

DB2 HADR – TUNING IT TO THE MAX

Make your Db2 HADR environment fly

Dale McInnis

STSM

dmcinnis@ca.ibm.com

WHO AM I

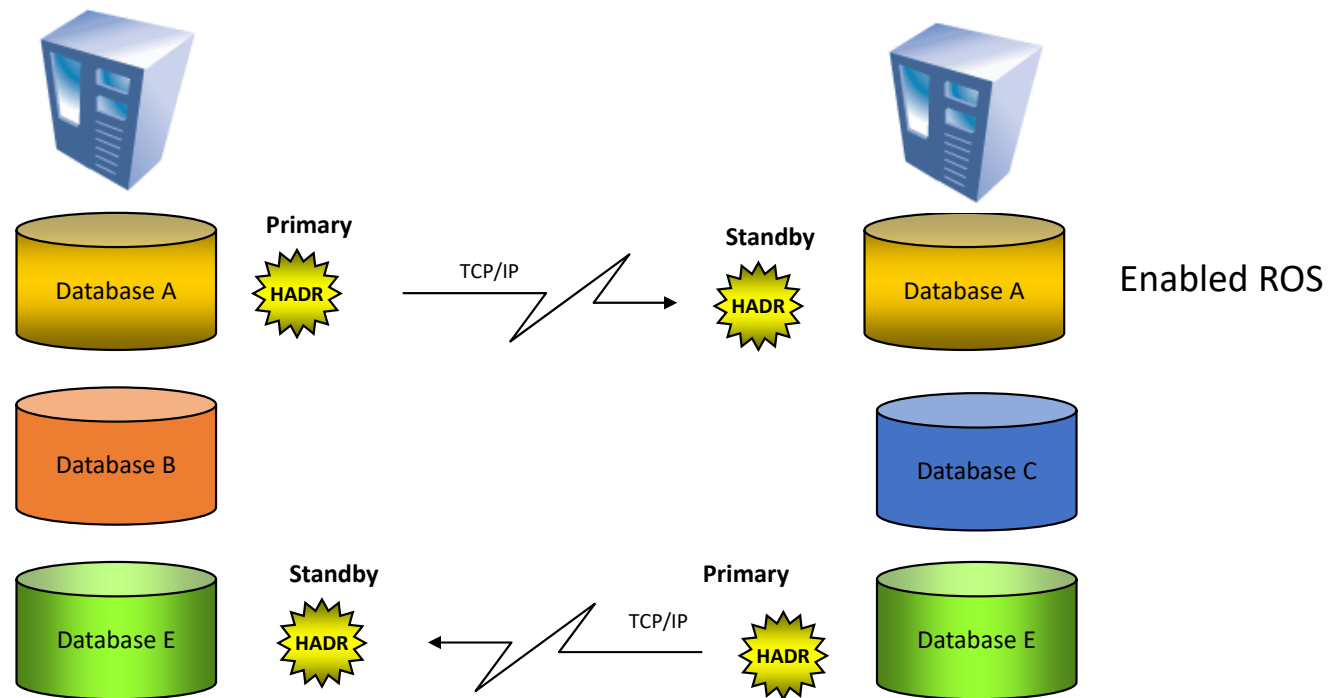
- » Team Lead Db2 Client Adoption Engineering Team
- » 38 years with IBM, 25 years at the Db2 LUW High Availability Architect
- » Information Systems Professional (I.S.P) Certified
- » Information Technology Certified Professional (ITCP)
- » 11 Patents in database resiliency
- » Published author
- » IDUG Speaker Hall of Fame member
- » Ph.D. Student at the Ontario Technical University



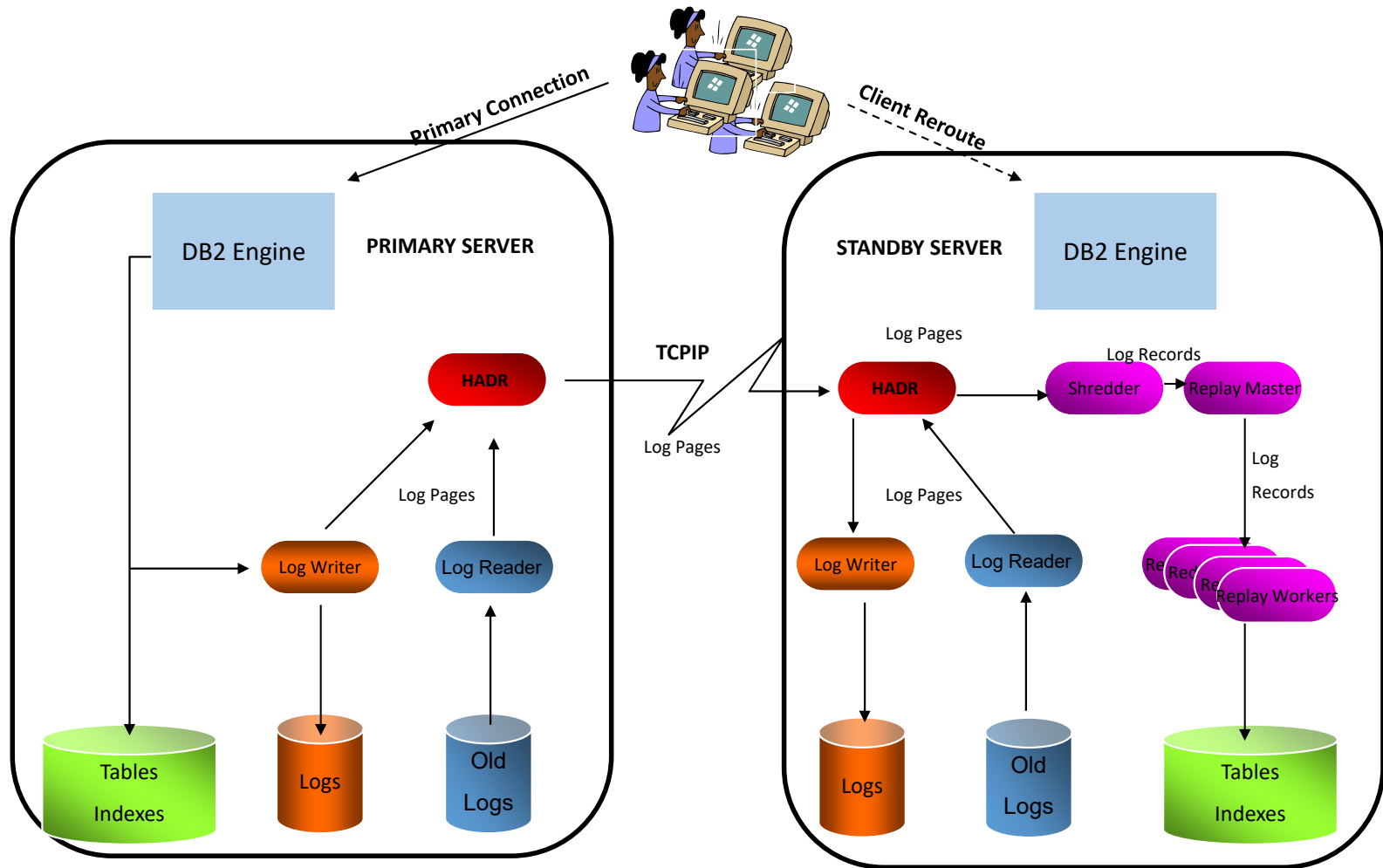
Agenda

- What is HADR
- How to configure HADR for optimal performance
- Monitoring options
- Best Practices

HADR replication takes place at the database level.



HADR Design



What's replicated, what's not?

- Logged operations are replicated
 - Example: DDL, DML, table space and buffer pool creation/deletion.
- Not logged operations are not replicated.
 - Example: database configuration parameter. not logged initially table, UDF libraries.
- Index pages are not replicated unless LOGINDEXBUILD is enabled
 - Ensure logsecond is maxed out as index rebuild is a single transaction
- How do I prevent non-logged operations?
 - Enable BLOCKNONLOGGED db cfg parameter

HADR Setup Fits on One Slide

Primary Setup

db2 backup db hadr_db online to backup_dir

db2 update db cfg for hadr_db using

```
HADR_LOCAL_HOST  host_a
HADR_REMOTE_HOST host_b
HADR_LOCAL_SVC   svc_a
HADR_REMOTE_SVC  svc_b
HADR_REMOTE_INST inst_b
HADR_TIMEOUT     120
HADR_SYNCMODE    ASYNC
```

db2 start hadr on database hadr_db as
primary

Standby Setup

db2 restore db hadr_db from backup_dir

db2 update db cfg for hadr_db using

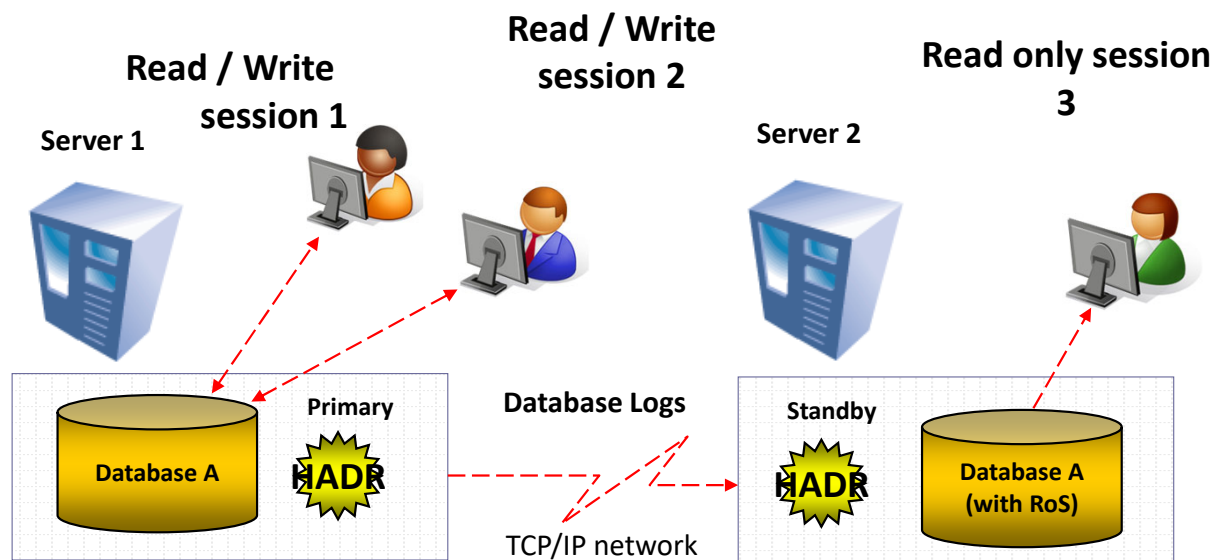
```
HADR_LOCAL_HOST  host_b
HADR_REMOTE_HOST host_a
HADR_LOCAL_SVC   svc_b
HADR_REMOTE_SVC  svc_a
HADR_REMOTE_INST inst_a
HADR_TIMEOUT     120
HADR_SYNCMODE    ASYNC
```

db2 start hadr on database hadr_db as
standby

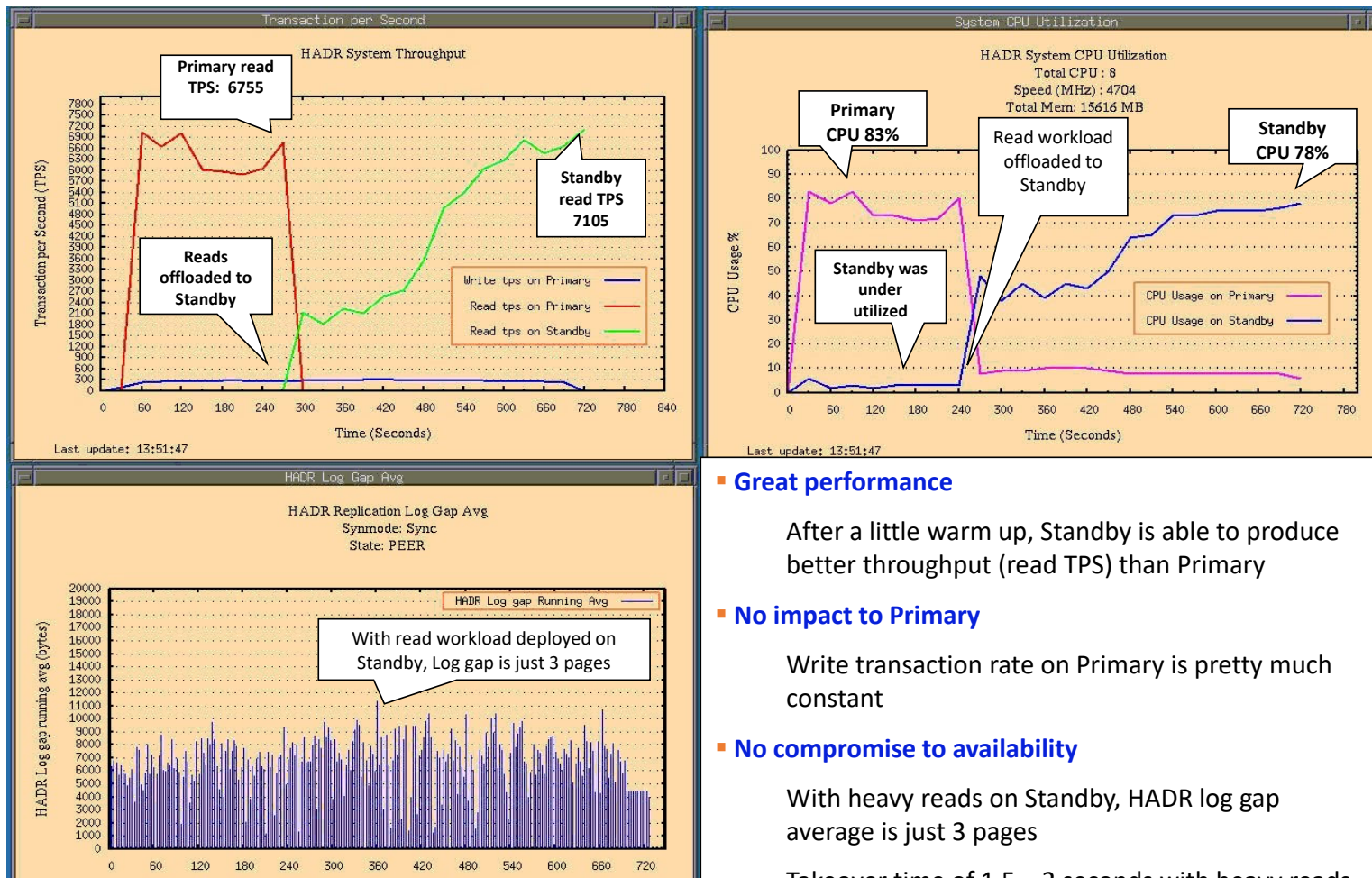


HADR Read On Standby (RoS)

- Reads on Standby provides high availability, disaster recovery and allows reporting workloads.
- Improve resource utilization on your HA or DR hardware
- Offload reporting work from your primary, Increase capacity of HADR system
- Maximize Return on Investment and decrease Total Cost of Ownership
- UR isolation level only



HADR Performance With Reads on Standby



- **Great performance**

After a little warm up, Standby is able to produce better throughput (read TPS) than Primary

- **No impact to Primary**

Write transaction rate on Primary is pretty much constant

- **No compromise to availability**

With heavy reads on Standby, HADR log gap average is just 3 pages

Takeover time of 1.5 – 2 seconds with heavy reads on Standby, comparable with no reads.

Tools available to measure HADR related performance

- Check out <https://ibm.github.io/db2-hadr-wiki/hadrPerf.html>
- **DB2 Log Scanner** to analyze database workload
<https://ibm.github.io/db2-hadr-wiki/db2logscan.html>
- **HADR Simulator** to measure network and disk speed
<https://ibm.github.io/db2-hadr-wiki/hadrSimulator.html>
- **HADR Calculator** to estimate performance of various HADR sync modes
https://ibm.github.io/db2-hadr-wiki/db2logscan.html#HADR_Calculator

ALL HADR Perf tools are now shipped as part of Db2 installation

4 steps to Db2 HADR Perf Tuning

- Step 1: Know Your Workload

- Db2 log scanner

- Step 2: Know Your Disks

- Db2 HADR Simulator

- Step 3: Know Your Network

- Ping + DB2 HADR Simulator

- Step 4: Know Your Sync Modes

- Db2 HADR Calculator

Step 1: Know Your Workload

[DB2 log scanner](#) – analyze the commit patterns in your transactions logs

Located in sqllib/bin directory

```
db2logscan S0003158.LOG S0003159.LOG ... S0003214.LOG > daily.scan
```

To scan a large number of files, you can create a file with the file names in it, one per line, then feed the file to scanner via the -f option. If some files need to be retrieved from archive, you can use the [-retrieve option](#).

Please save all output files from the scanner. We will need them in later steps.

Step 2: Know Your Disks

- [DB2 HADR simulator](#) to measure performance of your logging disks. Disk speed is modeled as:

$\text{write_time} = \text{per_write_overhead} + \text{data_amount} / \text{transfer_rate}$

- Command for small write (single page write to calculate write_overhead):

```
simhadr -write <logPath>/testFile -flushSize 1
```

Sample output:

```
Total 9409 writes in 4.000160 seconds, 0.000425 sec/write, 1 pages/write  
Total 38.539264 MBytes written in 4.000160 seconds. 9.634431 MBytes/sec
```

This means per_write_overhead is 0.000425 second (from the highlighted "sec/write" field).

Command for big write (1000 pages per write to calculate transfer rate):

```
simhadr-write <logPath>/testFile -flushSize 1000
```

Sample output:

```
Total 174 writes in 4.014913 seconds, 0.023074 sec/write, 1000 pages/write  
Total 712.704000 MBytes written in 4.014913 seconds. 177.514183 MBytes/sec
```

This means transfer rate is 177.514183 MBytes/sec (from the highlighted "MBytes/sec" field).

Step 3: Know Your Network

- Measure HADR primary/standby network speed. Network speed is specified by two numbers: send rate and round trip time. Round trip time can be easily measured by the "ping" command. Send rate may be sensitive to socket buffer size. A special procedure is used for the measurement.
- See [Using HADR simulator to determine optimal TCP window size](#) for detailed instructions. This step only gives the tentative socket buffer size based on TCP requirement. Final socket buffer size will also depend on the workload and flush size. It will be determined in step 4. Remember to configure DB2 with the chosen socket buffer size. Otherwise, DB2 won't get the send rate from the simulator test. See [TCP Tuning](#) for more info.
- In step 4, use "-network <send_rate> <round_trip_time>" for network speed option for HADR calculator.

Step 4: Know Your Sync Modes

[HADR calculator](#)

Usage: `hadrCalculator.pl [options] <inputFile1> <inputFile2> ... <inputFileN>`

HADR calculator annotates db2logscan output with theoretical HADR data rate.

`-syncmode <s>` Specify one or more HADR sync modes. Modes are SYNC, NEARSYNC, ASYNC, or SUPERASYNC (case insensitive). Multiple modes can be specified as comma delimited list. Default "SYNC,NEARSYNC,ASYNC".

`-network <f1> <f2>` Specify primary-standby network speed as <f1> MBytes/sec with round trip time of <f2> second.

`-disk <f1> <f2>` Specify disk write speed as <f1> MBytes/sec with overhead of <f2> second per write.

`hadrCalculator.pl -disk 200 0.001 -network 10 0.1 <scanOutFile1> ... <scanOutFileN> > daily.scan.calc`

Step 4: Know Your Sync Modes

Now look at the calculator output. Look for question mark ("?") in the output. Calculator uses question marks ?, ??, and ??? to indicate small, medium, and heavy impact to applications when HADR is enabled. If no question mark is found, then all is good (expect no impact to applications at all).

Once you have chosen a sync mode, find the max flush size section at end of calculator output:

SYNC	Max flush size: predicted 360 pages, workload max 1126 pages
NEARSYNC	Max flush size: predicted 360 pages, workload max 1126 pages
ASYNC	Max flush size: predicted 17 pages, workload max 1126 pages

HADR socket buffer size should be at least the "predicted" size of your chosen sync mode. This is the expected flush size for the given sync mode. Due to variations in workload, a larger size (such as workload max, which happens when log writing is slower than predicted) is preferred. Both predicted and workload max flush sizes are computed from average transaction size of log rate sample intervals. Actual workload may have peak size above the average. Thus a number even higher than workload max is preferred, as long as the size is reasonable (no more than 32MB). In the above example, a 2000 page (8MB) socket buffer size is recommended.

For verification, run HADR simulator using the chosen sync mode, socket buffer size, disk speed, and predicted flush size to confirm that the system will perform as expected (compare the throughput from simulator to the line with the same flush size in calculator output). Even though this is simulation, the network is stressed with real data. A simulator run with the final parameters is strongly recommended before applying the parameters to the database.

```
+ simhadr -lhost vaz-lpd-380-repl -lport 60015 -rhost dcalpd3622-repl -rport 60016 -role primary -syncmode SUPERASYNC -t 60
```

Measured sleep overhead: 0.000058 second, using spin time 0.000069 second.

flushSize = 16 pages

Resolving local host vaz-lpd-380-repl via gethostbyname()

hostname=vaz-lpd-380-repl.amat.com

address_type=2 address_length=4

address: 10.168.225.53

Resolving remote host dcalpd3622-repl via gethostbyname()

hostname=dcalpd3622-repl.amat.com

address_type=2 address_length=4

address: 10.41.220.247

Socket property upon creation

BlockingIO=true

NAGLE=true

SO_SNDBUF=16384

SO_RCVBUF=87380

SO_LINGER: onoff=0, length=0

Binding socket to local address.

Listening on local host TCP port 60015

Connected.

Calling fcntl(O_NONBLOCK)
Calling setsockopt(TCP_NODELAY)
Socket property upon connection
BlockingIO=false
NAGLE=false
SO_SNDBUF=46080
SO_RCVBUF=369280
SO_LINGER: onoff=0, length=0
Sending handshake message:
syncMode=UPERASYNC
flushSize=16
connTime=2024-12-10_10:21:54_PST
Sending log flushes. Press Ctrl-C to stop.
UPERASYNC: Total 1300299776 bytes in 60.041951 seconds, 21.656521 MBytes/sec
Total 19841 flushes, 0.003026 sec/flush, 16 pages (65536 bytes)/flush
Socket send buf size (bytes): requested 0(no request), actual 46080
Socket recv buf size (bytes): requested 0(no request), actual 369280
Total 1300299776 bytes sent in 60.041951 seconds. 21.656521 MBytes/sec
Total 20871 send calls, 62.302 KBytes/send,
Total 1018 congestions, 59.846875 seconds, 0.058789 second/congestion

Total 0 bytes recv in 60.041951 seconds. 0.000000 MBytes/sec
Total 0 recv calls, 0.000 KBytes/recv

Distribution of log write size (unit is byte):
Total 19841 numbers, Sum 1300299776, Min 65536, Max 65536, Avg 65536
Exactly 65536 19841 numbers 100%

Distribution of log shipping time (unit is second):
Total 19841 numbers, Sum 60.038802, Min 0.000004, Max 0.228480, Avg 0.003026
From 0.000004 to 0.000007 5882 numbers 29%
From 0.000008 to 0.000015 12318 numbers 62%
From 0.000016 to 0.000031 549 numbers 2%
From 0.000032 to 0.000063 70 numbers 0%
From 0.000064 to 0.000127 5 numbers 0%
From 0.000128 to 0.000255 1 numbers 0%
From 0.000256 to 0.000511 1 numbers 0%
From 0.002048 to 0.004095 2 numbers 0%
From 0.004096 to 0.008191 7 numbers 0%
From 0.008192 to 0.016383 57 numbers 0%
From 0.016384 to 0.032767 52 numbers 0%
From 0.032768 to 0.065535 703 numbers 3%
From 0.065536 to 0.131071 181 numbers 0%
From 0.131072 to 0.262143 13 numbers 0%

Distribution of congestion duration (unit is second):
Total 1018 numbers, Sum 59.846875, Min 0.002262, Max 0.228443, Avg 0.058789
From 0.002048 to 0.004095 2 numbers 0%
From 0.004096 to 0.008191 7 numbers 0%
From 0.008192 to 0.016383 57 numbers 5%
From 0.016384 to 0.032767 52 numbers 5%
From 0.032768 to 0.065535 703 numbers 69%
From 0.065536 to 0.131071 187 numbers 18%
From 0.131072 to 0.262143 10 numbers 0%

Distribution of send size (unit is byte):
Total 20871 numbers, Sum 1300299776, Min 24, Max 65536, Avg 62302
From 16 to 31 1 numbers 0%
From 64 to 127 3 numbers 0%
From 128 to 255 5 numbers 0%
From 256 to 511 8 numbers 0%
From 512 to 1023 13 numbers 0%
From 1024 to 2047 39 numbers 0%
From 2048 to 4095 67 numbers 0%
From 4096 to 8191 120 numbers 0%
From 8192 to 16383 260 numbers 1%
From 16384 to 32767 513 numbers 2%
From 32768 to 65535 1026 numbers 4%
From 65536 to 131071 18816 numbers 90%

Requests for Enhancements 1/3

Home / All ideas / DB24LUW-I-1432

Add a new idea

✓ Subscribed

12

VOTED

Status

Future consideration

Workspace

[Db2](#)

Components [High Availability and/or Disaster Recovery](#)

Created by

Guest

Created on

Jan 25, 2022

Backup on HADR Standby

[See this idea on ideas.ibm.com](#)

Backup consume resources in primary. If we offload backup in Standby then Primary can be easily served for application use alone . Currently we were able to take backup only in primary and all workloads along with Backup utilise CPU resources .If we have capability or feature to enable which allows Backup to run Standby then it will be easy for Primary server to focus on application workloads alone.

Needed By

Week

COMMENTS 0

MERGED IDEAS 1

Requests for Enhancements 2/3

22

VOTE

Status **Not under consideration**

Workspace [Db2](#)

Components [High Availability and/or Disaster Recovery](#)

Created by [Guest](#)

Created on Jul 27, 2018

DB2 HADR Standby Database act as CDC Source

[See this idea on ideas.ibm.com](#)

We use DB2 ,DB2 HADR and CDC in our production system. Now we can only use the DB2 HADR primary database as CDC source. DB2 HADR standby database can not support acting as CDC source now. So we hope add a new feature,ie : DB2 HADR Standby database can act as CDC source.

COMMENTS 2

Requests for Enhancements 3/3

✓ Idea created: DB24LUW-I-1984

1

VOTED

Status	Submitted
Workspace	Db2
Components	High Availability and/or Disaster Recovery
Created by	You

Allow updates to HADR Standbys

[See this idea on ideas.ibm.com](#)

Allow for updates to be issued on HADR standbys like Informix allows. Here is a extract from the informix manuals:

You can enable applications connected to secondary servers to update database data. If you enable write operations on a secondary server, DELETE, INSERT, MERGE, and UPDATE operations are propagated to the primary server.

Use the UPDATABLE_SECONDARY configuration parameter to control whether the secondary server can update data and to configure the number of connections that update operations use.

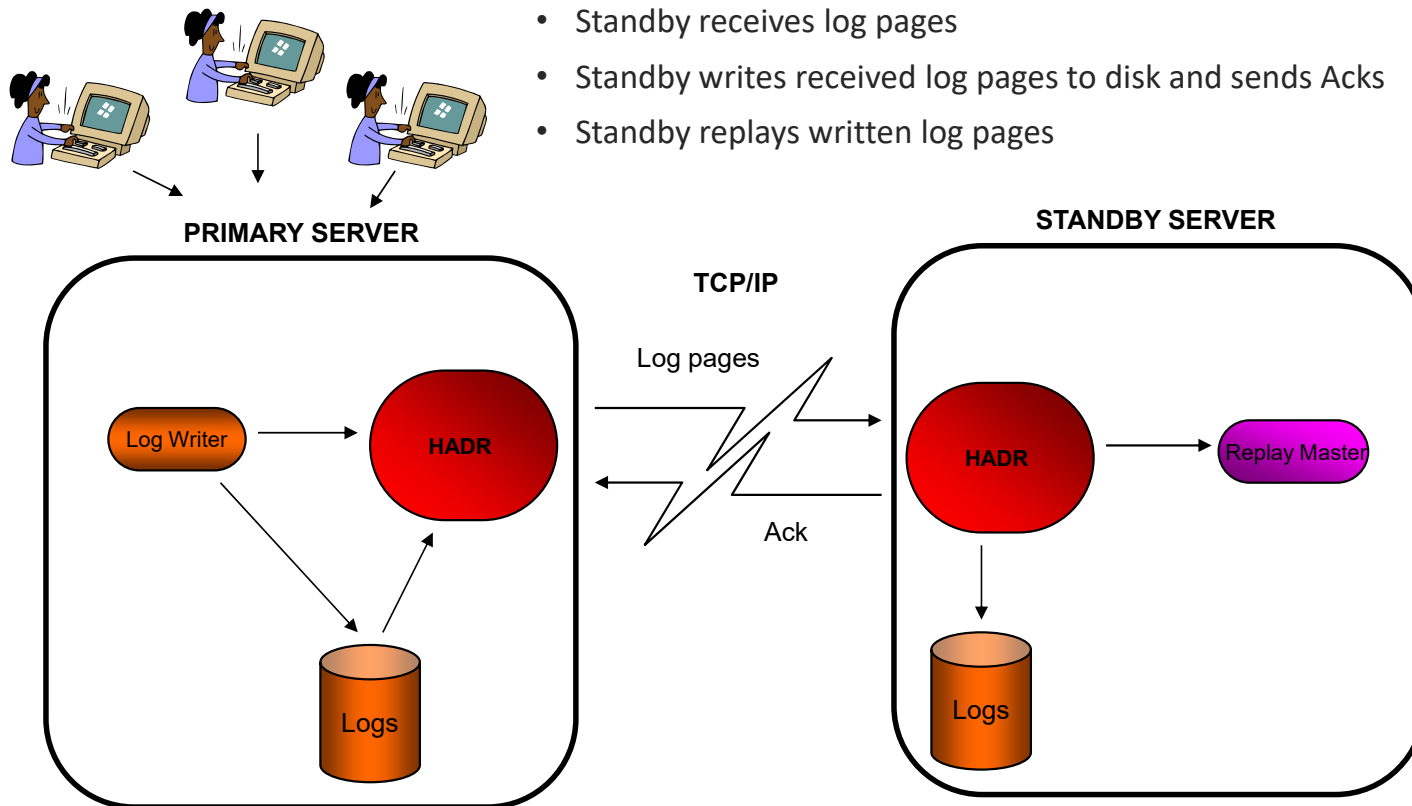
Both data definition language (DDL) statements and data manipulation language (DML) statements are supported on secondary servers.

Agenda

- What is HADR
- How to configure HADR for optimal performance
- Monitoring options
- Best practices

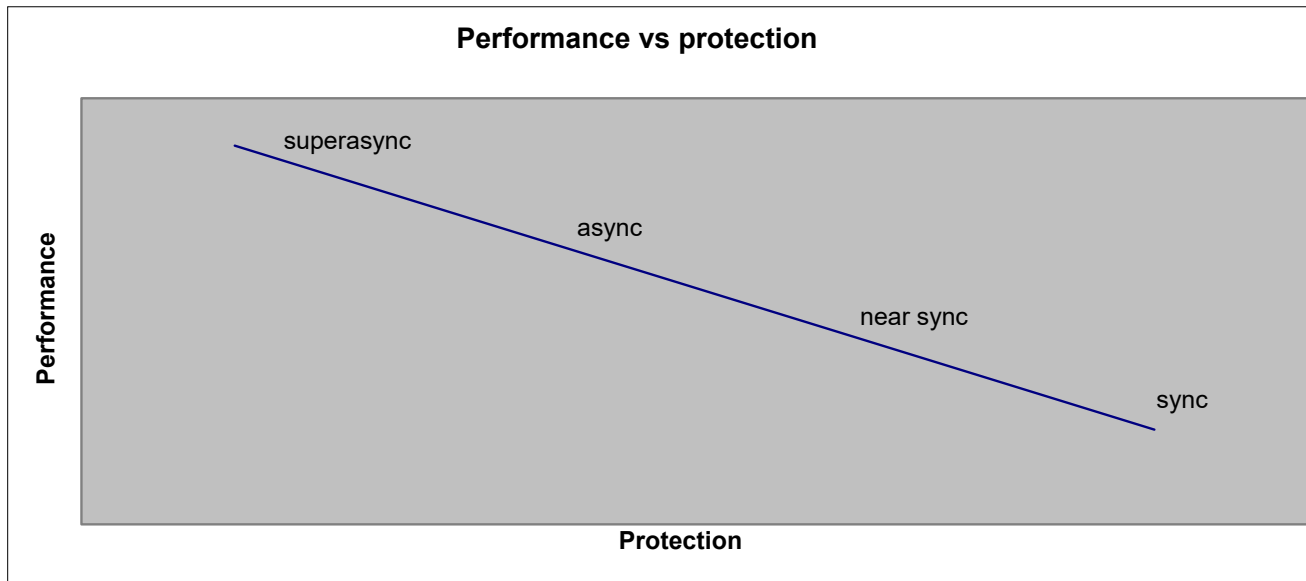
HADR replicate changes from the primary to the standby

- Transactions generate log records on the primary
- Primary sends log pages to the standby
- Standby receives log pages
- Standby writes received log pages to disk and sends Acks
- Standby replays written log pages



Delay in the operations on the critical path can impact transactions on the primary

Performance Overhead varies with sync mode

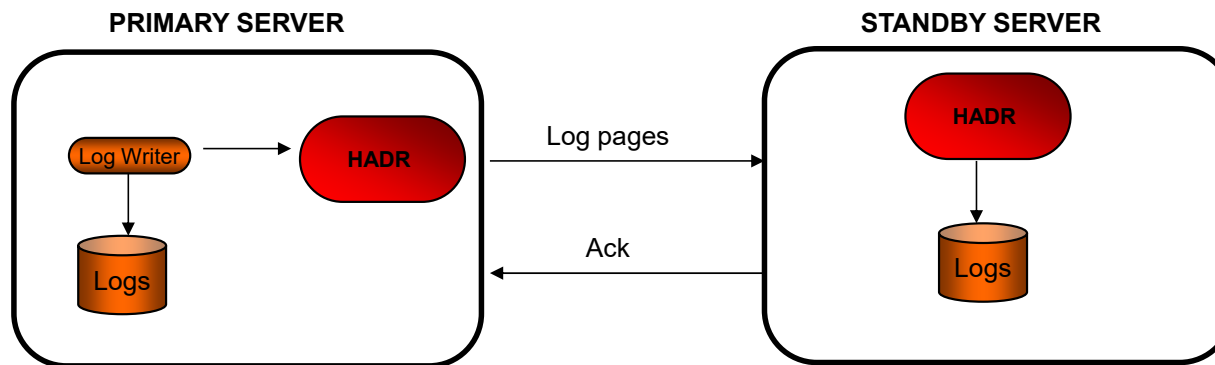


- HADR overhead on primary database logging varies depending on the synchronization mode
 - Stronger sync mode provides more HA and DR protection
 - Weaker sync mode has less impact on the primary database

Synchronization Modes

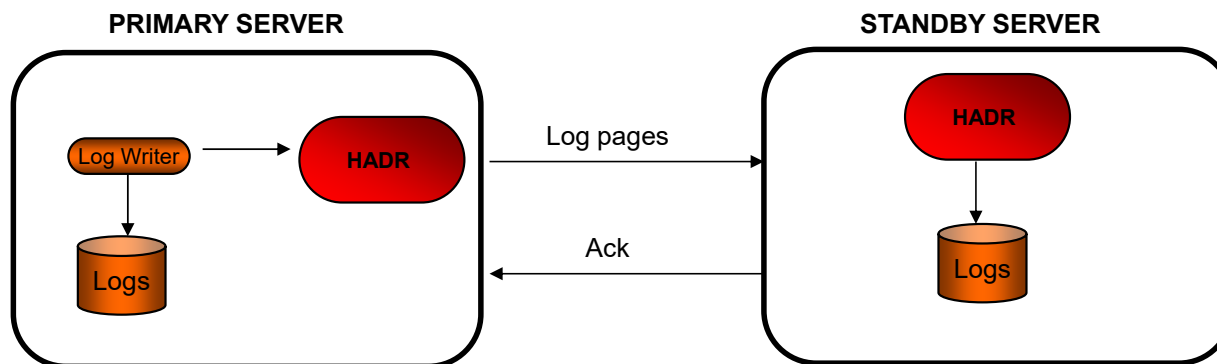
- SYNC mode:

- Logs are first written to the primary and are only then sent to standby (Serially)
- Two on-disk copies of the log data are required for transaction commit
- **Total log write time = primary log write + log send + standby log write + ack message**
- Best data protection
- But the cost of replication is higher than all other modes



Synchronization Modes

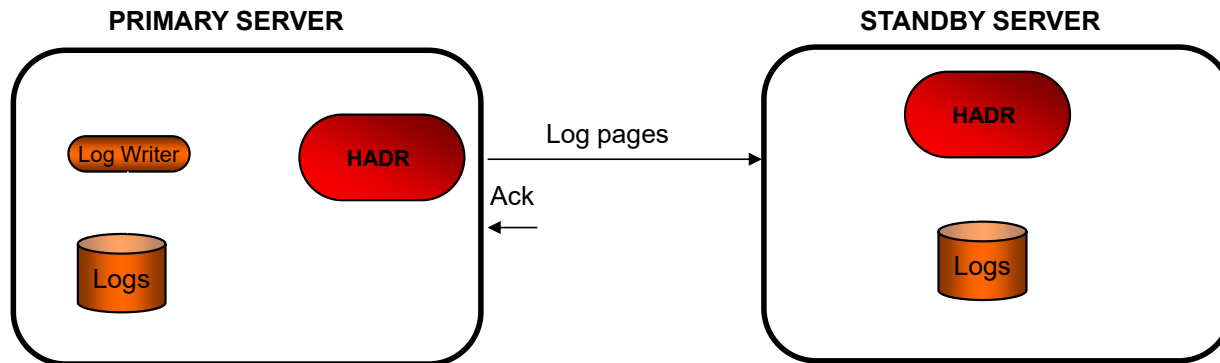
- NEARSYNC mode:
 - Writing logs on the primary and sending logs to standby are done in parallel
 - Standby sends ack message as soon as it receives the logs
 - On a fast network, log replication results in little or no overhead to primary
 - **Total log write time = Max (primary log write, log send + ack message)**
 - Exposure to the relatively rare 'double failure' scenario
 - primary fails and the standby fails before it has a chance to write received logs to disk
 - Good choice for many applications
 - providing near synch protection at lower performance cost



Synchronization Modes

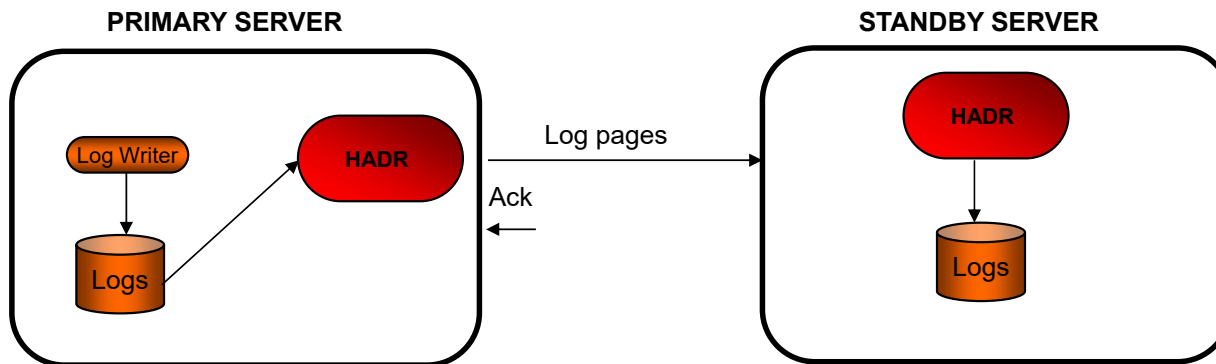
- ASYNC mode

- Writing logs on the primary and sending logs to standby are done in parallel
- Does not wait for ack messages from the standby
 - Just the ack that the message has been sent
- If the primary database fails, there is a higher chance that logs in transit are lost
- **Total log write time = Max (Primary log write rate, Submit log for sending)**
- Well suited for WAN application since network transmission delay does not impact performance



Synchronization Modes

- SUPERASYNC mode
 - Log writing and log shipping are completely independent
 - HADR remains in remote catchup state and never enters peer state
 - Zero impact on Log writing: **Total log write time = Primary log write**
 - But the log gap between the primary and the standby can grow
 - In a failover, data in the gap will be lost.
 - This mode has the least impact on primary, at the cost of the lowest data protection



Which Sync Mode to Use?

- Use SYNC or NEARSYNC mode:
 - Intended for HA
 - On a faster network
 - When you need the highest protection
- Use ASYNC or SUPERASYNC :
 - Intended for DR
 - On a slower network

Network Tuning

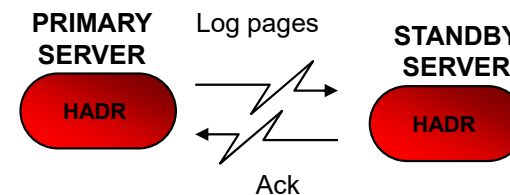
- **TCP performance is critical for HADR performance**
 - Slow TCP performance can slow down HADR log shipping
 - Slow log shipping slows the DB2 logger
 - Slow logger impacts transactions throughput
- A properly configured network is a happy network!
- **The TCP socket buffer size can be set in one of two way:**
 1. At the operating system level
 - the settings is applicable across all TCP connections on the server
 2. At the HADR level
 - Using the DB2 registry variables: **DB2_HADR_SOSNDBUF** and **DB2_HADR_SORCVBUF**
 - Allows tuning TCP window size for HADR connection without impacting other TCP connections
 - First available in V8fp17, V91fp5 and V95fp2
- **Best Practice:**
 - Use the same value for **DB2_HADR_SOSNDBUF** and **DB2_HADR_SORCVBUF**
 - Use the same value on the standby and the primary
 - Use a dedicate NIC card for HADR traffic



Optimal Send/Receive Buffer Size

DB2_HADR_SOSNDBUF / DB2_HADR_SORCVBUF

- Sending an HADR message on TCP:
 - Sender sends data onto the network
 - Data to arrives at the receiver
 - Receiver sends back an Ack message
 - Ack message arrives at the original sender
 - Sender releases buffer holding the Data



- While HADR is waiting for the Ack message, it needs to keep sending more data
- BEST PRACTICES:
 - The send buffer size should be $\text{send_rate} * \text{round_trip_time}$
 - Send rate is the amount of data that is sent over the network
 - Assume max bandwidth utilization
 - Or find the exact bandwidth utilization using the HADR Simulator
 - Round trip time is the amount of time required for sending a message and receive an acknowledgement
 - Assume maximal send rate and worst round trip time



Using a larger send buffer to mask a network hiccup

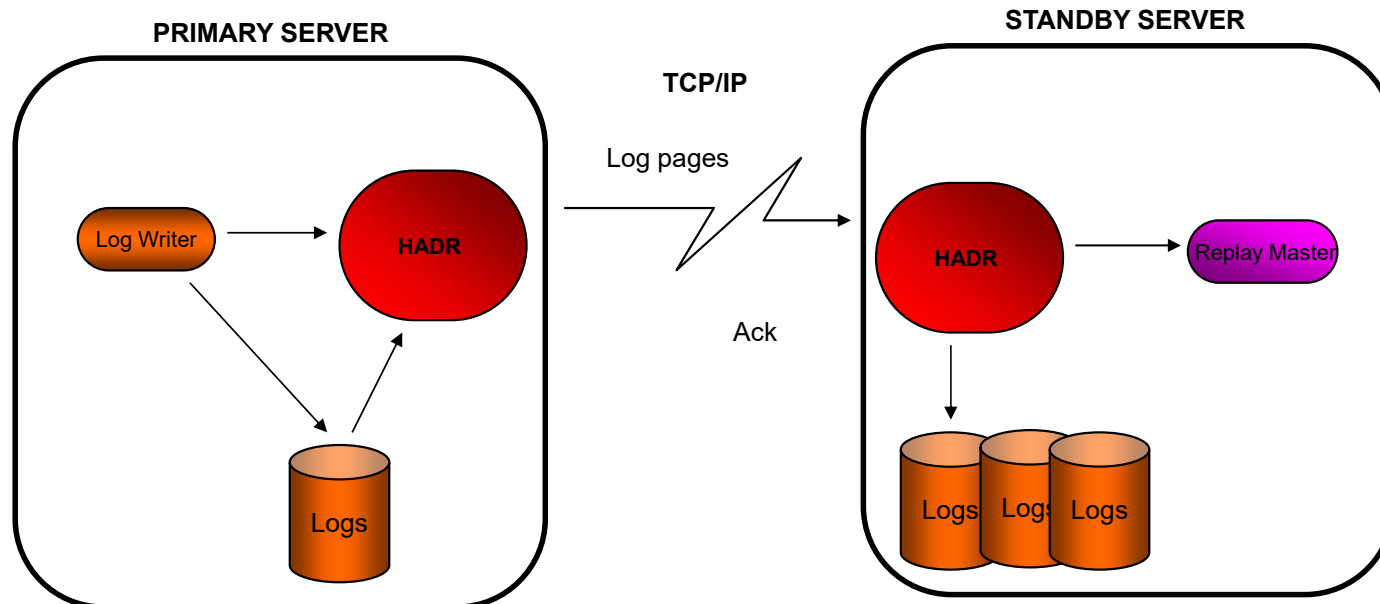
- For ASYNC mode, when there is intermittent network hiccup where the primary is unable to send out log data, a larger send buffer may help
- For example:
 - Assuming a work load generating 3MB/sec of logs
 - Set the sending buffer to 30MB, rather than 3MB
 - This will buffer 10 seconds of work load
 - A hiccup shorter than 10 seconds will not block primary logging
- This setup does not help SYNC and NEARSYNC mode
 - Primary is blocked until the standby receives and acknowledge
- The drawback:
 - During a crash, any data in the buffer which has not been sent, will be lost
 - A larger buffer increases the amount of potential data loss in a failure

Log flushing and send buffer size

- In peer mode, flushing a log page to disk triggers log shipping to the standby
- In remote catchup state, HADR reads log pages from disk and sends to the standby using 64KB packets
- The socket buffer size should there be
 - In Sync, Nearsync, Async: Max (64KB, log writer flush size)
 - In Supersync: at least 64KB
- You can get the log flush size using the db2 Log Scanner and the HADR Calculator
 - scan log files using DB2 Log Scanner
 - run the scanner output through the HADR Calculator

HADR Log Spooling

- All log records are written to the active log path on the standby
- Synchronization Mode is with respect to the log data shipped not with respect to the replay
- Size of spool controlled by the HADR_SPOOL_LIMIT configuration parameter
 - In v10.5+ defaults to AUTOMATIC which is (LOGPRIMARY + LOGSECOND) log files, computed at HADR startup



HADR Log Spooling

- Monitoring the spool usage via the database monitor
 - The `STANDBY_SPOOL_PERCENT` monitoring field returns percent of spool space used
- Performance benefit:
 - Log spooling will absorb load spikes in logging from the primary
 - Primary will no longer be affected by standby replay performance
- Potential draw backs are:
 - Takeover may take longer since the spool must be processed
 - Requires more disk space

Logger Performance

- Log records contain changes to the data in the database and are used during:
 - Rollback – reverse changes made by a statement/transaction
 - Crash recovery – redo/undo work to make database consistent
 - Rollforward – apply changes after a restore is performed
 - HADR – keep the standby in sync with the primary
 - Replication – reconstitute the SQL statements
- Write ahead logging - Log records must be flushed before affected pages are written to disk, to ensure that changes can be undone in the case of a crash
- Log records are written into log files
 - First two pages of each log file are reserved for metadata
 - Remaining pages are for data
 - Number of pages (4096 bytes per page) in a log file is $(\text{LOGFILSIZ} + 2)$

Logger Performance con't

- Multiple agents generate log records for different transactions concurrently into a single log stream
 - The buffer size is set using the database configuration parameter **LOGBUFSZ**
 - Monitor the **NUM_LOG_BUFFER_FULL** monitor element to see if you ever encounter a full buffer
- Writing to the log files is managed by the db2loggw thread
 - threads can be listed via "**db2pd -edus**" command.
 - The logger writes the log records to disk in pages (1 Page = 4096 bytes)
- Log writing can be a bottleneck in high volume systems, especially OLTP systems
 - High performance devices are recommended for logging
- Agents write log records to the log buffer while concurrently the logger thread write pages to disk
 - During commit, an agent waits for the logger to flush the its log record to disk before it can continue
 - With HADR peer mode, the logger also initiate shipping the pages to the standby
 - If the logger is slow, an agent may block waiting for the logger
 - This is why logger performance is critical - Especially in OLTP systems

Self Tuning Memory Manager Best Practices

- Self Tuning Memory Manager (STMM) only runs on the primary
- After a standby turns into a primary via takeover, the STMM thread may not start until the first client connects
- To speed log replay and reads on the standby manually configure the standby using values set by STMM on the primary
- All the changes made by STMM are logged in two places:
 1. db2diag.log – viewable via the db2diag tool
 2. STMM log files – viewable via parseStmmLogFile.pl tool

HADR Timeout

- While connected, the Primary and Standby exchange heartbeat messages
- The user can configure a timeout value using the database configuration parameter **HADR_TIMEOUT**
- If no message is received for the duration of HADR_TIMEOUT seconds, the TCP connection will be closed
 - A standby will then attempt to re-establish the connection by sending a handshake message to the primary
 - A primary will send a redirection message to the standby to probe it to start the handshake protocol
- Heartbeat interval is the minimum of the following:
 - 1/4 of HADR_TIMEOUT
 - 1/4 of HADR_PEER_WINDOW
 - 30 seconds
 - Find the exact heartbeat interval using the monitor element **HEARTBEAT_INTERVAL**

HADR_PEER_WINDOW

- Required when automating HADR Takeover
- The `hadr_peer_window` configuration parameter determines whether the database goes into disconnected peer state after the connection is lost, and how long the database should remain in that state.
- HADR will break the connection as soon as a network error is detected during send, receive, or poll on the TCP socket. HADR polls the socket every 100 milliseconds.
- This allows it to respond quickly to network errors detected by the OS. Only in the worst case, HADR will wait until timeout to break a bad connection.
- In this case, a database application that is running at the time of failure can be blocked for a period of time equal to the sum of the `hadr_timeout` and `hadr_peer_window` database configuration parameters

HADR Recommendations

- Enabling HADR on an existing OLTP database could result in poor performance
 - Ensure you test the network response
- While HADR does provide higher resiliency it does come at a cost of performance
 - Depending on the sync mode used up to 30% overhead could be added
- Avoid the use of the LOAD utility
 - Use the INGEST utility wherever possible as a replacement for LOAD

Agenda

- What is HADR
- How to configure HADR for optimal performance
- **Monitoring options**
- Best practices

Monitoring HADR Performance

- Two interfaces:
 - **MON_GET_HADR** table function
 - Available on the primary
 - Available on the standby when reads on standby is enabled
 - **db2pd -hadr**
 - Available on both the primary and the standby
- The older snapshot interface has been deprecated
- Information for a remote database can be slightly out of date
 - Standby DBs only have information about themselves
 - Primary DB has information on themselves and ALL standbys, but the data could be stale
 - Query each individual DB for the most accurate status

db2pd Example

- By default , **db2pd -hadr** will return streams being processed by the local member.
- On primary: it will return the stream owned by the member and all streams being assisted by the member
- On standby: all streams will be returned if it is issued on the replay member and no data will be returned if it is issued on a non replay member.

```
Database Member 0 -- Database SAMPLE -- Active -- Up 0 days 00:06:32 -- Date 2025-05-29-01.58.21.013424
HADR_ROLE = PRIMARY
REPLAY_TYPE = PHYSICAL
HADR_SYNCMODE = NEARSYNC
STANDBY_ID = 1
LOG_STREAM_ID = 0
HADR_STATE = PEER
HADR_FLAGS = TCP_PROTOCOL
PRIMARY_MEMBER_HOST = db2hadrnnode3-priv
PRIMARY_INSTANCE = db2inst1
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = db2hadrnnode4-priv
STANDBY_INSTANCE = db2inst1
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 05/29/2025 01:52:26.679696 (1748501546)
HEARTBEAT_INTERVAL(seconds) = 5
HEARTBEAT_MISSED = 0
HEARTBEAT_EXPECTED = 71
HADR_TIMEOUT(seconds) = 120
TIME SINCE LAST RECV(seconds) = 5
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.000161
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.024
LOG_HADR_WAIT_COUNT = 148
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 87040
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 131072
PRIMARY_LOG_FILE,PAGE,POS = S0000000.LOG, 117, 61620033
STANDBY_LOG_FILE,PAGE,POS = S0000000.LOG, 117, 61620033
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000000.LOG, 117, 61620033
STANDBY_RECV_REPLAY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 05/29/2025 01:58:10.000000 (1748501890)
STANDBY_LOG_TIME = 05/29/2025 01:58:10.000000 (1748501890)
STANDBY_REPLAY_LOG_TIME = 05/29/2025 01:58:10.000000 (1748501890)
STANDBY_RECV_BUF_SIZE(pages) = 512
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 13000
STANDBY_SPOOL_PERCENT = 0
STANDBY_ERROR_TIME = NULL
PEER_WINDOW(seconds) = 120
PEER_WINDOW_END = 05/29/2025 02:00:20.000000 (1748502020)
READS_ON_STANDBY_ENABLED = N
HADR_LAST_TAKEOVER_TIME = NULL
```

```
Database Member 0 -- Database SAMPLE -- Standby -- Up 0 days 00:04:13 -- Date 2025-05-29-01.58.21.013424
HADR_ROLE = STANDBY
REPLAY_TYPE = PHYSICAL
HADR_SYNCMODE = NEARSYNC
STANDBY_ID = 0
LOG_STREAM_ID = 0
HADR_STATE = PEER
HADR_FLAGS = TCP_PROTOCOL
PRIMARY_MEMBER_HOST = db2hadrnnode3-priv
PRIMARY_INSTANCE = db2inst1
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = db2hadrnnode4-priv
STANDBY_INSTANCE = db2inst1
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 05/29/2025 01:48:18.384070 (1748501298)
HEARTBEAT_INTERVAL(seconds) = 5
HEARTBEAT_MISSED = 0
HEARTBEAT_EXPECTED = 37
HADR_TIMEOUT(seconds) = 120
TIME SINCE LAST RECV(seconds) = 4
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.000166
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.024
LOG_HADR_WAIT_COUNT = 144
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 87040
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 131072
PRIMARY_LOG_FILE,PAGE,POS = S0000000.LOG, 108, 61583863
STANDBY_LOG_FILE,PAGE,POS = S0000000.LOG, 108, 61583863
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000000.LOG, 108, 61583863
STANDBY_RECV_REPLAY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 05/29/2025 01:53:29.000000 (1748501609)
STANDBY_LOG_TIME = 05/29/2025 01:53:29.000000 (1748501609)
STANDBY_REPLAY_LOG_TIME = 05/29/2025 01:53:29.000000 (1748501609)
STANDBY_RECV_BUF_SIZE(pages) = 512
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 13000
STANDBY_SPOOL_PERCENT = 0
STANDBY_ERROR_TIME = NULL
PEER_WINDOW(seconds) = 120
PEER_WINDOW_END = 05/29/2025 01:57:29.000000 (1748501849)
READS_ON_STANDBY_ENABLED = N
HADR_LAST_TAKEOVER_TIME = NULL
```

Worth Paying Attention To

- **HADR_STATE**
 - Verify the primary and standby are in connected and in PEER mode (or REMOTE_CATCHUP for superasync sync mode)
- **HADR_CONNECT_STATUS**
 - SQLM_HADR_CONN_CONNECTED - the good
 - SQLM_HADR_CONN_DISCONNECTED - the bad
 - SQLM_HADR_CONN_CONGESTED - and the ugly
 - If you see congestion, either there are network issues, or the standby is not keeping up.
 - Use Spooling if the standby is not keeping up
- **LOG_HADR_WAIT_CUR**
 - Provides current logger waiting time on an HADR log shipping request
 - Helps identify a primary that is being blocked by log shipping to the standby
 - Should never reach more than 5 seconds
 - Can serve as an early indication of network or standby problems
 - You can cap this using the registry variable **DB2_HADR_PEER_WAIT_LIMIT**
- **HADR_LOG_GAP**
 - Use this element to determine the gap between the primary and standby HADR database logs
 - It should be kept to a minimum
 - An increasing number indicates a standby that is not keeping up
- **HEARTBEAT_EXPECTED vs HEARTBEAT_MISSED**
 - Number of heartbeat messages expected vs those that were not received, accumulated since database startup on the local member

Monitoring HADR Standby elements

STANDBY_RECV_BUF_PERCENT

- receive buffer full (100% used) will cause standby log receive to be blocked and primary log writing and transactions will eventually be blocked. As replay progresses, part of the buffer will be released and log receive will resume. If spooling is enabled, 100% buffer use does not indicate a bad condition. HADR will release the buffer for new incoming data if it needs to, even if log data in the buffer has not been replayed.

STANDBY_RECV_BLOCKED

- flag from the HADR_FLAGS field directly indicates that the standby log receiving is blocked. Primary log send and transactions will eventually be blocked if the condition persists. There are multiple scenarios of this condition:
 - When log spooling is disabled (or not supported), standby receive buffer is full (STANDBY_RECV_BUF_PERCENT is 100%).
 - When log spooling is enabled, spooling has reached configured spool limit (STANDBY_SPOOL_PERCENT is 100%).
 - The standby logging device is full (STANDBY_LOG_DEVICE_FULL flag from HADR_FLAGS field is set), regardless of whether spooling is enabled or not.

STANDBY_LOG_DEVICE_FULL

- Flag from HADR_FLAGS reports that standby log device is full. The STANDBY_RECV_BLOCKED flag will also be turned on when log device is full. The device full flag allows you to identify the cause of the blocked receiving.

Db2mon – shipped in all current Db2 Editions

All SQL and shell scripts are under

`~/sqllib/samples/perf`

For most tasks, collect activity for 30 seconds

`./db2mon.sh 30`

For a busy database, we recommend up to 5 minutes collection time only



Data capture and reporting

- Db2mon (db2mon.sh) performs the following processing
 - configure a separate bufferpool / tablespace to limit impact on running system
 - capture data at start - record in-flight statements
 - capture data at end - record in-flight statements
 - calculate differences between data captures
 - generate report
 - remove separate bufferpool / tablespace

Prereqs

- monitoring must be enabled at the database level with the following database configuration parameters: MON_ACT_METRICS must be set at least to BASE, which is the default value.
- MON_REQ_METRICS must be set at least to BASE. Its default value is EXTENDED, which gives full monitor information on tables and indexes, and is an ideal setting for db2mon.

What about the rest of the scripts?

- Db2mon can be run in a number of different ways
 - Full report mode for a time period → db2mon.sh
 - both data collection and analysis are done on the original host
 - Offline mode → db2mon_export.sql
 - data collection is done on the production system
 - all data is exported to IXF files → copy and analyze elsewhere
 - db2mon_import.sql and db2mon_report.pl

FAQ

Can I run db2mon from a remote client?

if you can connect to the database using CLP, then **Yes**

either of:

CATALOG TCPIP NODE, etc.

db2dsdriver.cfg

works for pureScale and MPP clusters too!

collects data on all hosts in cluster

How much performance impact does db2mon have?

collection is lightweight – typically not noticeable

analysis is slightly more intensive

if there are performance concerns (i.e. existing server is running near CPU capacity) then use offline report generation

What if I want to monitor a specific task?

- Collecting data for N seconds will only capture activities that complete within the capture period
 - ideal for monitoring activity of a busy database
 - tricky to capture one specific task or statement
- Solution: sandwich your task between the “Before” and “After” sections
 - use the db2mon.sh script as a guide

```
scriptRoot=$HOME/sqllib/samples/perf  
export DB2OPTIONS="+c -tvf"  
db2 $scriptRoot/db2monBefore.sql  
db2 $scriptRoot/yourScript.sql  
db2 $scriptRoot/db2monAfter.sql
```


Finding what you need in the db2mon report

- Four sections
 - Monitoring sanity check
 - Start Data capture
 - End Data capture
 - Analysis report
- Search strings for sections
 - Checking db2mon
 - start of capture
 - end of capture
 - Data collected

Start Data Capture

- Look under “REPORT STARTS HERE”
 - CAPTURE_TIME for the first collection
- Three reports
 - START#EXSQL: Currently executing SQL at start of capture (non-zero metrics only)
 - START#LOCKW: Current lock waits at start of capture
 - START#EXUTL: Currently executing utilities at start of capture

Note: all DB2MON statements have this comment



```
/* IBM_DB2MON */ select min(ts)  
capture_time from  
mon_current_sql_plus_start
```

```
CAPTURE_TIME  
-----
```

```
2023-03-29-22.07.58.490334
```

```
1 record(s) selected.
```

End Data Capture

- Three sections:
 - END#EXSQL: Currently executing SQL at end of capture (non-zero metrics only)
 - END#LOCKW: Current lock waits at end of capture
 - END#EXUTL: Currently executing utilities at end of capture
- Note: the two capture times are more than 30 seconds apart

Note: all DB2MON statements have this comment

```
/* IBM_DB2MON */ select min(ts)  
capture_time from  
mon_current_sql_plus_end
```

```
CAPTURE_TIME
```

```
-----  
2023-03-29-22.08.30.296941
```

```
1 record(s) selected.
```

Analysis Report

- Labelled by “scope” with “TAG#”:
 - INS# - instance
 - DB# - database
 - CFG# - configuration
 - TSP# - tablespace
 - BPL# - bufferpool
 - PAG# - page
 - LTC# - latch
 - BLU# - column-organized tables
 - TBL# - table
 - IDX# - index
 - CON# - connection
 - WLB# - workload balancing
 - PKG# - package
 - SQL# - SQL statements
 - CF# - pureScale cluster caching facility

Queries, queries everywhere

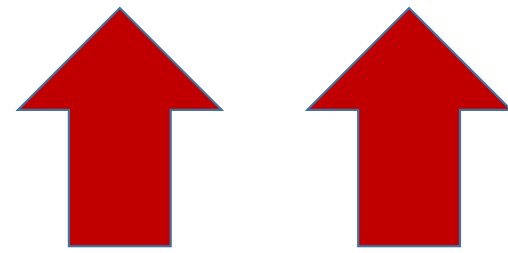
- There are 77 separate queries
 - 71 in the Analysis report
- Top down investigations start from
 - Database
 - Bufferpool
 - Indexes
 - SQL

BLU	1
BPL	9
CF	12
CFG	3
CON	3
DB	16
IDX	4
INF	1
INS	1
LTC	1
PAG	2
PKG	1
SQL	7
TBL	2
TSP	7
WLB	1

DB Log Write Times

```
=====
DB#LOGWR: Database log write times
=====

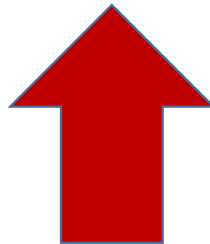
select member, num_log_write_io, case when ts_delta > 0 then decimal( double(num_log_write_io) / ts_delta, 10, 4 ) else null end as log_write_io_per_s,
MEMBER NUM_LOG_WRITE_IO LOG_WRITE_IO_PER_S LOG_WRITE_MB_PER_S LOG_WRITE_TIME LOG_WRITE_TIME_PER_IO_MS NUM_LOG_BUFFER_FULL
-----
0 23918 79.1986 31.9304 40170 1.6794 16668
1 record(s) selected.
```



DB Log Read Times

```
=====
DB#LOGRE: Database log read times
=====

select member, integer(num_log_read_io) num_log_read_io, case when ts_delta > 0 then decimal( double(num_log_read_io) / ts_delta, 10, 4 ) else null end as log_read_io_per_s,
MEMBER NUM_LOG_READ_IO LOG_READ_IO_PER_S LOG_READS LOG_READ_TIME LOG_READ_TIME_PER_IO_MS NUM_LOG_DATA_IN_BUFFER CUR_COM_LOG_BUFF_LOG_READS CUR_COM_DISK_LOG_READS
-----
0 15 0.0496 15 12 0.8000 0 -2 15
1 record(s) selected.
```



Other HADR Log Stats

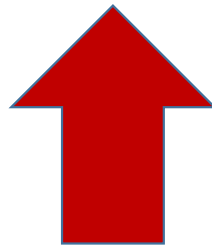
```
=====
DB#LOGST: Other database log statistics
=====
```

```
select member, num_log_write_io, num_log_part_page_io, case when num_log_part_page_io > 0 then decimal( double(num_log_part_page_io) / num_log_write_io, 10, 4 ) else null end as lo
```

```
MEMBER NUM_LOG_WRITE_IO      NUM_LOG_PART_PAGE_IO LOG_PART_PAGE_RATIO AVG_LOG_HADR_WAIT_TIME
```

```
-----
| 0                23918                128                0.0053                10.8084
```

```
1 record(s) selected.
```



Check Sync disk Writes

TSP#DSKIOSYNC: Disk read and write I/O times (synchronous)

MEMBER	TBSP_NAME	NUM_READS	AVG_SYNC_READ_TIME	NUM_WRITES	AVG_SYNC_WRITE_TIME
0	TS_YFS_REL	52975	1.03	0	-
0	TS_YSR_ORL	52072	0.92	0	-
0	TS_YFS_COMD	33637	0.74	0	-
0	TS_YSR_DH	27083	0.83	0	-
0	TS_YFS_OLS	22000	0.78	0	-
0	TS_YFS_COMG	11585	0.73	0	-
0	TS_YFS_IX_REL	9787	1.03	0	-
0	TS_YFS_IX_SHP	8138	0.91	0	-
0	TS_YFS_IX_COMD	6834	0.74	0	-
0	TS_INV_DAT	6603	0.73	0	-
0	TS_YSR_IX_ORL	6213	0.96	0	-
0	TS_ORH_IDX	3357	0.84	0	-
0	TS_YFS_COMF	3012	0.73	0	-
0	TS_INV_IDX	1815	0.82	0	-
0	TS_YSH_IDX	1493	0.87	0	-
0	TS_YFS_IX_COMG	1252	0.81	0	-
0	TS_ORH_DAT	913	0.73	0	-
0	TS_YSR_IX_DH	644	1.08	0	-
0	TS_YSR_IDX	478	0.74	0	-
0	TS_YFS_IX_COMF	418	0.92	0	-
0	TS_YFS_IX_OLS	402	0.83	0	-
0	TS_YSR_DAT	79	0.68	0	-
0	TS_YSR_IX_AT	79	0.45	0	-
0	TS_OAL_LOB	1	2.00	76	1.17
0	DB2MONTMPTBSP	0	-	15	0.06
0	TS_YSH_DAT	11	0.81	0	-
0	YFS_AUDIT_IDX	10	0.90	0	-
0	TS_YFS_IX_COME	8	0.87	0	-
0	TS_YFS_SHP	8	1.37	0	-
0	YFS_AUDIT_DAT	6	0.66	0	-
0	TS_INB_IDX	3	1.00	0	-
0	YFS_HIST3_IDX	3	1.00	0	-
0	TS_ALT_IDX	2	1.00	0	-
0	TS_YSR_IX_LV	2	1.00	0	-
0	SYSCATSPACE	1	1.00	0	-

Check Bufferpool Logical vs Physical Reads

BPL#STATS: Bufferpool statistics by tablespace

```

=====
BPL#STATS: Bufferpool statistics by tablespace
=====
/* IBM_DB2MON */ select member, cast(substr(tbsp_name,1,20) as varchar(20)) as tbsp_name, pool_data_l_reads + p
MEMBER  TBSP_NAME          DATA_L_READS      DATA_P_READS      SYNC_DATA_P_READS  DATA_HR  DATA_LBP_HR (
-----  -----
0 PENSIONDATA_16K      993730415          2522371            608645             99.93     99.93
0 PENSIONIDX_16K       28                 11                 11                 -         -
0 CORPIDX_16K          88                 0                  0                  -         -
0 CORPTRDI_16K         0                  0                  0                  -         -
0 CORPDATA_16K        74409910           339007             56968              99.92     99.92
0 ACGDBDATA_16K       1200137            992024             9477                99.21     99.21
0 TDAIDX_16K           0                  0                  0                  -         -
0 TDAPARTDATA36_16K   72064              12095              11722               83.73     83.73
0 TDAPARTIDX25_16K    0                  0                  0                  -         -
0 TDAPARTDATA16_16K   119609             12096              10981               90.81     90.81
0 AEGONDATA_16K       4070092            12106              2580                99.93     99.93
0 TDAPARTDATA14_16K   71427              7852               7503                89.49     89.49
0 TDADATA_16K         89951056           10023              6560                99.99     99.99
0 TDAPARTIDX1_16K     0                  0                  0                  -         -
0 TDAPARTDATA4_16K    75080              6765               5655                92.46     92.46
0 TDAPARTIDX24_16K    0                  0                  0                  -         -
0 TDAPARTDATA6_16K   71938              6345               5837                91.88     91.88
0 TDAPARTDATA15_16K  56833              5172               4874                91.42     91.42
0 TDAPARTIDX2_16K     0                  0                  0                  -         -
0 TDAPARTDATA35_16K  13047              3895               3032                76.76     76.76
0 TDAPARTIDX19_16K   0                  0                  0                  -         -
0 TDAPARTIDX17_16K   0                  0                  0                  -         -
0 TDAPARTDATA34_16K  126196             2860               2736                97.83     97.83
0 TDAPARTIDX26_16K   0                  0                  0                  -         -
0 TDAPARTIDX15_16K   0                  0                  0                  -         -
0 TDAPARTIDX14_16K   0                  0                  0                  -         -
0 TDAPARTIDX22_16K   0                  0                  0                  -         -
0 TDAPARTIDX23_16K   0                  0                  0                  -         -
0 TDAPARTIDX4_16K    0                  0                  0                  -         -
0 TDAPARTIDX18_16K   0                  0                  0                  -         -
0 TDAPARTIDX5_16K    0                  0                  0                  -         -
0 TDAPARTDATA2_16K   18026              2208               1933                89.27     89.27
0 TDAPARTIDX21_16K   0                  0                  0                  -         -
0 TDAPARTIDX16_16K   0                  0                  0                  -         -
0 TDAPARTIDX27_16K   0                  0                  0                  -         -
0 TDAPARTIDX7_16K    0                  0                  0                  -         -
0 ACGDBIDX_16K        3                  0                  0                  -         -
0 TDAPARTIDX20_16K   0                  0                  0                  -         -
0 CORPTRDD47_16K     481179             3011               1311                99.72     99.72
0 TDAPARTDATA3_16K   25682              1614               1404                94.53     94.53
0 TDAPARTDATA5_16K   13314              1663               1663                87.50     87.50
0 TDAPARTDATA25_16K  20775              2451               1119                94.61     94.61
0 TDAPARTIDX8_16K    0                  0                  0                  -         -
0 CORPTRDI47_16K     0                  0                  0                  -         -
0 TDAPARTDATA24_16K  127258             2318               146                 99.88     99.88
0 CORPTRDI35_16K     0                  0                  0                  -         -
=====

```

Agenda

- What is HADR
- How to configure HADR for optimal performance
- Monitoring options
- **Best practices**

Most Common HADR Tuning Practices

LOGBUFSIZ

- Controls the amount of memory that DB2 uses to buffer I/O to its recovery log files. This will get flushed to disk when any of the following three considerations are met
 1. A transaction issues a commit
 2. A transaction issues a rollback
 3. The memory buffer is full – worse case scenario for HADR as this will result in multiple round trips between the primary and the standby
- Ensure NUM_LOG_BUFFER_FULL is zero

=====> Have there been any log buffer full conditions?

db2 "Select member, NUM_LOG_BUFFER_FULL from table(mon_get_transaction_log(-2)) with UR"

MEMBER NUM_LOG_BUFFER_FULL

0 430664

In this case increase the LOGBUFSIZ parameter



REORGs

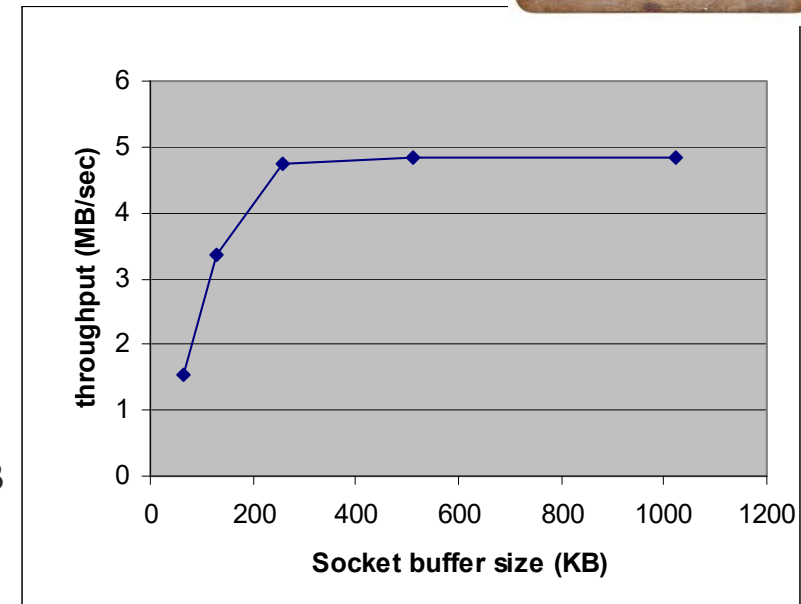
- What is the best REORG?
 - NO REORG!
- Why are you reorging
 - To recluster the data
 - No longer required on SSDs / nVMEs
 - No such thing as shoeshining the disk
 - To reclaim space
 - Consider using Range Partitioned Tables
 - If you detect a lot of overflows use **ONLINE REORG CLEANUP OVERFLOWS**
 - db2pd -tcbstats will show the number of overflow reads/creates. But that is only valid since the db was last activate
 - query the OVERFLOW column in the SYSSTAT.TABLES view
- If you are still convinced you need to reorg consider using ADMIN_MOVE_TABLE stored proc



Using the HADR simulator

- Below is actual result between IBM labs in Portland, Oregon, and Silicon Valley, California
- The physical distance is about 1000km
- Buffer Throughput results:

64KB	1.554048 MBytes/sec
128KB	3.347476 MBytes/sec
256KB	4.734673 MBytes/sec
512KB	4.834047 MBytes/sec
1MB	4.821998 MBytes/sec
- Throughput peaks at 4.8 MB/sec with buffer size at around 256KB
- Set DB2_HADR_SOSNDBUF and DB2_HADR_SORCVBUF to 256K



Best practices for HADR TCP buffer size

- Use HADR simulator to find out TCP requirement
- Test new size with HADR simulator before you apply it to database
- Use a minimum of 64KB
- Monitor actual size to verify that the requested size is indeed being used
- Consider larger size to mask network hiccup if you are using ASYNC mode
- Use the same value for the send and receive buffer sizes
- Use same values on the primary and the standby



Setting HADR Timeout

- If HADR_TIMEOUT is too long, it will slow detection of a lost connection or a failed standby
 - This may end up blocking transactions on primary
- If HADR_TIMEOUT is too short, HADR may get too many false alarms
 - Resulting in breaking the connection more often than necessary
- BEST PRACTICES:
 - The recommended HADR_TIMEOUT is at least 30 seconds
 - The default is 120 seconds
 - Some customers set HADR_TIMEOUT to very low values in order to avoid ever blocking the primary, at the cost of a disconnection every time the network hiccups



Logging Best Practices

- Log buffer should be multiple times the size of the max number of log pages flushed
- Use a dedicated device for the log path
 - Do not share devices between table spaces and log path
- A high-performance device is recommended as log device
 - Log stream writing can easily become a bottleneck, even when HADR is not enabled
 - This is particularly important for OLTP systems
- DB2 Log Scanner can be used to analyze logger behavior



Validate storage devices used

Verify your disks are similar on all HADR nodes

- Ensure you are using the same class of disks on all nodes and identical underlying infrastructure (# of spindles / SSDs)
- Preferably use SSD / NvME disk for transactions logs
 - Do NOT mix SSD/NvME with Spinning Disk for active logs
- HADR Simulator will measure the performance of your disk
 - $\text{write_time} = \text{per_write_overhead} + \text{data_amount} / \text{transfer_rate}$
- Little benefit to having multiple devices for the active logs as there will only be a single thread writing to the active log at any one time.



HADR's Network Adapter usage

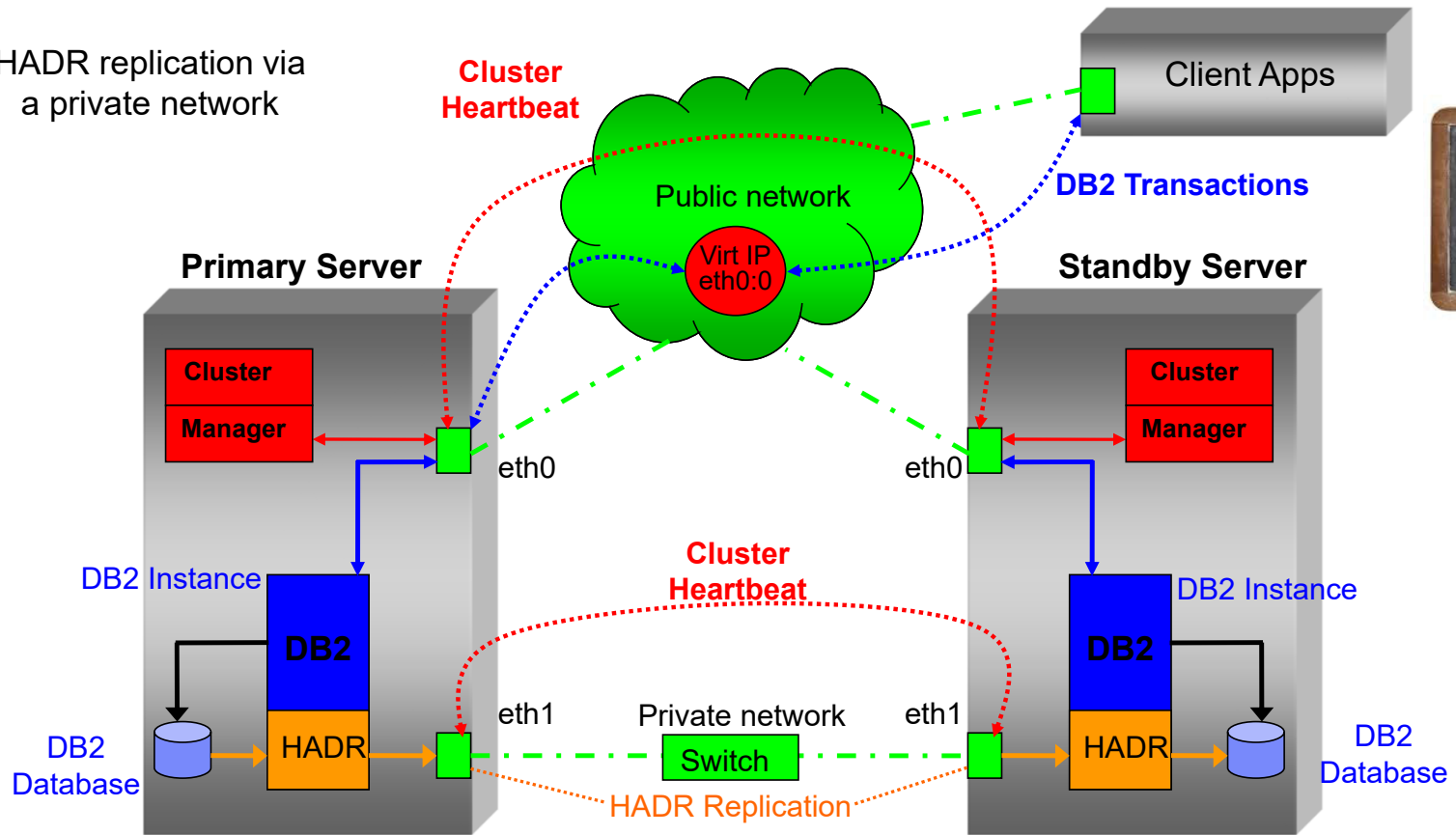
Use a dedicated NIC for HADR traffic

- If possible, use a dedicated private NIC for HADR log transmission between the nodes, including AIX standbys in a remote DC
- This will provide better performance as the NIC card will not get flooded with both user and HADR traffic.



Preferred DB2 HADR environment

HADR replication via a private network



Collect monitoring information

- To gather information for diagnostics, monitor HADR at regular intervals
- The information on the primary pertaining to the standby may be old, always execute db2pd on each node.



- Example shell script:

```
while :
do
  issue "db2pd -hadr" command on primary
  record output

  issue "db2pd -hadr" command on standby
  record output

  sleep 60
done
```

- db2pd is preferred over MON_GET_HADR because
 - it is light weight
 - can run on a standby without reads on standby enabled

Tuning a slow standby

- **Hardware Utilization**

- Check hardware bottleneck on standby using tools like vmstat and iostat
- It is recommended that primary and standby have the same hardware

- **Number of Replay Threads**

- Recovery is done in parallel using multiple worker threads, which defaults to the number of physical CPUs
- When there are a large number of CPUs, the default may be too high

- To check the number of threads used, look for lines like this in db2diag.log:
“Using parallel recovery with 6 agents 4 QSets 20 queues and 0 chunks”

- Tune Log Replay – V 11.5.9+

- `db2set DB2_RECOVERY_SETTINGS=NUM_AGENTS:13;QSETSIZE:8;`

- Tune log replay (prior to V 11.5.9)

- `db2set DB2BPVARS=$HOME/mybpvars.cfg`
- where the contents of files has below settings to speed up the rollforward .
- `PREC_NUM_AGENTS=13`
- `PREC_NUM_QSETSIZE=8`
- Recent tests indicate `PREC_NUM_AGENTS=13 PREC_NUM_QSETSIZE=8` provides the best replay speed (approx. 45 MB/sec)

- **Reads on Standby**

- When reads on standby is enabled, read queries will compete against replay thread for resources
- Experiment with disabling reads on standby and gauge the impact



Recent HADR Enhancements

Db12 V 12.1

- Db2 pureScale HADR support for enterprise-grade end-to-end SSL encryption

Db2 V 12.1.3

- HADR support in pureScale environments with mixed topology between primary and standby clusters

Thank
YOU

The text "Thank YOU" is rendered in a large, bold, sans-serif font. Each letter is filled with a different portrait of a diverse individual. The "T" shows a man in a white shirt and orange tie. The "h" features a woman with dark hair. The "a" contains a young boy with a green face. The "n" shows a woman with dark hair. The "k" depicts a man with glasses. The "Y" features a man in a white lab coat. The "O" shows a young boy with a green face. The "U" contains a woman with dark hair. The portraits are set against a light blue background with a subtle grid pattern.