IDUG

2024 NA Db2 Tech Conference

Db2 HADR Performance Tuning - How to make it fly

Dale McInnis

IBM Canada Ltd.

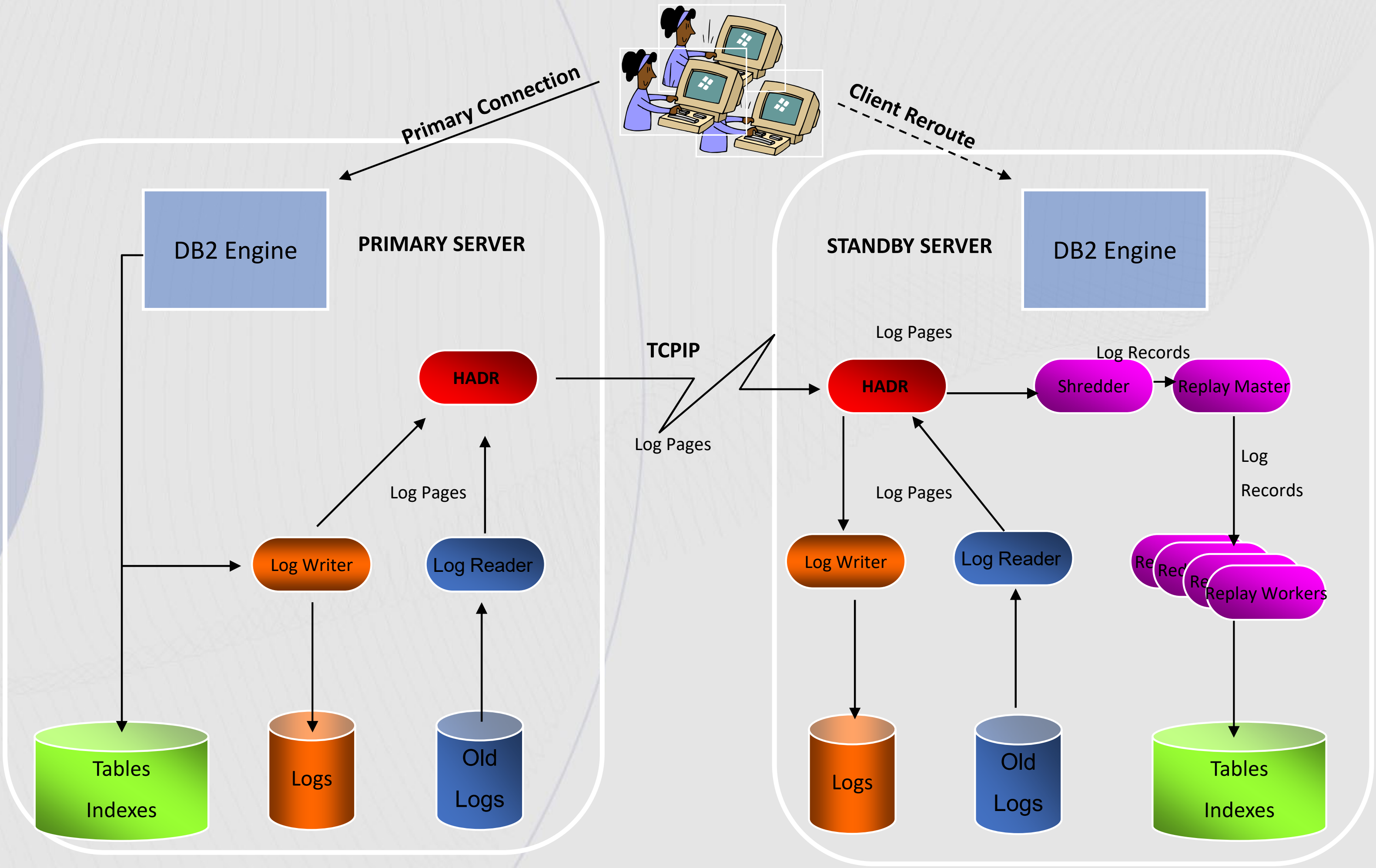
@IDUGDb2
#IDUG_NA24

Session Code: RAS3| Platform: LUW

Agenda

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

HADR Implementation



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

Tools available to measure HADR related performance

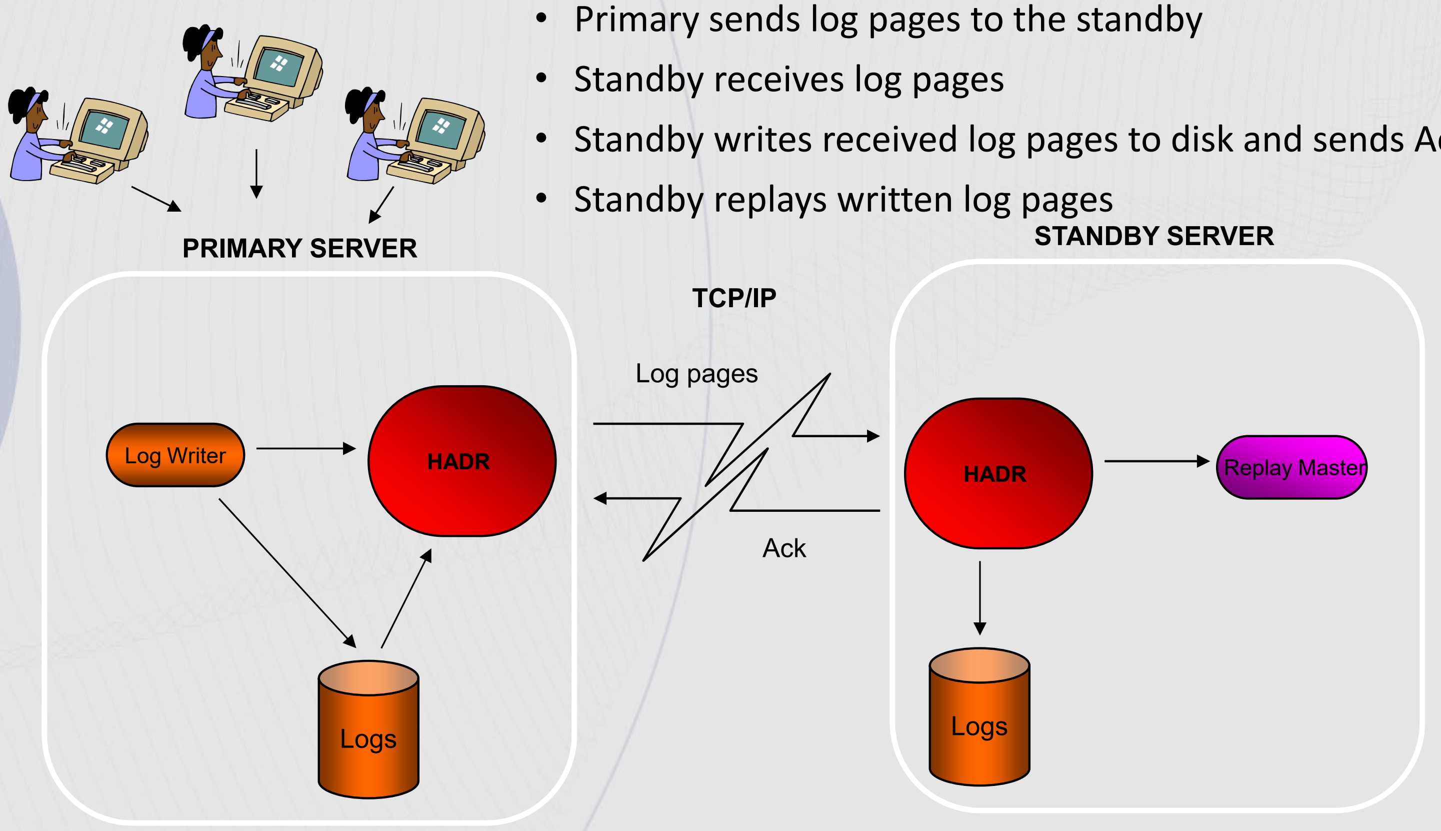
- Check out
  https://ibm.github.io/db2-hadr-wiki/hadrPerf.html
- HADR Simulator to measure network and disk speed
  https://ibm.github.io/db2-hadr-wiki/hadrSimulator.html
- DB2 Log Scanner to analyze database workload
  https://ibm.github.io/db2-hadr-wiki/db2logscan.html
- HADR Calculator to estimate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

Agenda

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

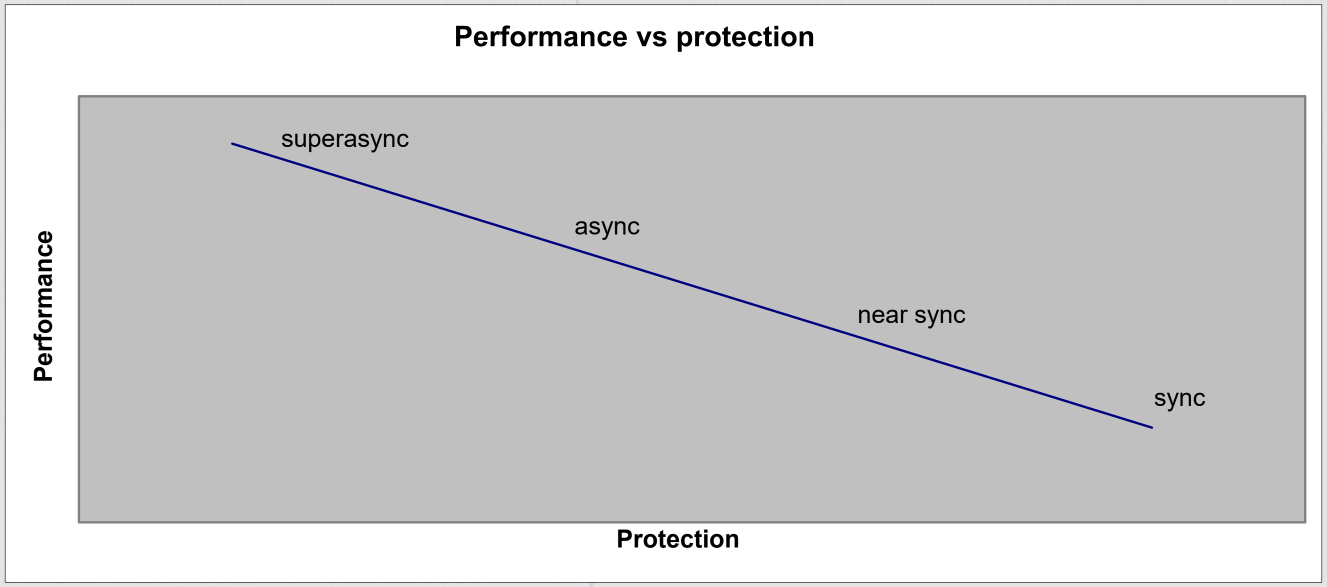
HADR replicate changes from the primary to the standby

- 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

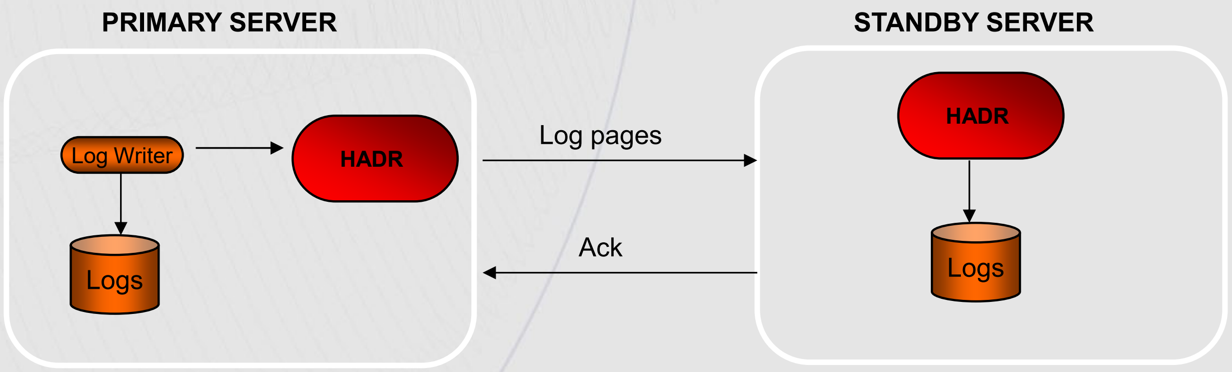
Performance Overhead varies with sync mode



- 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

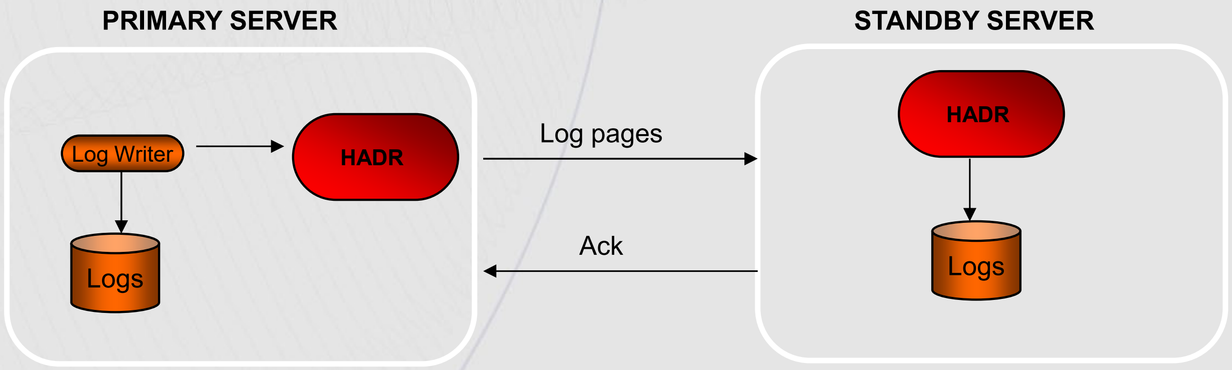
Synchronization Modes

- 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



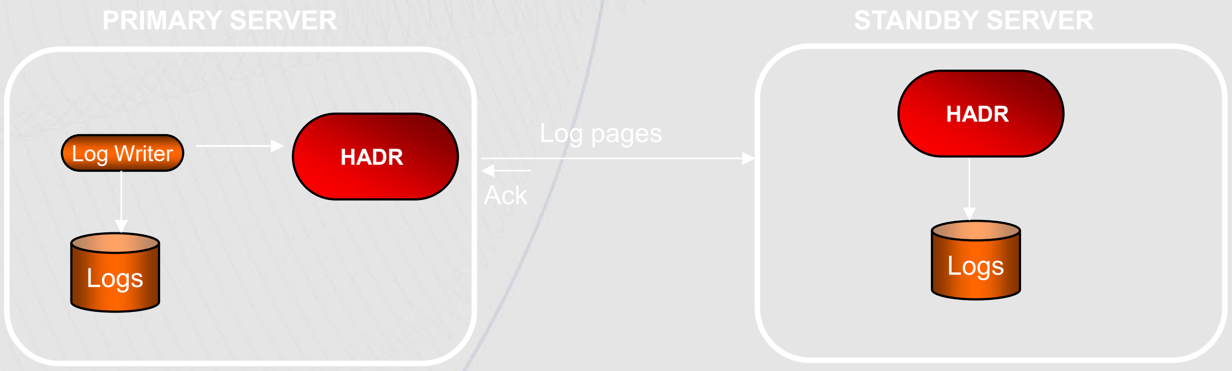
Synchronization Modes

- 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



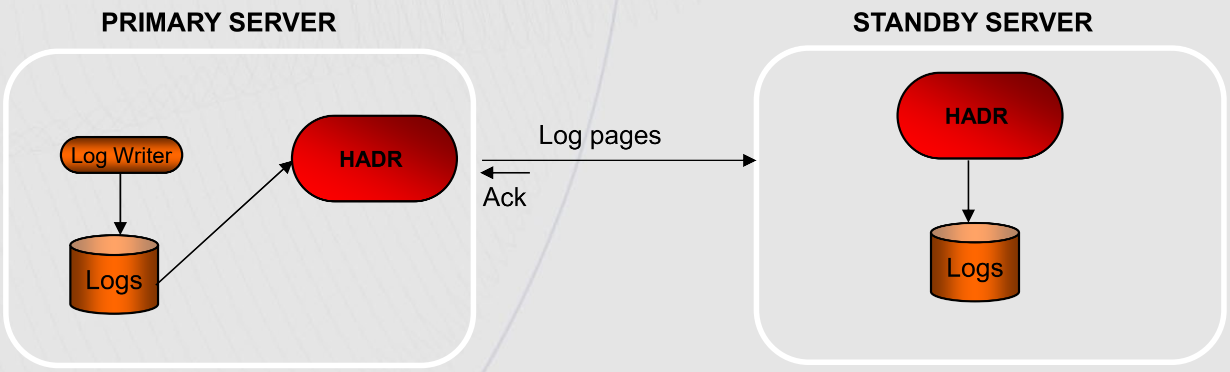
Synchronization Modes

- 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!

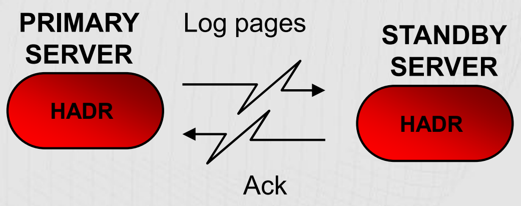
- **The TCP socket buffer size can be set in one of two way:**
  1. At the operating system level
     - the settings is applicable across all TCP connections on the server

  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 Buffer Size – DB2_HADR_SOSNDBUF

- 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 send_rate * 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 Superasync: 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

Monitoring TCP buffer size

- Look for the following fields in db2pd -hadr or MON_GET_HADR table function:
  - SOCK_SEND_BUF_REQUESTED
  - SOCK_RECV_BUF_REQUESTED
  - SOCK_SEND_BUF_ACTUAL
  - SOCK_RECV_BUF_ACTUAL
- The host machine may round up or down the requested size to certain sizes
  - Like power of two or multiple of network packet size
  - Or cap the requested size at system limit without returning an error
- Therefore actual size does not always match requested size
  - DB2 does not fail HADR startup if actual size is smaller than requested size
  - Verify the actual size against the requested size
- Some OS may assign a buffer size on socket creation, then adjust the size upon connection
  - The data returned before connection reflect the initial size on socket creation
  - The data returned after connection will reflect the adjusted size

Monitoring other 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.

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

HADR Log Spooling – a remedy for a slow standby

- A slow standby can fall behind playing log records sent by the primary
- The primary in the meantime is blocked waiting for an acknowledgment
- Log spooling decouples a slow standby log replay from a primary log shipping
- Log spooling allows log shipping regardless of log replay position on the standby
- Sync mode is with respect to LOG RECEIVE not LOG REPLAY
- When this feature is enabled, log data sent by the primary is spooled, or written, to disk on the standby, and that log data is later read by log replay
  - In pureScale, spooling allows log shipping on a fast stream to continue even if log merge is blocked on a slower stream
- Configured via the HADR_SPOOL_LIMIT database configuration parameter.
  - The default is AUTOMATIC which is (LOGPRIMARY + LOGSECOND) log files, computed at HADR startup
  - A value of -2 allows for space up to the limit of the filesystem

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 (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

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
- Tune log replay
  - 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

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 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

- It HADR_TIMEOUT is too short, HADR may get too many false alarms
  - Resulting in breaking the connection more often than necessary

- <u>BEST PRACTICES</u>:
  - 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

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 close 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

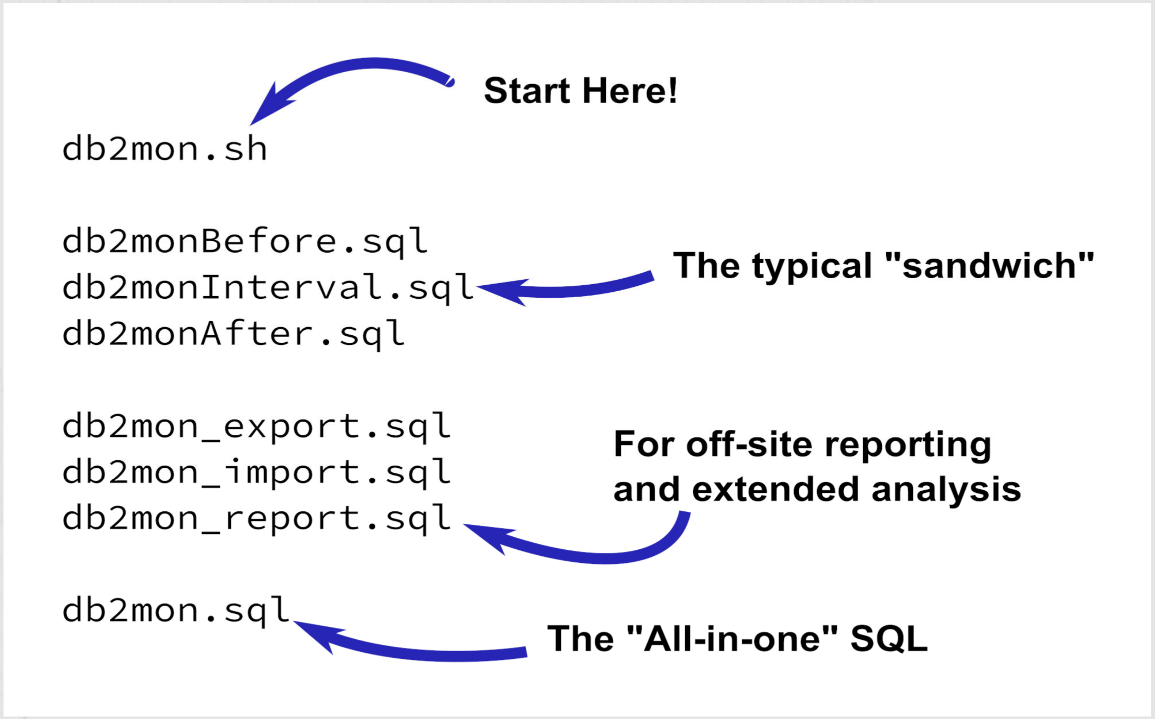
- What is HADR
- How to configure HADR for optimal performance
- Monitoring options
- Best practices

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
  - Standy 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

Db2mon – shipped in all current Db2 releases

- 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 <u>only</u>



```
db2mon.sh

db2monBefore.sql
db2monInterval.sql
db2monAfter.sql

db2mon_export.sql
db2mon_import.sql
db2mon_report.sql

db2mon.sql
```

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

```
vfulswUrrw@'KRPH2vtoole2vdpsohv2shui
h{sruwGE5RSWIRQV@».f0wyi»
ge5 'vfulswUrrw2ge5prqEhiruh1vto
ge5 'vfulswUrrw2|rxuVfulsw1vto
ge5 'vfulswUrrw2ge5prqDiwhu1vto
```

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

Note: all DB2MON statements have this comment

- 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

```
2 - IEP bGE5P RQ -2 vhdhfwp lq+wv,
fdswxuhbwlp h iurp
p rqbfxuuhqwbvt dbsoxvbvwduw

FDSWXUHbWLP H
0000000000000000000000000000
535603605<05513:18;17<3667

 4 uhfrug+v, vhdhfwhg1
```

End Data Capture

Note: all DB2MON statements have this comment

- 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

```
2- IEP bGE5P RQ -2 vhchfwp lq+wv,
fdswxuhbwlp h iurp
p rqbfxuuhqwbvtdbsoxvbhqg

FDSWXUHbWLP H
00000000000000000000000000
5356036 05<05513;16315<9<74

 4 uhfrug+v, vhchfwhg1
```

Analysis Report

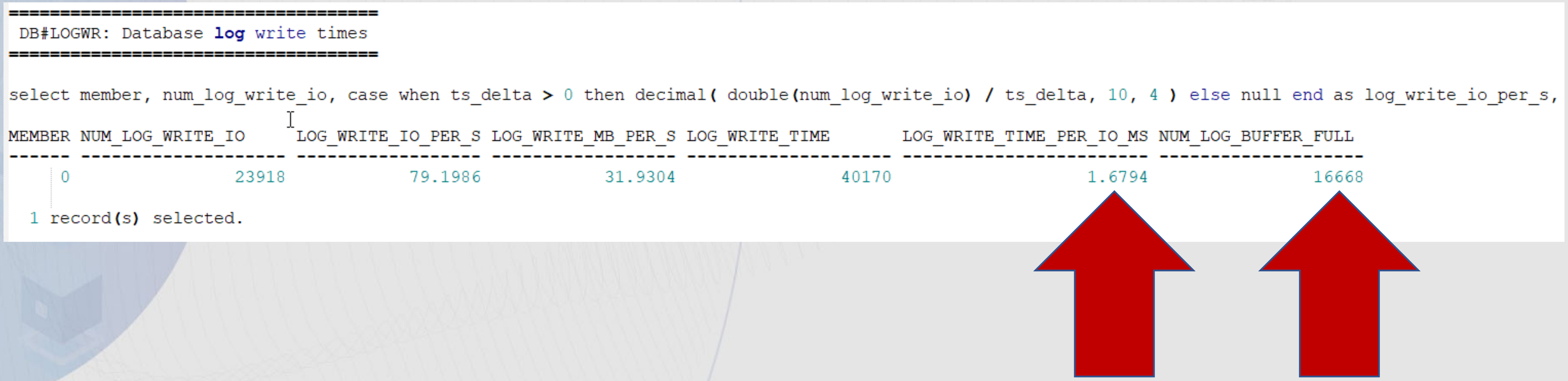
- Labelled by "scope" with "TAG#":
  - INS# - instance
  - DB# - database
  - CFG# - configuration
  - TSP# - tablespace
  - BPL# - bufferpool
  - PAG# - page
  - LTC# - latch
  - BLU# - column-organized tables
  - TBL# - table
  - IDX# - index
  - CON# - connection
  - WLB# - workload balancing
  - 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

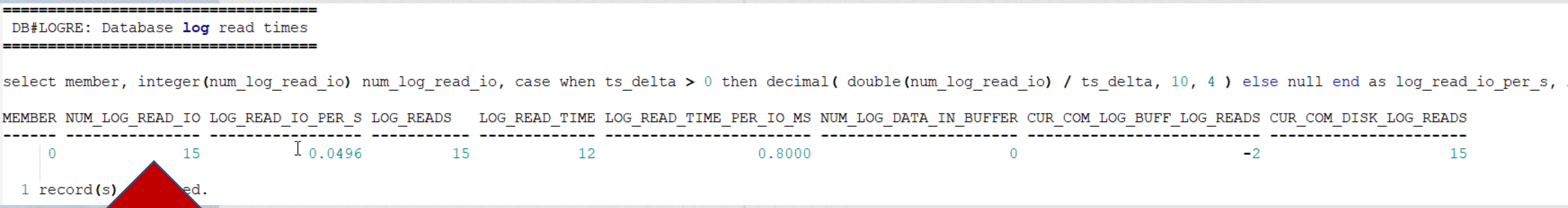
BLU	1
BPL	9
CF	12
CFG	3
CON	3
DB	16
IDX	4
INF	1
INS	1
LTC	1
PAG	2
PKG	1
SQL	7
TBL	2
TSP	7
WLB	1

DB Log Write Times



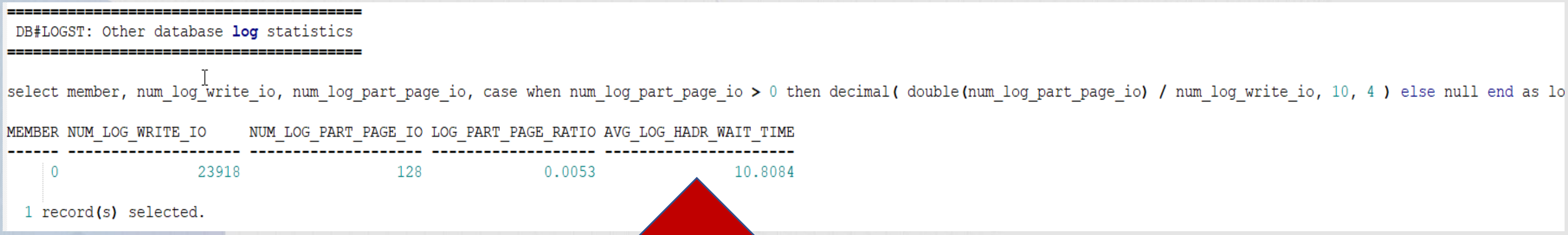
```
======================================
 DB#LOGWR: Database log write times
======================================

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)       ed.
```

Other HADR Log Stats



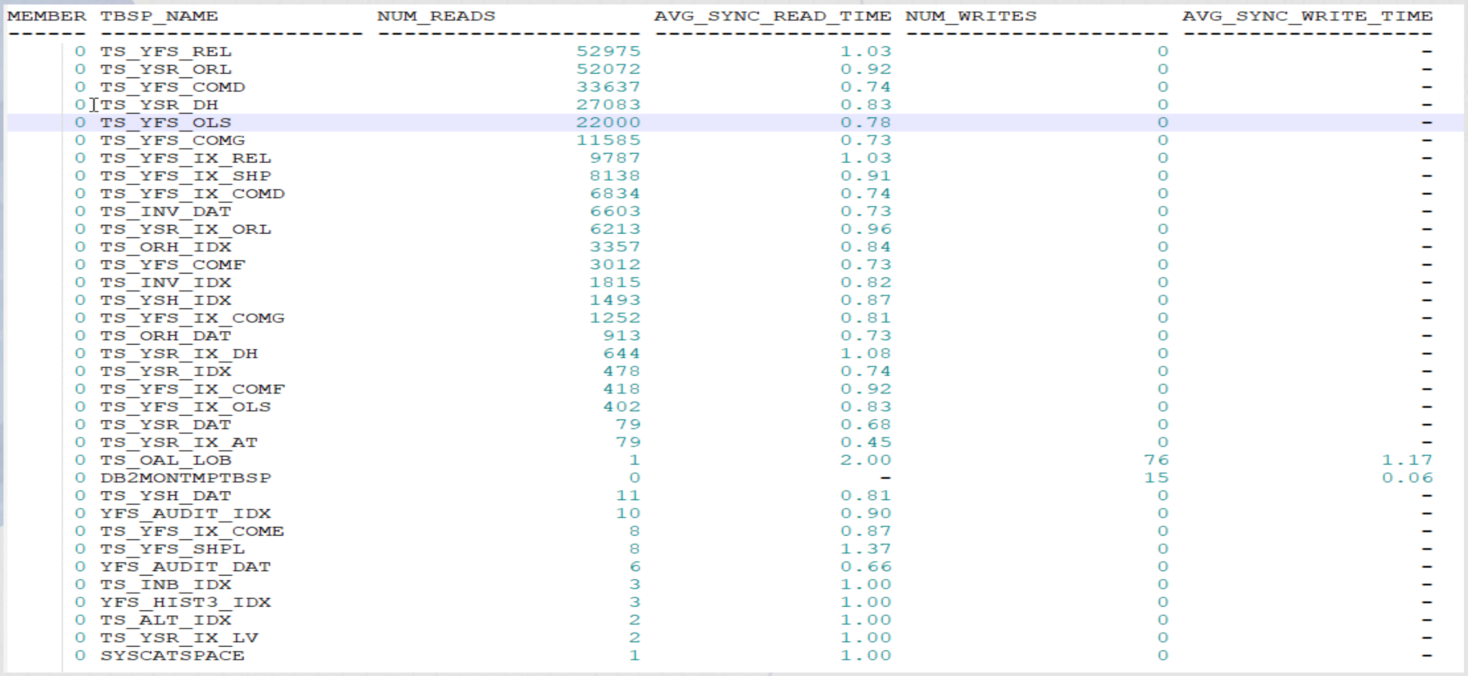
```
==========================================
 DB#LOGST: Other database log statistics
==========================================

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 for

==

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_SHPL                             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                   -
```

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 considation are meet
  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
```

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

- **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

Using the HADR simulator

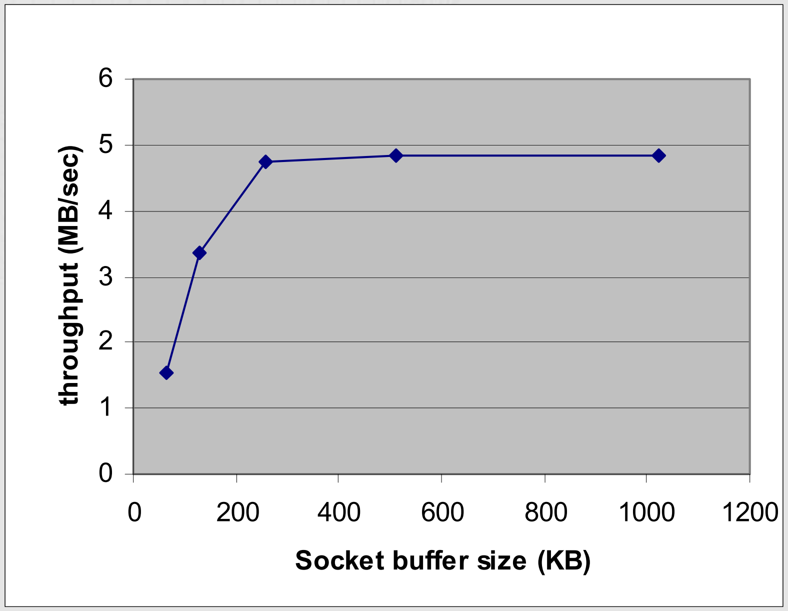
- Below is actual result between IBM labs in Portland, Oregon, and Silicon Valley, California
- The physical distance is about 1000km

- 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
- It 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
  - write_time = per_write_overhead + data_amount / 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.

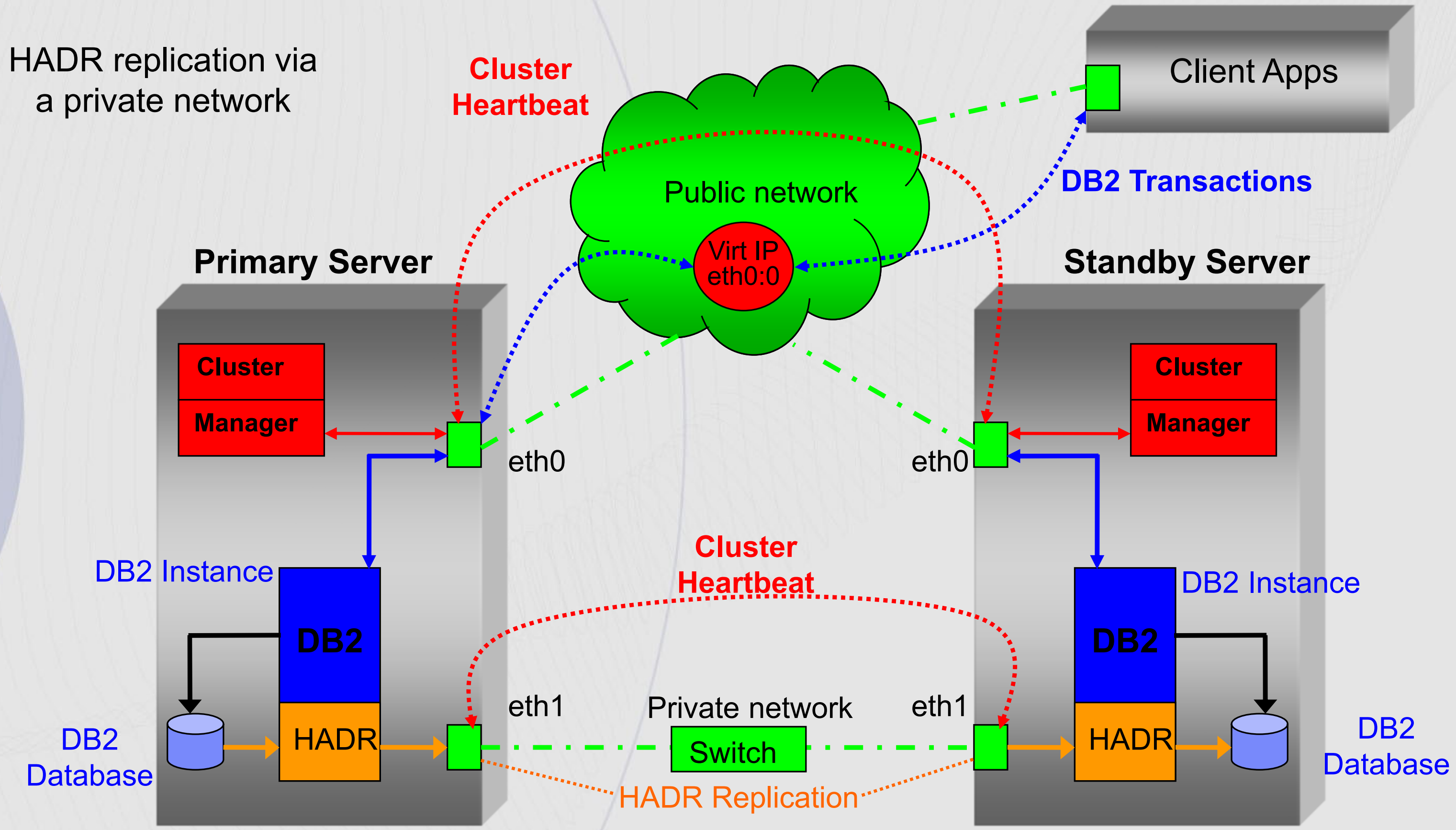
Use of a NICs

Use a dedicated NIC for HADR traffic

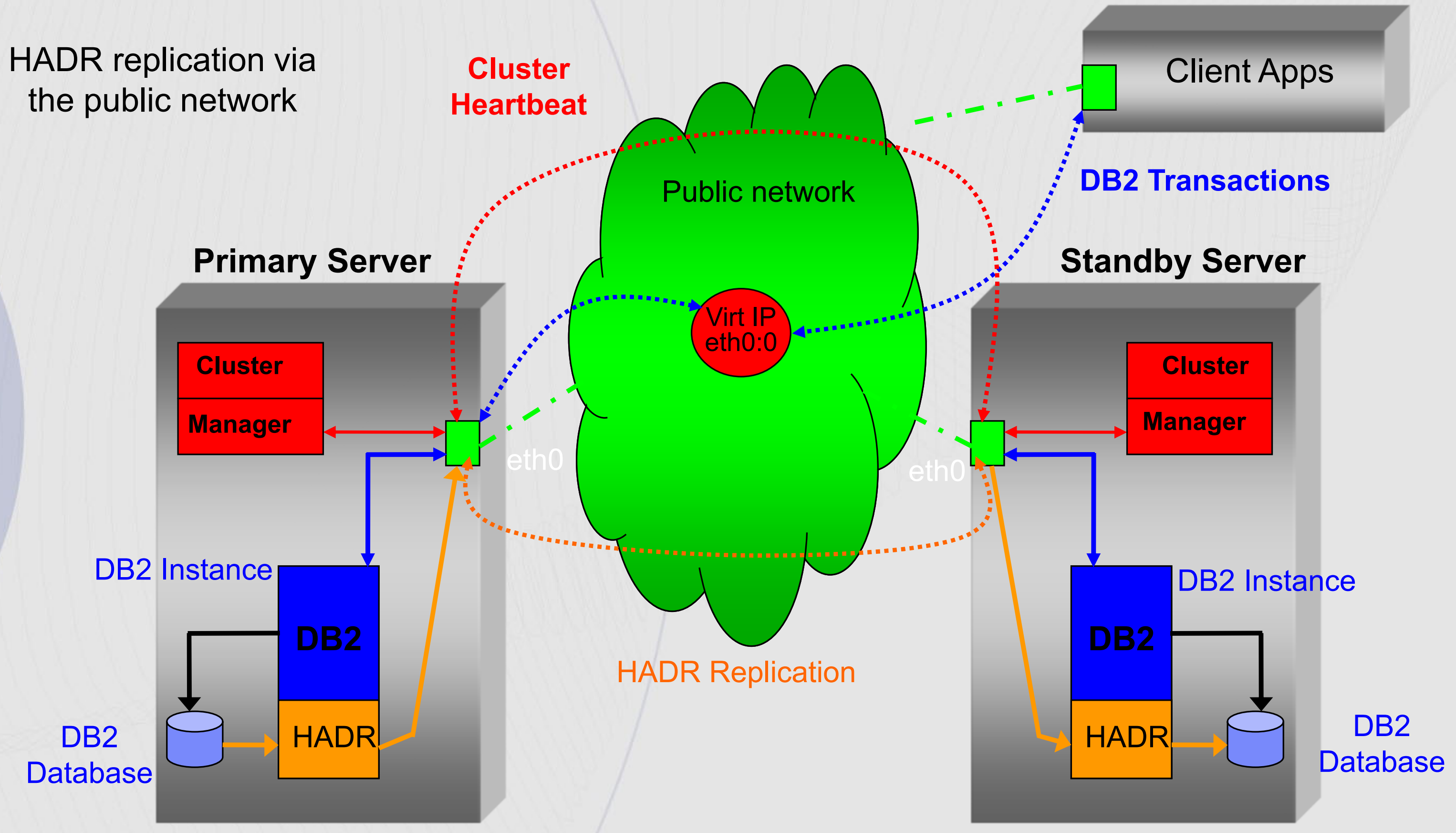
- If possible, use a dedicated private NIC for HADR log transmission between the nodes, including AIX standbys in a remote DC

- This will provide better performance as the NIC card will not get flooded with both user and HADR traffic.

Preferred DB2 HADR environment



Alternate example of a DB2 HADR environment



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
  - 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"
  - To tune down the number of threads, use DB2 registry variable DB2BPVARS:

    ```
    db2set DB2BPVARS=<path to buffer pool config file>
    In the config file, put this line:
    PREC_NUM_AGENTS=<number of threads>
    ```
  - 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

Agenda

- List Bulleted agenda items

IDUG

2024 NA **Db2** Tech Conference

@IDUGDb2
#IDUG_NA24

IDUG

Db2 HADR Performance Tuning - How to make it fly

Dale McInnis

dmcinnis@ca.ibm.com

RAS3

Please fill out your session evaluation!