Oracle Maximum Availability Architecture

Best Practices for Synchronous Redo Transport

Data Guard and Active Data Guard

ORACLE WHITE PAPER | MARCH 2015

ORACLE

Table of Contents

Introduction

The Oracle Maximum Availability Architecture (MAA) with Oracle Data Guard provides the most comprehensive solution available to eliminate single points of failure for mission critical Oracle Databases. It prevents data loss and downtime in the simplest and most economical manner by maintaining one or more synchronized physical replicas of a production database at a remote location. If the production database becomes unavailable for any reason, client connections can quickly, and in some configurations transparently, failover to a synchronized replica to immediately restore service. Active Data Guard extends basic Data Guard capabilities to eliminate the high cost of idle redundancy by enabling reporting applications, ad-hoc queries, data extracts, and fast incremental backups to be offloaded to read-only copies of the production database. Active Data Guard's complete focus on real-time data protection and availability and its deep integration with the Oracle Database eliminates compromises in data protection, performance, availability and complexity found in storage remote mirroring or other host-based replication solutions.

Data Guard and Active Data Guard are the only Oracle-aware replication solutions can guarantee zero data loss failover to an already running, synchronized copy of a production database. New capabilities with Active Data Guard 12c extend zero data loss protection using synchronized replicas located at any distance from the production database, without impacting production database performance or adding complexity to failover operations. This provides both high availability and data protection for the Oracle database in the event of database, cluster, site, region, and geographic outages.

This paper provides Oracle MAA best practices for using synchronous redo transport in a Data Guard or Active Data Guard configuration. It is designed for database administrators who have a working knowledge of Data Guard and Active Data Guard configurations using Maximum Availability or Maximum Protection modes. These are the modes where the integration of synchronous redo transport with the Data Guard managed recovery process (MRP) provides a guarantee of zero data loss should there be an unplanned outage of the production database.

.

Data Guard Synchronous Transport – an Overview

A Data Guard configuration includes a production database referred to as the primary database, and up to 30 directly connected replicas referred to as standby databases. Primary and standby databases connect over TCP/IP using Oracle Net Services. There are no restrictions on where the databases are physically located provided they can communicate with each other. A standby database is initially created from a backup of the primary database. Data Guard automatically synchronizes the primary database and all standby databases by transmitting primary database redo - the information used by every Oracle Database to protect transactions - and applying it to the standby database.

Data Guard transport services handle all aspects of transmitting redo from a primary to a standby databases(s). As users commit transactions at a primary database, redo records are generated and written to a local online log file. Data Guard transport services simultaneously transmit the same redo directly from the primary database log buffer (memory allocated within system global area) to the standby database(s) where it is written to a standby redo log file. Data Guard transport is very efficient for the following reasons:

» Data Guard's direct transmission from memory avoids disk I/O overhead on a primary database. This is different from how many other host-based replication solutions increase I/O on a primary database by reading data from disk and then transmitting the changes.

» Data Guard transmits only database redo. This is in stark contrast to storage remote-mirroring which must transmit every changed block in every file in order to maintain real-time synchronization. Oracle tests have shown that storage remote-mirroring transmits up to 7 times more network volume, and 27 times more network I/O operations than Data Guard. For a more complete discussion of the advantages of Data Guard compared to storage remote-mirroring solutions refer to *Oracle Active Data Guard vs Storage Remote Mirroring*.[1]

Data Guard offers two choices of transport services: synchronous and asynchronous. Synchronous redo transport - the subject of this paper - requires a primary database to wait for confirmation from the standby that redo has been received and written to disk (a standby redo log file) before commit success is signaled to the application. Synchronous transport combined with the deep understanding of transaction semantics by Data Guard apply services provides a guarantee of zero data loss protection should the primary database suddenly fail.

Data Guard also provides three different modes of operation that help users balance cost, availability, performance, and data protection - shown in Table 1. Each mode defines the behavior of the Data Guard configuration if a primary database loses contact with its standby. Two of the three modes use synchronous transport.

TABLE 1: DATA GUARD PROTECTION MODES

Mode	Risk of data loss	Transport	If no acknowledgement from the standby database, then:
Maximum Protection	Zero data loss Double failure protection	SYNC	Signal commit success to the application only after acknowledgement is received from a standby database that redo for that transaction has been hardened to disk. The production database cannot proceed until acknowledgement has been received.
Maximum Availability	Zero data loss Single failure protection	SYNC FAST SYNC FAR SYNC	Signal commit success to the application only after acknowledgement is received from a standby database or after NET_TIMEOUT threshold period expires – whichever occurs first. A network or standby database outage does not affect the availability of the production database.
Maximum Performance	Potential for minimal data loss	ASYNC	Primary never waits for standby acknowledgment to signal commit success to the application. There can be no guarantee of zero data loss.

1 http://www.oracle.com/technetwork/database/availability/dataguardvsstoragemirroring-2082185.pdf

Synchronous Transport Performance

Data Guard also provides a number of performance advantages compared to synchronous solutions based upon storage remote- mirroring. Recall from previous discussion that Data Guard only transmits redo. Storage remote-mirroring, in contrast, must transmit every change to every block to maintain the same real-time protection offered by Data Guard. This means storage remote-mirroring transmits the volume of data transmitted by Data Guard *plus* all writes to: data files, additional members of online log file groups, archived log files, control file, flashback log files, etc. The impact is significant. For example, the Oracle process that writes to data files is called Database Writer (DBWR) and anything that slows down DBWR can have a significant impact on database performance. Synchronous storage remote mirroring will impact DBWR by design, because each write by DBWR is delayed by the round trip network latency (RTT) between mirrored volumes. Data Guard is designed never to affect DBWR on the primary database.

Oracle has conducted numerous tests to characterize the impact of synchronous transport on a production database. The results of two representative tests are provided below. This data provides a general perspective on performance impact – it should not be used to extrapolate an expected impact for your production workloads.

Each application will have a different tolerance for synchronous replication. Differences in application concurrency, number of sessions, the transaction size in bytes, how often sessions commit, and log switch frequency – result in differences in impact from one application to the next even if round-trip network latency (RTT), bandwidth and log file write i/o performance are all equal. In general Oracle sees customers having greater success with synchronous transport when round trip network latency is less than 5ms, than when latency is greater than 5ms. Testing is always recommended before drawing any specific conclusions on the impact of synchronous replication on your workloads.

Test 1: OLTP Workload, Small Inserts

Swingbench, (a public domain load generator[2]) was used to simulate OLTP workload that generated redo at a rate of 30MB/second. Results showed 3% performance impact at 1ms RTT and 5% impact at 5ms RTT (see Figure 1).

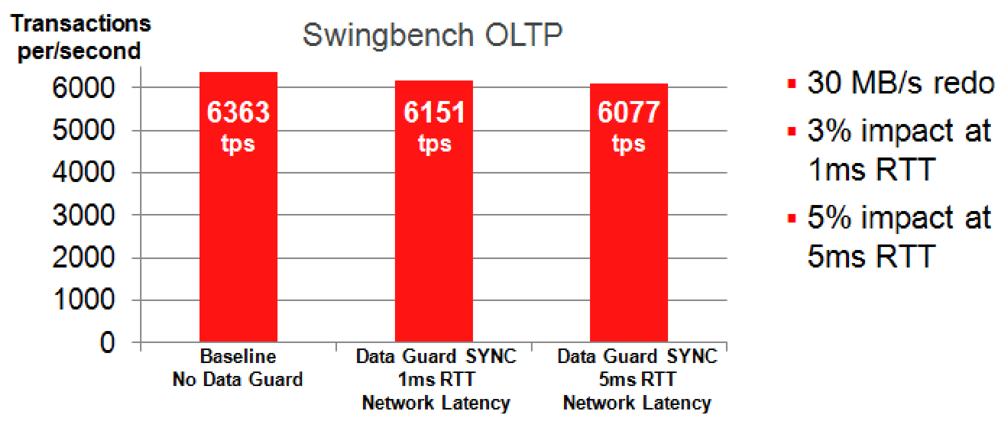

Figure 1: Performance Impact of Synchronous Transport for OLTP Workload - Small Inserts

2 http://dominicgiles.com/swingbench.html

The Swingbench workload in Test 1 had the following characteristics:

» Random DMLs, 1ms think time, 400 users, 6000+ transactions per second, 30MB/s peak redo rate
» Small inserts: 5K redo size, 120 logical reads, 30 block changes per transaction
» Three test runs: no Data Guard, and two Data Guard test runs with round-trip network latencies of 1 and 5ms
» Exadata 2-node RAC database, Oracle 11.2.0.4, Exadata Smart Flash Logging, and Write-Back flash cache

Test 2: OLTP Workload, Large Inserts

A second test with the same system and database configuration profiled the impact on OLTP workload when average transaction size was increased to 440K along with a corresponding increase in redo throughput. There was a 1% performance impact at <1ms RTT and 7% impact at 2ms RTT, and 12% impact at 5ms RTT (see Figure 2).

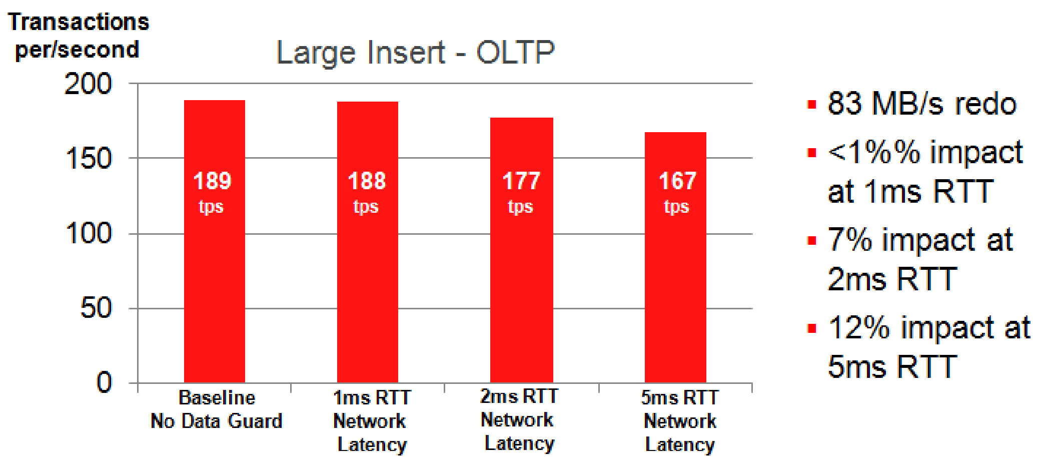

Figure 2: Performance Impact of Synchronous Transport for OLTP Workload - Large Inserts

The Swingbench workload in Test 2 had the following characteristics:

» Large insert OLTP workload: 180+ transactions per second, 83MB/s peak redo rate, random tables
» Transaction profile: 440K redo size, 6000 logical reads, 2100 block changes per transaction
» Baseline with no Data Guard, then Data Guard with 1, 2 and 5ms RTT network latencies

Synchronous Transport Enhancements

Synchronous transport has evolved over the course of numerous Oracle Database Releases. The technical details of synchronous transport and related Oracle MAA best practices are identical to both Data Guard and Active Data Guard. All Data Guard best practices described in this paper apply to Active Data Guard as well.

Oracle Database 11g Release 2

Data Guard 11g Release 2 reduces the performance impact of synchronous transport by transmitting redo to the remote standby in parallel with LGWR writing redo to the local online log file of the primary database – reducing the total round trip time required by synchronous replication. This is an improvement over previous Data Guard releases where transport would wait for the local log file write to complete before transmitting redo to the remote standby. The reduction in total round trip time enables greater geographic separation between primary and standby databases in a synchronous zero data loss configuration. Alternatively, on low latency networks it can reduce the impact of SYNC replication on primary database performance to near zero. This same architecture is used by Oracle Database 12c.

Oracle Database 12c

Data Guard Fast Sync (SYNC NOAFFIRM) with Oracle Database 12c provides an easy way of improving performance in synchronous zero data loss configurations. Fast Sync allows a standby to acknowledge the primary database as soon as it receives redo in memory, without waiting for disk I/O to a standby redo log file. This reduces the impact of synchronous transport on primary database performance by further shortening the total round-trip time between primary and standby. Fast Sync can introduce a very small exposure to data loss should simultaneous failures impact both primary and standby databases before the standby I/O completes. The time interval, however, is so brief (both failures must occur within milliseconds of the other) and the circumstances so unique that there is a very low likelihood of this occurring. Fast Sync is included with Data Guard 12c (an Active Data Guard license is not required). Fast Sync must be explicitly enabled by the administrator otherwise Data Guard 12c synchronous redo transport defaults to the same behavior as Data Guard 11g (SYNC AFFIRM).

Active Data Guard 12c includes a new capability called Far Sync. Active Data Guard Far Sync enables zero data loss protection even when a primary and standby database are located hundreds or thousands of miles apart, reducing or eliminating the impact to the production database of synchronous communication across a wide area network. When combined with Oracle Advanced Compression, Far Sync enables off-host redo transport compression, conserving bandwith with zero overhead on the production system. Far Sync is out of scope of this paper, for more information and best practices see, *Oracle Active Data Guard Far Sync - Zero Data Loss at Any Distance*[3].

Configuration Best Practices

The following MAA best practices are designed to minimize the performance impact of configuring Data Guard synchronous redo transport to achieve zero data loss protection for a production database.

Set tcp socket buffer size to 3 x BDP

For optimal network throughput the minimum recommended settings for TCP send and receive socket buffer sizes is a value equal to the bandwidth-delay product (BDP) of the network link between the primary and standby systems. Settings higher than the BDP may also yield incremental improvement. For example, MAA tests simulating high-latency, high-bandwidth networks continued to realize small, incremental increases in throughput as TCP send and receive socket buffer settings were increased to 3xBDP.

BDP is product of the network bandwidth and latency. Socket buffer sizes are set using the Oracle Net parameters RECV_BUF_SIZE and SEND_BUF_SIZE, so that the socket buffer size setting affects only Oracle TCP connections. The operating system may impose limits on the socket buffer size that must be adjusted so Oracle can use larger values. For example, on Linux, the parameters net.core.rmem_max and net.core.wmem_max limit the socket buffer size and must be set larger than RECV_BUF_SIZE and SEND_BUF_SIZE.

3 http://www.oracle.com/technetwork/database/availability/farsync-2267608.pdf

For example, if bandwidth is 622 Mbits and latency is 30 ms, then you would calculate the minimum size for the RECV_BUF_SIZE and SEND_BUF_SIZE parameters as follows:

Bandwidth Delay Product (BDP) = bandwidth x latency

BDP = 622,000,000 (bandwidth) / 8 x 0.030 (latency) = 2,332,500 bytes.

Given this example the optimal send and receive socket buffer sizes are calculated as follows:

Socket buffer size = 3 x BDP

= 2,332,500 (BDP) x 3

= 6,997,500 bytes

The size of the socket buffers can be set at the operating system level or at the Oracle Net level. As socket buffer size requirements can become quite large (depending on network conditions) it is recommended to set them at the Oracle Net level so that normal TCP sessions, such as telnet, do not use additional memory. Please note that some operating systems have parameters that set the maximum size for all send and receive socket buffers. You must ensure that these values have been adjusted to allow Oracle Net to use a larger socket buffer size.

With Oracle Net you can set the send and receive socket buffer sizes globally for all connections using the following parameters in the sqlnet.ora:

```
RECV_BUF_SIZE=6997500
SEND_BUF_SIZE=6997500
```

If you only want the larger buffer sizes for the connections associated with Data Guard transport then we recommend you configure RECV_BUF_SIZE and SEND_BUF_SIZE in the Oracle Net alias used for transport as well as in the listener on the standby host. The following shows an example of the send and receive socket buffer size being set as a description attribute for a particular connect descriptor:

```
standby =
 (DESCRIPTION=
   (SEND_BUF_SIZE=6997500)
   (RECV_BUF_SIZE=6997500)
     (ADDRESS=(PROTOCOL=tcp)
     (HOST=stby_host)(PORT=1521))
   (CONNECT_DATA=
     (SERVICE_NAME=standby)))
```

The socket buffer sizes must be configured the same for all databases within a Data Guard configuration. On a standby side or the receiving side this can be accomplished within either the sqlnet.ora or listener.ora file. In the listener.ora file, you can either specify the buffer space parameters for a particular protocol address or for a description.

```
LISTENER =
  (DESCRIPTION=
    (ADDRESS=(PROTOCOL=tcp)
    (HOST=stby_host)(PORT=1521)
    (SEND_BUF_SIZE=9375000)
    (RECV_BUF_SIZE=9375000)))
```

Configure standby redo logs

Online redo logs and standby redo logs should use redo log size = 4GB or redo log size >= peak redo rate/minute x 20 minutes. To extract peak redo rates, please refer to AWR reports during peak workload periods such as batch processing, quarter or year end processing. It is very important to use peak workload and not averages (averages can obscure peak redo rates and lead to provisioning insufficient network bandwidth). Table 2 provides a quick mapping of redo-rate to the minimum recommended redo log size:

TABLE 2: RECOMMENDED REDO LOG SIZE

Peak redo rate according to EM or AWR reports	**Recommended redo log group size**
<= 5 MB/sec	4 GB
<= 25 MB/sec	16 GB
<= 50 MB/sec	32GB
> 50 MB/sec	64 GB

Once the online redo logs have been appropriately sized you should create standby redo logs of the same size**. It is critical for performance that standby redo log groups only contain a single member.** In addition, for each redo log thread (a thread is associated with an Oracle RAC database instance), the number of Standby Redo Logs = number of Redo Log Groups + 1.

The additional standby redo log eliminates the possibility of a standby database waiting on standby redo log. For example, if a primary database has two instances (threads) and each thread has three online log groups, then you should pre-configure 8 standby redo logs on the primary database and each standby database. Furthermore, if the primary or standby databases are not symmetrical Real Application Clusters (e.g. an 8-node primary Oracle RAC cluster and a 2-node standby Oracle RAC cluster), then the primary and standby databases should have an equal number of standby redo logs (based upon the largest cluster in the configuration) and all threads should be represented. The size of the standby redo logs must always be exactly the same size as the online redo logs on the primary.

Be sure to place single member standby redo log groups in the fastest available diskgroup. The objective is to have standby log file write times that are comparable to log file I/O on the primary database optimal performance.

Set SDU size to 65535

With Oracle Net Services it is possible to control data transfer by adjusting the size of the Oracle Net setting for the session data unit (SDU). Oracle testing has shown that setting the SDU to its maximum value of 65535 can improve performance of SYNC transport. You can set SDU on a per connection basis using the SDU parameter in the local naming configuration file (TNSNAMES.ORA) and the listener configuration file (LISTENER.ORA), or you can set the SDU for all Oracle Net connections with the profile parameter DEFAULT_SDU_SIZE in the SQLNET.ORA file.

Note that the ASYNC transport uses the new streaming protocol and increasing the SDU size from the default has no performance benefit for asynchronous configurations.

Configure sufficient resources for optimal system performance

Sufficient resources, in particular for log file I/O on both the primary and standby databases and for network bandwidth between primary and standby locations, are critical to the performance of a synchronous Data Guard configuration. While Fast SYNC with Oracle Database 12c will eliminate slow I/O on a standby database from impacting production database performance, sufficient I/O performance for primary database log file writes and sufficient network bandwidth to transmit redo volume are still required for optimal results.

Use Fast Sync - SYNC NOAFFIRM

Fast Sync (SYNC NOAFFIRM) enables all sessions waiting for a remote RFS write to proceed as soon as the write is submitted, not when the write completes. This offers the best performance since it reduces the overall SYNC remote write wait event.

Consider Exadata for enhanced performance in a zero data loss configuration

Exadata has several features that make it an optimal system for deploying SYNC configurations.

» Smart Flash Logging: The Exadata Smart Flash Cache implements a special algorithm to reduce the latency of log write I/Os called Exadata Smart Flash Logging. The time to commit user transactions or perform critical updates is very sensitive to the latency of log writes. Smart Flash Logging takes advantage of the flash cache in Exadata storage combined with the high speed RAM memory in the Exadata disk controllers to greatly reduce the latency of log writes and avoid the latency spikes that frequently occur in all storage solutions including flash based storage solutions. The Exadata Smart Flash Logging algorithms are unique to Exadata and speed up both primary and standby log writes.

» Networking: Exadata compute nodes have multiple 1GigE and 10GigE network adapters that are preconfigured for high availability and provide ample bandwidth to handle high transport rates.

» Storage Performance: Exadata is a modern architecture featuring scale-out industry-standard database servers, scale-out intelligent storage servers, state-of-the-art PCI flash storage servers and an extremely high speed InfiniBand internal fabric that connects all servers and storage. Unique software algorithms in Exadata implement database intelligence in storage, PCI based flash, and InfiniBand networking to deliver higher performance and capacity at lower costs than other platforms.

TUNING

Data Guard performance is directly dependent upon the performance of primary and standby systems, the network that connects them, and the IO subsystem. Understanding the topology of the Data Guard configuration and its relevance to Data Guard performance helps eliminate infrastructure weaknesses that are often misattributed to Data Guard architecture.

Configuration pre-requisites

Data Guard architecture is very streamlined and efficient however like any application, there are reasonable perquisites needed to achieve satisfactory performance:

Primary Database

» Sufficient CPU utilization for LGWR to post foregrounds efficiently

» Sufficient I/O bandwidth so local log writes maintain low I/O latency during peak rates

» Network interfaces that can handle peak redo rate volumes combined with any other network activity across the same interface

» Primary AWR, ASH and OSwatcher data that is gathered for tuning and troubleshooting

Network:

» The round trip network latency (RTT) between primary and standby databases must not be so large that it impacts the primary database to the point where it compromises performance SLAs. This means that there is a practical limit to the distance between primary and standby databases since round-trip latency will increase as distance increases. Oracle is often asked what the maximum latency is that can be supported or whether there is an equation that can be used to project performance impact. Unfortunately every application will have a different ability to tolerate network latency. Differences in application concurrency, number of sessions, the transaction size in bytes, how often sessions commit, and log switch frequency – can cause differences in impact from one application to the next even if round-trip network latency, bandwidth and log file write i/o performance are all equal. In general Oracle sees customers having greater success with synchronous redo transport when round trip network latency is less than 5ms, than when latency is greater than 5ms.

» Sufficient network bandwidth to support peak redo rates (steady state and when resolving gaps) combined with any other network activity that shares the same network. Please note that your point to point network bandwidth will be throttled by the network segment, switch, router, interface with the lowest network bandwidth. If you have 10gigE for 90% of your network route and your existing switch or network interface only supports 1 GigE, then your maximum network bandwidth is 1 GigE.

» Netstat and/or any network monitoring stast should be gathered

Note: The top network problems encountered are inconsistent network response and insufficient network bandwidth.

Standby Database

» Sufficient CPU utilization for RFS (the Data Guard process that receives redo at the standby database) to efficiently write to standby redo logs.

» Sufficient I/O bandwidth to enable local log writes to maintain low I/O latency during peak rates.

» A network interface that can receive the peak redo rate volumes combined with any other network activity across the same interface

» Standby statspack, ASH and OSwatcher data should be gathered

Note: The top problem encountered with the standby database is poor standby log write latency due to insufficient I/O bandwidth. This problem can be mitigated by using Data Guard Fast Sync with Oracle 12c or by using Exadata.

Monitoring system resources

The following provides information on how to monitor system resources on primary and standby hosts.

Monitoring CPU

The uptime, mpstat, sar, dstat, and top utilities allow you to monitor CPU usage. When a system's CPU cores are all occupied executing work for processes, other processes must wait until a CPU core or thread becomes free or the scheduler switches a CPU to run their code. If too many processes are queued too often, this can represent a bottleneck in the performance of the system.

The commands mpstat -P ALL and sar -u -P ALL display CPU usage statistics for each CPU core and averaged across all CPU cores.

The %idle value shows the percentage of time that a CPU was not running system code or process code. If the value of %idle is near 0% most of the time on all CPU cores, the system is CPU-bound for the workload that it is running. The percentage of time spent running system code (%systemor %sys) should not usually exceed 30%, especially if %idle is close to 0%.

The system load average represents the number of processes that are running on CPU cores, waiting to run, or waiting for disk I/O activity to complete averaged over a period of time. On a busy system, the load average reported by uptime or sar -q should not exceed two times the number of CPU cores. If the load average exceeds four times the number of CPU cores for long periods, the system is overloaded.

In addition to load averages (ldavg-*), the sar -q command reports the number of processes currently waiting to run (the *run-queue size*, runq-sz) and the total number of processes (plist_sz). The value of runq-sz also provides an indication of CPU saturation.

Determine the system's average load under normal loads where users and applications do not experience problems with system responsiveness, and then look for deviations from this benchmark over time. A dramatic rise in the load average can indicate a serious performance problem.

Monitoring memory usage

The sar -r command reports memory utilization statistics, including %memused, which is the percentage of physical memory in use.

» sar -B reports memory paging statistics, including pgscank/s, which is the number of memory pages scanned by the kswapd daemon per second, and pgscand/s, which is the number of memory pages scanned directly per second.

» sar -W reports swapping statistics, including pswpin/s and pswpout/s, which are the numbers of pages per second swapped in and out per second.

If %memused is near 100% and the scan rate is continuously over 200 pages per second, the system has a memory shortage.

Once a system runs out of real or physical memory and starts using swap space, its performance deteriorates dramatically. If you run out of swap space, your programs or the entire operating system are likely to crash. If free or top indicate that little swap space remains available, this is also an indication you are running low on memory.

The output from the dmesg command might include notification of any problems with physical memory that were detected at boot time.

Monitoring I/O:

The iostat command monitors the loading of block I/O devices by observing the time that the devices are active relative to the average data transfer rates. You can use this information to adjust the system configuration to balance the I/O loading across disks and host adapters.

iostat -x reports extended statistics about block I/O activity at one second intervals, including %util, which is the percentage of CPU time spent handling I/O requests to a device, and avgqu-sz, which is the average queue length of I/O requests that were issued to that device. If %util approaches 100% or avgqu-sz is greater than 1, device saturation is occurring and the storage I/O Bandwidth needs to be augmented by adding disks or storage.

You can also use the sar -d command to report on block I/O activity, including values for %util and avgqu-sz.

The iotop utility can help you identify which processes are responsible for excessive disk I/O. iotop has a similar user interface to top. In its upper section, iotop displays the total disk input and output usage in bytes per second. In its lower section, iotop displays I/O information for each process, including disk input output usage in bytes per second, the percentage of time spent swapping in pages from disk or waiting on I/O, and the command name. Use the left and right arrow keys to change the sort field, and press A to toggle the I/O units between bytes per second and total number of bytes, or O to toggle between displaying all processes or only those processes that are performing I/O.

Monitoring network:

The ip -s link command displays network statistics and errors for all network devices, including the numbers of bytes transmitted (TX) and received (RX). The dropped and overrun fields provide an indicator of network interface saturation.

The ss -s command displays summary statistics for each protocol.

To monitor the current rate being transmitted via an interface use the sar –n DEV command.

Assessing database wait events

Once you have verified that you are not bottlenecked on any system or network resources you are ready to assess database wait events. On the primary database this is done using AWR reports while on the standby database you will use standby statspack reports (See My Oracle Support Note 454848.1 for complete details on Standby Statspack). Wait events specific to Data Guard with Oracle Database 11.2 are described in Table 3.

TABLE 3: WAIT EVENTS RELEVANT TO DATA GUARD 11.2

Event Name	Description
ARCH wait on ATTACH	Monitors the amount of time spent by all archiver processes to spawn a new RFS connection.
ARCH wait on SENDREQ	Monitors the amount of time spent by all archiver processes to write archive logs to the local disk as well as write them remotely.
ARCH wait on DETACH	Monitors the amount of time spent by all archiver processes to delete an RFS connection.
LNS wait on ATTACH	Monitors the amount of time spent by all LNS processes to spawn a new RFS connection.
LNS wait on SENDREQ	Monitors the amount of time spent by all LNS processes to write the received redo to disk as well as open and close the remote archived redo logs.
LNS wait on DETACH	Monitors the amount of time spent by all LNS processes to delete an RFS connection.
LGWR wait on LNS	Monitors the amount of time spent by the log writer (LGWR) process waiting to receive messages from LNS processes.
LNS wait on LGWR	Monitors the amount of time spent by LNS processes waiting to receive messages from the log writer (LGWR) process.
LGWR-LNS wait on channel	Monitors the amount of time spent by the log writer (LGWR) process or the LNS processes waiting to receive messages.

Oracle 11.2 wait events were replaced with the following events provided in Table 4 for all database versions from 12.1.0.1 onward.

TABLE 4: WAIT EVENTS RELEVANT TO DATA GUARD FROM ORACLE DATABASE 12C ONWARD

Event Name	Description
ARCH Remote Write	The time (in centi-seconds) that ARCn background processes spend blocked waiting for network write operations to complete.
ASYNC Remote Write	The time (in centi-seconds) for asynchronous streaming RFSWRITE operations.This includes stall reaps and streaming network submission time. This time is accumulated by TTnn (Redo Transport Slave) background processes..
Redo Transport Attach	The time spent (in centi-seconds) doing Connect, Logon, and RFSATTACH for any network process.
Redo Transport Close	The time spent (in centi-seconds) by ARCn, NSSn, and TTnn processes doing RFSCLOSE and RFSRGSTR operations.
Redo Transport Detach	The time spent (in centi-seconds) doing RFSDETACH and Disconnect for any network process..
Redo Transport Open	The time spent (in centi-seconds) by ARCn, NSSn, and TTnn background processes doing RFSCREAT and RFSANNCE operations.
Redo Transport Ping	The time spent (in centi-seconds) by ARCn background processes doing RFSPING operations.
Redo Transport Slave Shutdown	The time spent (in centi-seconds) by LGWR doing NSSn and TTnn process shutdown and termination.
Redo Transport Slave Startup	The time spent (in centi-seconds) by LGWR doing NSSn and TTnn process startup and initialization
Redo Writer Remote Sync Complete	The time spent (in centi-seconds) by LGWR reaping completed network writes to remote destinations.
Redo Writer Remote Sync Notify	The time spent (in centi-seconds) by LGWR issuing network writes to remote destinations.
Remote SYNC Ping	The time spent (in centi-seconds) by the LGWR and NSSn background processes doing synchronous PING operations.
SYNC Remote Write	The time spent by LGWR doing SYNC RFSWRITE operations.

Understanding how synchronous transport insures data integrity

The following algorithms ensure data consistency in a Data Guard synchronous configuration:

» LGWR redo write on the primary database and the Data Guard NSS network redo write are identical

» The Data Guard Managed Recovery Process (MRP) at the standby database can NOT apply redo unless the redo has been written to the primary's online redo log with the only exception being during a Data Guard failover operation (primary is gone). In addition to shipping redo synchronously, NSS and LGWR also exchange information regarding the safe redo block boundary that standby recovery can apply up to from its standby redo logs. This prevents the standby from applying redo it may have received, but which the primary has not yet acknowledged as committed to its own online redo logs.

The possible failure scenarios include:

» If primary database's LGWR cannot write to online redo log, then LGWR and instance will crash. Instance or crash recovery will recover to the last committed transaction in the online redo log and roll back any uncommitted transactions. The current log will be completed and archived.

» On the standby, we complete the partial standby redo log with the correct value for the size to match the corresponding online redo log. If any redo blocks are missing from the SRL, we ship these over (without reshipping the entire redo log).

» If the primary database crashes resulting in an automatic or manual zero data loss failover, then part of the Data Guard failover operation will do "terminal recovery" and read and recover the current standby redo log. Once recovery completes applying all the redo in the SRLs, the new primary database comes up and it archives the newly completed log group. All new and existing standby databases will discard any redo in its online redo logs, flashback to consistent SCN and only apply the archives coming from the new primary database. Once again the Data Guard environment is in sync with the (new) primary database.

Assessing performance

When assessing performance in a SYNC environment it is important to know how the different wait events relate to each other. The impact of enabling SYNC will vary between applications. To understand why consider the following description of work LGWR performs when a commit is issued:

1) Foreground process posts LGWR for commit ("log file sync" starts). If there are concurrent commit requests queued, LGWR will batch all outstanding commit requests together resulting in a continuous strand of redo.
2) LGWR waits for CPU
3) LGWR starts redo write ("redo write time" starts)
4) For RAC, LGWR broadcasts the current write to other instances
5) After preprocessing, if there is a SYNC standby, LGWR starts the remote write (“SYNC remote write” starts)
6) LGWR issues local write ("log file parallel write")
7) If there is a SYNC standby, LGWR waits for the remote write to complete
8) After checking the I/O status, LGWR ends "redo write time / SYNC remote write"
9) For RAC, LGWR waits for the broadcast ack
10) LGWR updates the on-disk SCN
11) LGWR posts the foregrounds
12) Foregrounds wait for CPU
13) Foregrounds ends "log file sync"

We recommend the following approach to assess performance:

For batch loads the most important factor is to monitor the elapsed time as most of these processes must be completed in a fixed period of time. The database workloads for these operations are very different than the normal OLTP workloads, for example the size of the writes can be significantly larger, so using log file sync averages does not give an accurate view / comparison.

For OLTP workloads we recommend monitoring the volume of transactions per second (from AWR) and the redo rate (redo size per second) from the AWR report. This gives a clear picture of the application throughput and how it is impacted by enabling SYNC.

How not to assess the impact of enabling SYNC:

Typically we see that the first place people look when assessing the impact of enabling SYNC is at the log file sync wait event on the primary. If the average log file sync wait before enabling SYNC was 3ms and after was 6ms then the assumption is that SYNC has impacted performance by one hundred percent. Oracle does not recommend using log file sync wait times to measure the impact of SYNC since the averages can be very deceiving and the actual impact of SYNC on response time and throughput may be much lower than the event indicates.

When a user session commits LGWR will go through the process of getting on the CPU, submitting the I/O, waiting for the I/O to complete, and then getting back on the CPU to post foreground processes that their commit has completed. This whole time period is covered by the ‘log file sync’ wait event. While LGWR is performing its work there are, in most cases, other sessions committing that must wait for LGWR to finish before processing their commits. The size and number of sessions waiting are determined by how many sessions an application has and how frequently those sessions commit. We generally refer to this batching up of commits as application concurrency.

For example, let’s assume that it normally takes 0.5ms to perform log writes (log file parallel write), 1ms to service commits (log file sync), and on average we are servicing 100 sessions for each commit. If there was an anomaly in the storage tier and the log write I/O for one commit took 20ms to complete then we could have up to 2,000 sessions waiting on log file sync while there would only be 1 long wait attributed to log file parallel write. Having a large number of sessions waiting on one long outlier can greatly skew the log file sync averages.

The output from v$event_histogram for the log file sync wait event for a particular period in time is shown in Table 5.

TABLE 5: FROM V$EVENT_HISTOGRAM FOR THE LOG FILE SYNC WAIT EVENT

Milliseconds	**Number of Waits**	**Percent of Total Waits**
1	17610	21.83%
2	43670	54.14%
4	8394	10.41%
8	4072	5.05%
16	4344	5.39%
32	2109	2.61%
64	460	0.57%
128	6	0.01%

The output shows that 92% of the log file sync wait times are less than 8ms with the vast majority less than 4ms (86%). Waits over 8ms, or what we call outliers, only comprise of 8% overall but due to the number of sessions waiting on those outliers (due to batching of commits) the averages get skewed. This skewing makes using log file sync average waits times as a metric for assessing the impact of SYNC misleading.

What is causing the outliers?

With synchronous transport any disruption to the I/O on the primary or standby, or spikes in network latency can also cause high log file sync outliers. We typically see this when the standby system has an inferior I/O subsystem from that of the primary. Often time’s customers will host multiple databases such as dev and test on standby systems that can impair IO response. It is important to monitor I/O by using iostat as described above to determine if disk are reaching maximum IOPS as this will affect the performance of SYNC writes.

Perhaps one of the biggest causes of outliers is frequent log switches. Consider what occurs on the standby when a log switch on the primary occurs.

1. RFS on the standby must finish updates to the standby redo log header
2. RFS then switches into a new standby redo log with additional header updates
3. Switching logs forces a full checkpoint on the standby. This causes all dirty buffers in the buffer cache to be written to disk causing a spike in write I/O. In a non-symmetric configuration where the standby storage subsystem does not have the same performance as the primary database, this will result in higher IO latencies.
4. The previous standby reo log must be archived increasing both read and write I/O.

The chart in Figure 3 was created using a heavy insert workload producing about 30MB/sec. It displays the time in milliseconds for each remote write for time periods that included a log switch and shows the increase is remote write times that occur at each log switch.

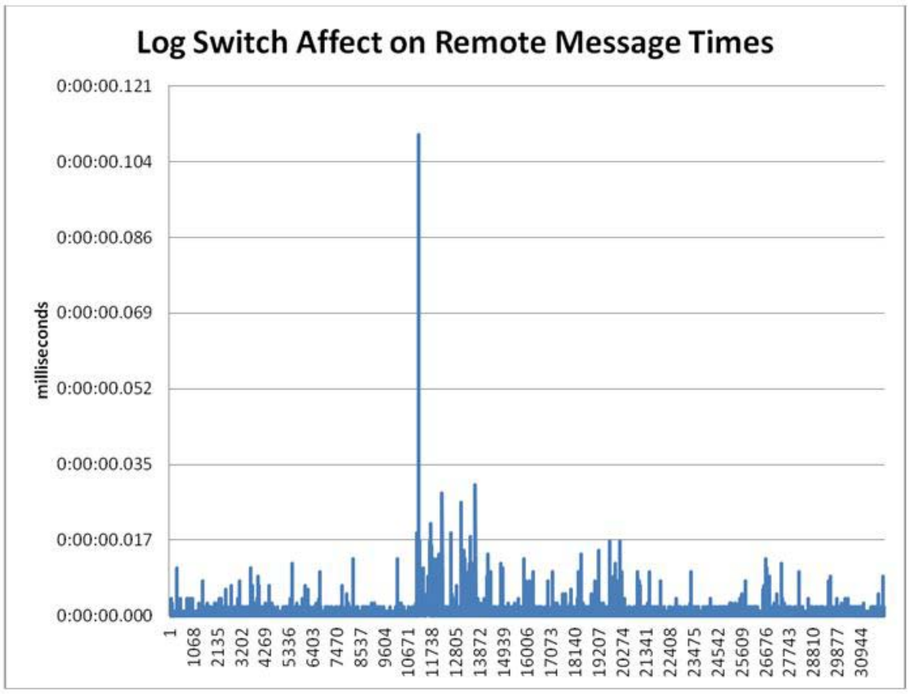

Figure 3: Log Switch Affect on Remote Message Times

What else does enabling SYNC affect?

When SYNC is enabled we introduce a remote write (RFS write to a standby redo log) in addition to the normal local write for commit processing (unless using Fast Sync with Oracle Database 12c). This remote write, depending on network latency and remote I/O bandwidth, can make commit processing increase in time. Since commit processing is taking longer that means that we will see more sessions waiting on LGWR to finish its work and begin work on their commit request, which is to say that application concurrency has increased. Increased application concurrency can be seen by analyzing database statistics and wait events. Consider the example in table 6.

TABLE 6: AFFECT OF SYNC TRANSPORT INCREASING APPLICATION CONCURRENCY

SYNC	Redo Rate	Network Latency	TPS from AWR	log file sync avg. ms	Log file parallel write avg. mis	RFS random I/O	SYNC remote write avg. ms	Redo write size (kb)	Redo writes
Defer	25MB	0	5,514.94	0.74	0.47	NA	NA	10.58	2,246,356
Yes	25MB	0	5,280.20	2.6	.51	.65	.95	20.50	989,791
Impact	0	-	-4%	+251%	+8.5%	NA	NA	+93.8%	-55.9%

In the above example we see that enabling sync reduced the number of redo writes but increase the size of each redo write. Since the size of the redo write has increased we should expect the time spent doing the I/O (both local and remote) to increase. We should expect 'log file sync' wait time to be higher since we are doing more work per wait. However, at the application level the impact to the transaction rate or the transaction response time might change very little as we are servicing more sessions per commit. This is why it is important to measure the impact of SYNC at the application level and not depend entirely on database wait events (more on that below). **It is also a perfect example of why log file sync wait event is a very misleading indicator of actual application impact.**

How to assess SYNC performance with Oracle Database 11.2

To look at performance we recommend calculating the time spent for local redo writes latency, average redo write size per write, and overall redo write latency. You can use the following wait events to do this:

» local redo write latency = 'log file parallel write'

» average redo write size per write = 'redo size' / 'redo writes'

» overall redo write latency (both local and remote) = 'redo write time' / 'redo writes' * 10

» average commit latency seen by foregrounds = 'log file sync'

» transaction per second

» redo rate

Stats from an AWR report from an Oracle 11.2 database are provided in Table 7. SYNC transport was enabled to a local standby with 1ms network latency to compare the performance impact to a baseline with SYNC disabled:

TABLE 7: ASSESSING SYNC PERFORMANCE WITH ORACLE DATABASE 11.2

Metric	Baseline – no SYNC	SYNC	Impact
redo rate (K/sec)	3,689	3,670	no change
log file sync	0.43	2.45	+469%
log file parallel write	0.30	0.30	no change
TPS	438	429	-2.1%
overall write latency	0.42	1.87	+345%
total redo size	3,418,477,080	3,450,849,356	+0.9%
redo writes	426,201	396,199	-7.0%
redo write size	8,020	8,709	+8.6%

In this example we see that enabling SYNC impacted overall transaction per second as seen by the application by 2% while the database redo rate declined by less than 1%. We also see that the number of redo writes decreased in number but increased in size. This is due to concurrency, or the increases in the number of sessions being serviced by any one commit or redo write. The average time spent writing to the standby was 1.87ms while spending on average less than half a millisecond on local writes. This means that of the 1.87ms for the remote write we spent 1ms on the network leaving approximately.87ms for the I/O to complete into the standby redo log.

How to assess SYNC performance with Oracle Database 12c

To look at performance we recommend calculating the time spent for local redo writes latency, average redo write size per write, and overall redo write latency. You can use the following wait events to do this:

» local redo write latency = 'log file parallel write'

» remote write latency = 'SYNC remote write'

» average redo write size per write = 'redo size' / 'redo writes'

» average commit latency seen by foregrounds = 'log file sync'

Stats from an AWR report from an Oracle 12c database are provided in Table 8. SYNC transport was enabled to a local standby with 1ms network latency to compare the performance impact to a baseline with SYNC disabled.

TABLE 8: ASSESSING SYNC PERFORMANCE WITH ORACLE DATABASE 12C

Metric	Baseline – no SYNC	SYNC	Impact
redo rate (MB/s)	25	25	no change
log file sync	0.68	4.60	+576%
log file parallel write	0.57	0.62	+8.8%
TPS	7,814.92	6224.03	-20.3%
RFS random I/O	NA	2.89	NA
SYNC remote write avg. ms	NA	3.45	NA
redo writes	2,312,366	897,751	-61,2%
redo write size (kb)	10.58	20.50	+93.8%

In the above example we see that log file sync waits averages increased dramatically after enabling SYNC. While the local writes remained fairly constant the biggest factor in increasing log file sync was the addition of the SYNC remote write. Of the SYNC remote write we know the network latency is zero so focusing on the remote write into the standby redo log shows an average time of 2.89ms. This was an immediate red flag given that primary and standby were using the same hardware and the remote write should be in line with the local write times.

In the above example it was determined that the standby redo logs (SRLs) had multiple members and they were placed into a slower performing diskgroup. After reducing to a single member and placing into a fast diskgroup the test was repeated yielding the results shown in Table 9.

TABLE 9: SYNC PERFORMANCE AFTER REDUCING SRLS TO SINGLE MEMBER AND PLACING ON FAST DISKGROUP

Metric	Baseline – no SYNC	SYNC	Impact
redo rate (MB/s)	25	25	no change
log file sync	0.67	1.60	+139%
log file parallel write	0.51	0.63	+23.5%
TPS	7714.36	7458.08	-3.3%
RFS random I/O	NA	.89	NA
SYNC remote write avg. ms	NA	1.45	NA
redo writes	2,364,388	996,532	-57.9%
redo write size (kb)	10.61	20.32	+91.5%

Due to faster I/O rates on the standby the overall log file sync waits were reduced which resulted in a higher application transaction per second rate.

Diagnosing High Log File SYNC Waits

When a long log write occurs there is a message similar to the following written to the LGWR trace file on the primary database:

```
Warning: log write elapsed time 163ms, size 18KB
```

By default Oracle only prints warnings for long log writes that are longer than 500ms. You can lower the threshold for these messages by using the following underscore parameter:

```
_long_log_write_warning_threshold=<some number of milliseconds>
```

Using the timestamps associated with the above warning enables you to diagnose whether the standby is the root cause by tracing the sending and receiving of redo. Set the following tracing:

- » On primary and standby set event 16410 (dynamic): alter system set 16410 trace name context forever, level 16
- » On the standby determine the pid of RFS by querying v$managed_standby and set event 10046 level 12.

With the above tracing set you can trace the time spent for each remote log write by tracking the message id. For example, for each remote log write we print the following message and place the sending timestamp below the message:

```
Client sending SQLNET request data [kpurcs] oper='Write' flag=67111042 thrd=1
seq=1584 msgid=117468

*** 2014-12-02 08:46:34.598 5837 krsu.c
```

On the standby side we can see the receipt of that remote log write by looking for the same msgid identified above:

```
... Server received SQLNET data [krsr_rfs_dsp] oper='Write' flag=67111042 thrd=1
seq=1584 msgid=117468

*** 2014-12-02 08:46:34.598 826 krsr.c
```

Once RFS has completed the write into the SRL, RFS sends an ack back to the primary shown by the following message:

```
... Server finished processing SQLNET data [krsr_rfs_dsp] oper='Write' flag=67111042
thrd=1 seq=1584 msgid=117468

*** 2014-12-02 08:46:34.761 1459 krsr.c
```

On the primary you can see NSS receiving the ack from RFS and the following message with the timestamp printed above the message:

```
*** 2014-12-02 08:46:34.761 6058 krsu.c

... Client received SQLNET call [kpurcs] response oper='Write' flag=67111042 thrd=1
seq=1584 msgid=117468
```

EXAMPLES

The following shows and example of tracing and how to analyze a long log write that took 163ms:

From the NSS trace file we see that almost all of the 163ms was spent either on the network or within RFS:

```
Client sending SQLNET request data [kpurcs] oper='Write' flag=67111042 thrd=1
seq=1584 msgid=117468

*** 2014-12-02 08:46:34.598 5837 krsu.c

*** 2014-12-02 08:46:34.761 6058 krsu.c

... Client received SQLNET call [kpurcs] response oper='Write' flag=67111042 thrd=1
seq=1584 msgid=117468
```

With event 10046 as well at 16410 enabled on the RFS we see the following for the above msgid:

```
... Server received SQLNET data [krsr_rfs_dsp] oper='Write' flag=67111042 thrd=1
seq=1584 msgid=117468

*** 2014-12-02 08:46:34.598 826 krsr.c

WAIT #0: nam='RFS random i/o' ela= 298 p1=0 p2=0 p3=2147483647 obj#=-1
tim=1417535194598944

WAIT #0: nam='RFS write' ela= 162381 p1=0 p2=0 p3=0 obj#=-1 tim=1417535194761203

... Server finished processing SQLNET data [krsr_rfs_dsp] oper='Write' flag=67111042
thrd=1 seq=1584 msgid=117468

*** 2014-12-02 08:46:34.761 1459 krsr.c
```

So the above elapsed time for RFS write accounts for almost all of the 163ms.

The next example shows time being spent on the network for a 138ms long log writes:

From the NSS trace file we see that almost all of the 138ms was spent either on the network or within RFS:

```
Client sending SQLNET request data [kpurcs] oper='Write' flag=67111042 thrd=2
seq=1311 msgid=124705

*** 2014-12-02 08:46:49.879 5837 krsu.c

*** 2014-12-02 08:46:50.017 6058 krsu.c

... Client received SQLNET call [kpurcs] response oper='Write' flag=67111042 thrd=2
seq=1311 msgid=124705
```

On the RFS side we see it didn't receiving the msgid until almost 170ms has expired:

```
... Server received SQLNET data [krsr_rfs_dsp] oper='Write' flag=67111042 thrd=2
seq=1311 msgid=124705

*** 2014-12-02 08:46:50.016 826 krsr.c

WAIT #0: nam='RFS random i/o' ela= 360 p1=0 p2=0 p3=2147483647 obj#=-1
tim=1417535210017013

WAIT #0: nam='RFS write' ela= 246 p1=0 p2=0 p3=0 obj#=-1 tim=1417535210017052

... Server finished processing SQLNET data [krsr_rfs_dsp] oper='Write' flag=67111042
thrd=2 seq=1311 msgid=124705

*** 2014-12-02 08:46:50.017 1459 krsr.c
```

Data Guard Fast Sync

Fast Sync is a new Data Guard capability available with Oracle Database 12c. Fast Sync enables use of the destination parameter NOAFFIRM which specifies that the standby acknowledge receipt of redo without waiting for the write to the standby redo log file to complete. Fast Sync can improve application response time in a SYNC configuration by removing remote I/O from the total round trip time. It also prevents fluctuations and outliers in standby I/O performance from impacting application response time. Fast Sync can make it practical to increase the distance between the primary and any Data Guard synchronous destination to provide greater geographic protection.

Oracle performance testing on Exadata demonstrated that primary database throughput increased by 4% when Fast Sync was configured. Ironically, the fast I/O on an Exadata system using Exadata Smart Flashlogs results in more modest performance advantages. A more substantial performance advantage is seen for production databases deployed on systems having slower I/O. In a performance test using Oracle Databases on virtual machines deployed on commodity x86 with NAS storage, for example, Fast Sync resulted in a 12% increase in primary database throughput (Figure 4). This is due to the fact that disk I/O is a larger percentage of total round-trip time in the virtual machine example.

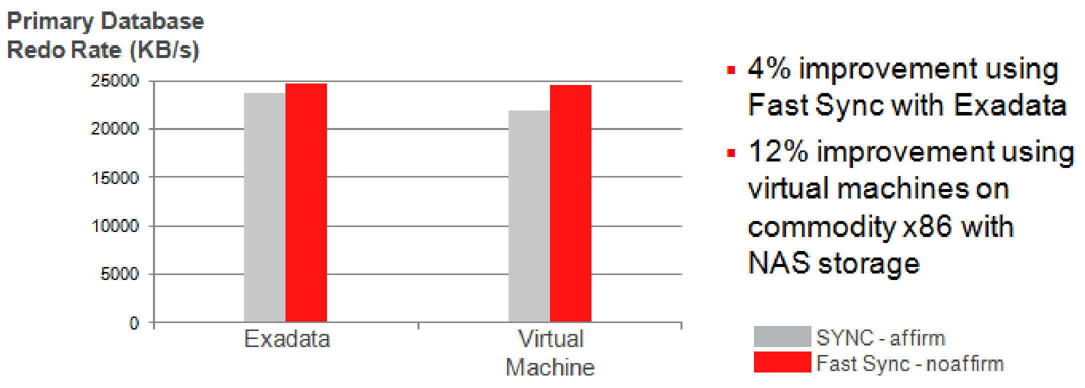

Figure 4: Performance Comparison: FASTSYNC vs. SYNC

Conclusion

Data Guard and Active Data Guard synchronous redo transport provides zero data loss protection during database, cluster, and site outages as well as natural disasters and other events that can impact a widespread geography. Synchronous transport combined with fast database failover (initiated manually by the administrator or automatically using Data Guard Fast-Start Failover[4]) is the only Oracle replication solution that provides zero data loss protection, disaster recovery, and high availability. By definition, any synchronous replication solution will impact production database performance. Data Guard's unique integration with the Oracle Database combined with MAA best practices described in this paper provides the best possible data protection with optimal performance.

4 http://docs.oracle.com/database/121/DGBKR/sofo.htm#i1027843

ORACLE

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

Authors: Andy Steinorth, Michael Smith
Contributors: Joe Meeks, Lawrence To

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

Oracle is committed to developing practices and products that help protect the environment