. CPU utilization on the source node was 78.5% of the 38 cores and 56.32% of the 8 cores on the target node.

The second version ran with 38 cores on the source node and 6 cores on the target node achieving 44,700 TPS at an average response time of 5.0 milliseconds. CPU utilization on the source node was 79.38% of the 38 cores and 78.74% of the 6 cores on the target node.

7.3.5 Active/Active Scenarios

The first version ran with 22 cores on each node and used 487 GB of memory on each node, but split this memory 80/20%, with 390 GB in the application pool and 97 GB in *BASE where the replication jobs run. This test saw 43,500 TPS at 7.2 and 6.5 millisecond response times on the primary and secondary nodes respectively. CPU utilization was 80.16% on the primary node and 81.31% on the secondary node. To achieve the same TPS as other tests would have required an additional core on both nodes.

The second version also ran with 22 cores on each node and used 487 GB of memory on each node again but had the application work and replication jobs share the same memory pool. This test achieved 44,600 TPS at a 5.2 millisecond response time on each node. CPU utilization was 79.62% on the primary node and 79.04% on the secondary node.

The third version again ran with 22 cores on each node but used 584 GB of memory (120% of baseline), with 487 GB in the application memory pool and 97 GB in *BASE for the replication jobs on each node. This test achieved 44,300 TPS and 5.6 / 5.1 millisecond response times with CPU utilization of 77.21% on the primary node and 77.30% on the secondary node.

The fourth Active-Active version also ran with 22 cores on each node and used 584 GB of memory, but with the memory in one pool shared by the application and replication jobs. This test achieved 44,400 TPS at an average response time of 4.3 / 4.2 milliseconds. CPU utilization on the primary node was 76.08% and 75.80% on the secondary node.

The last Active-Active test ran with 21 cores on each node and 584 GB of memory again with the application jobs and replication jobs sharing a memory pool. This test achieved 44,600 TPS at an average response time of 4.5 milliseconds on each node. CPU utilization was 79.32% on the primary node and 78.99% on the secondary node.

The results summary table for all test scenarios is found in Figure 7-7. Charts of comparable results are in Figure 7-8.

	Source CPU%	Target CPU%	TPS	Response Time (ms)	Cores Src/Tgt	Job Sets
Baseline	78.90	N/A	44,500	3.7	32 / NA	140
Act/Pas #1	78.50	56.32	44,600	4.7	38 / 8	164
Act/Pas #2	79.38	78.74	44,700	5.0	38 / 6	172

	Primary CPU%	Secondary CPU%	TPS	Response Time (ms)	Cores Prim/Sec	Job Sets
Act/Act #1	80.16	81.31	43,500	7.2 / 6.5	22 / 22	108 / 108
Act/Act #2	79.62	79.04	44,600	5.2 / 5.2	22 / 22	90 / 90
Act/Act #3	77.21	77.30	44,300	5.6 / 5.1	22 / 22	86 / 86
Act/Act #4	76.08	75.80	44,400	4.3 / 4.2	22 / 22	72 / 72
Act/Act #5	79.32	78.99	44,600	4.5 / 4.5	21 / 21	76 / 76

Figure 7-7 Case Study #3 Results Table

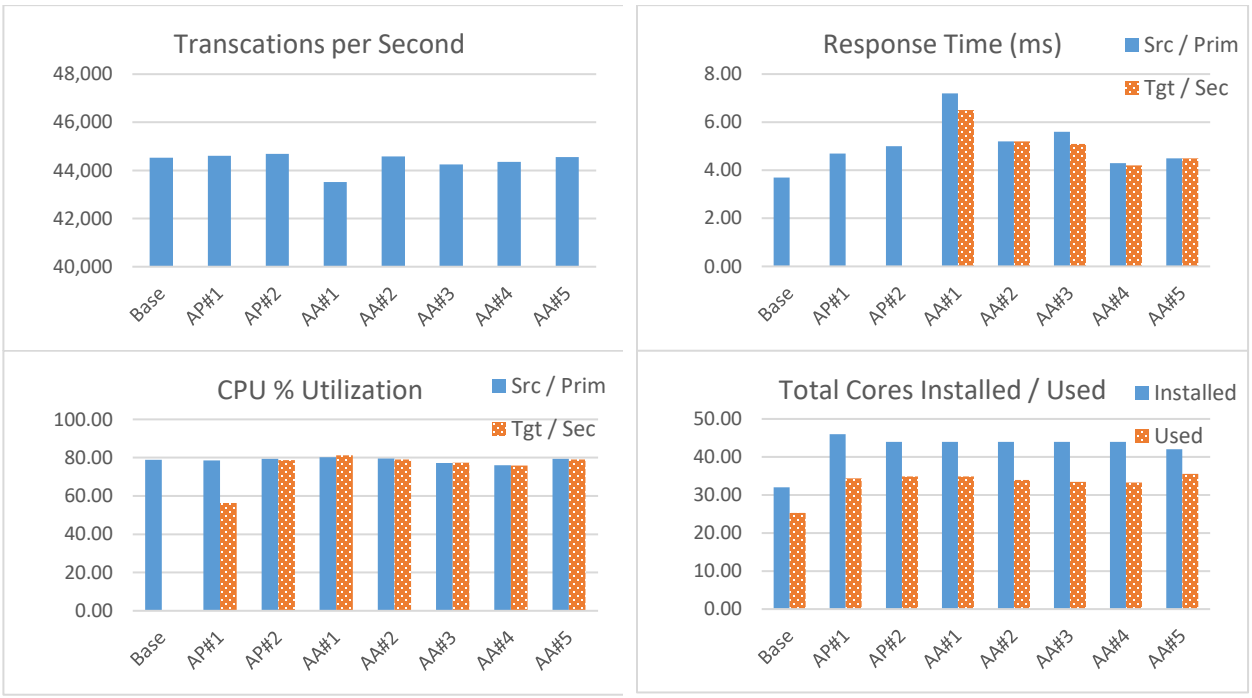


Figure 7-8 Case Study #3 Summary Charts across scenarios

7.3.6 Summary

The findings for the active/passive scenarios compared to the baseline was very similar to the previous case study. For the active/active scenarios, having the QDBMSRVR replication jobs share

the same memory pool as the application resulted in the best performance. Having additional memory in the pool that is again shared provided even slightly better performance.

7.4 Client Scenarios

This section includes workloads that were based on actual client requirements or client-based workloads. The summaries are intended to give real-world performance results without sharing any sensitive information. Therefore, these write-ups are much less detailed than the internal workload case studies.

7.4.1 Interactive Workload

This workload simulated interactive users in a retail banking environment. It was run active-passive. The application was written in RPG and uses native I/O to access database files. The workload is 62% reads and 38% writes. The client worked with IBM Lab Services to analyze the performance of their applications while running with Db2 Mirror. After a series of application changes were implemented, the results were:

- Throughput 19% lower (using Db2 Mirror compared to baseline behavior)
- Response time 24% longer
- CPU utilization 1% higher

Additional improvements to the application were identified but not implemented because the results had achieved the client's performance requirements.

7.4.2 End of Day Batch – Example #1

This workload simulated end of day batch and was run active-passive. It was a more write intensive workload, consisting of 37% reads and 63% writes. No application changes were made to improve performance. The Set Object Access (SETOBJACC) command was used on the target partition to pre-load objects into memory. The results were:

- Run time 34% longer
- CPU consumption 19% higher

7.4.3 End of Day Batch – Example #2

This workload simulated a batch banking application and was run active-passive. It uses RPG programs and native I/O to access the database. While the workload is read intensive (82% reads and 18% writes), most of the read activity is read for update, which causes read locks to be acquired on both partitions. Therefore, the workload is nearly 100% writes. The results were:

- Run time 86% longer
- CPU utilization 0.8% lower

No application changes made to optimize performance of this workload. Changing the read for updates activity to be read-only would significantly improve the performance of this application running with Db2 Mirror enabled.

8 Conclusion

Performance is critical to successfully deploying Db2 Mirror. This document describes ways to configure your system, partition, and Db2 Mirror to help improve performance. It also includes potential application changes to reduce the impact of running with Db2 Mirror.

References for more information are included in the next section. If you need additional help in assessing the potential impact of implementing Db2 Mirror or identifying ways to improve the performance of Db2 Mirror in your environment, IBM Power Systems Lab Services can help either through general consulting or through our Db2 Mirror workshop.

The Lab Services' Db2 Mirror for i Readiness Assessment Workshop is a two week, hands-on workshop where IBM provides the expertise and test environment and you provide your application(s) and test data. The first week covers planning, implementation, and setting up libraries in a Db2 Mirror environment. The second week focuses on application and performance requirements and testing.

You can contact IBM Power Systems Lab Services at ibmsls@us.ibm.com.

9 References

The following is a list of IBM documents that are good references:

IBM i on Power – Performance FAQ

IBM i Db2 Mirror Manual

IBM Db2 Mirror for i Getting Started Redbook

Db2 Mirror for IBM i Readiness Assessment

Monitoring the Db2 Mirror environment: Db2 Mirror jobs

Striving for Optimal Journal Performance Redbook

Journal Management and System Performance

Disclaimer – IBM Db2 Mirror for i: Performance Considerations

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This information may include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or programs(s) at any time without notice. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT.

IBM shall have no responsibility to update this information. IBM products are warranted according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. IBM is not responsible for the performance or interoperability of any non-IBM products discussed herein.

The performance data contained herein was obtained in a controlled, isolated environment. Actual results that may be obtained in other operating environments may vary significantly. While IBM has reviewed each item for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere.

Statements regarding IBM’s future direction and intent are subject to change or withdrawal without notice and represent goals and objectives only.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

IBM Corporation
Systems and Technology Group
Route 100
Somers, New York 10589

Produced in the United States of America May 2020

10032410USEN-00