Case Study: Active-Active Solutions using Oracle Data Guard

Aris Prassinos

Chief Engineer

MorphoTrak

Introduction

This case study discusses MorphoTrak's experiences using Oracle Active Data Guard as a mechanism for deploying Disaster Recovery solutions that make optimal use of the Standby by offloading actual application query workloads. The lessons learned, as well as the recommendations provided, are general enough to apply to most Oracle deployments using Oracle Active Data Guard.

MorphoTrak, SAFRAN Group

Safran Group is a leading technology company headquartered in France, with three core businesses: aerospace, defense and security. Safran has annual revenue of US$14.7 billion and 54,500 employees worldwide.

MorphoTrak is one of the companies within Safran's security group, and it is a leading ISV of biometric and identity management systems sold to government and commercial customers.

MorphoTrak BIS

The MorphoTrak Biometrics Identification Solution (BIS) is an Automated Fingerprint Identification System (AFIS) with additional biometric capabilities such as facial and iris recognition. It has been deployed to more than 100 large customers worldwide and delivers a comprehensive solution for investigation, identification and verification in both the criminal and civil markets.

Current applications include criminal investigation, applicant background checks, biometric visa and passport, border patrol and security, and social services fraud detection.

MorphoTrak BIS Database

MorphoTrak BIS is an online transaction processing (OLTP) application based on Java Enterprise Edition and using a Service Oriented Architecture. It heavily leverages Oracle SecureFiles and XML DB technologies.

The majority of the data stored in its database is unstructured, primarily in the form of Binary Large Objects (LOB) and XML, representing biometrics and their associated metadata. An

average transaction reads or writes about 50 LOBs totaling 1 – 5 MB. In addition, the database is used to store workflows, JMS queues and auditing logs. As a result, the system processes multiple workloads with different characteristics, but the overall read load is much greater than the update load.

Disaster Recovery Objectives

MorphoTrak deploys turnkey fixed cost systems with very well defined throughput and response time requirements that must be contractually met. Some of MorphoTrak's larger customers, especially at the national and state level, also have requirements for Disaster Recovery. One of MorphoTrak's primary goals when designing a Disaster Recovery solution is to fully meet its customer requirements at the lowest overall cost. This cost includes hardware, licensing, software development and maintenance effort, and ongoing system support. Also, data loss must be kept to a minimum and the Recovery Time Objective is measured in minutes.

Typically, the Primary and Disaster Recovery datacenters of MorphoTrak's customers are located in two different cities up to 60 miles apart, with network latency between them of up to 10ms. Clients can transparently connect to either datacenter, effectively experiencing the same latency regardless of where they connect to.

Obviously, it is imperative to design a solution that achieves maximum utilization of the Disaster Recovery site while at the same time minimizing data loss and performance impact. At the same time, the solution needs to be easy to manage without requiring expert staff and extensive monitoring.

Why Oracle Data Guard

Oracle Extended Real Application Clusters (RAC), commonly referred to as a geo-cluster, along with the Oracle Database 11g Automatic Storage Management local read capabilities can be a very attractive solution for deploying active / active solutions, but the above network characteristics and cost considerations would make it impractical to implement in our case.

Multi-master (Active-Active) replication could also provide the maximum utilization of both sites, however, MorphoTrak's customers have very strict data consistency requirements, and the data stored in MorphoTrak BIS cannot be logically partitioned to allow update-anywhere without conflicts. At the same time, it would not be practical or cost effective to have the necessary staff to detect and repair any data conflicts that could occur if a multi-master replication solution were to be used. In addition, a replication solution would be harder to manage and maintain as the application evolves over time.

Oracle Active Data Guard when used in Maximum Availability (synchronous redo transport) mode is a simple and easy to maintain solution that satisfies our Recovery Point and Recovery Time Objectives, while guaranteeing data convergence. It can also satisfy our cost effectiveness goal by offloading a high proportion of our application read workload to the Standby with minimal development effort. To do this we need to be able to automatically route all the application writes to the Primary and load balance the application reads on both Primary and Standby. This is illustrated in Figure 1.

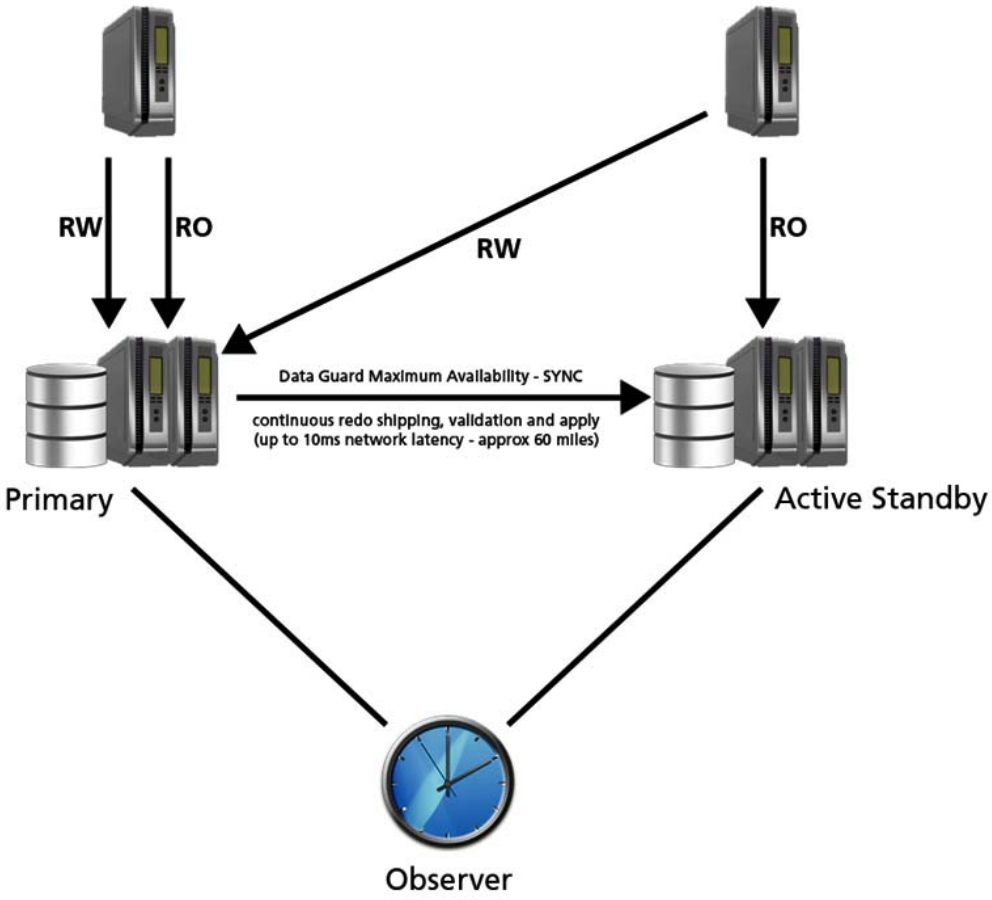

FIGURE 1: HIGH AVAILABILITY / DISASTER RECOVERY – ACTIVE / ACTIVE

Why Not Just Use Database Links

The typical mechanism for offloading queries of packaged applications to the Standby involves using database links and synonyms, along with a combination of Read-Write and Read-Only services and connection pools.

This would not be feasible in our case since our application heavily uses LOBs, which cannot be handled transparently when using database links.

We also found that links would not be totally adequate even for the other parts of our complex OLTP application that do not use LOBs, even if used in conjunction with the STANDBY_MAX_DATA_DELAY parameter, available with Active Data Guard 11g Release 2. First, we need the ability to automatically and quickly redirect a read transaction to the Primary without blocking, if the Standby lag does not meet its latency tolerance. Second, each application method has a different latency tolerance, depending on its business logic. For example, one method may tolerate a 30 second lag while another can only tolerate 5 seconds. Third, certain transactions need to re-read an object from the database soon after it has been updated within the same transaction. In this case, we need to guarantee that we retrieve it from the Primary.

Application Changes – The Best Approach for Our Requirements

To properly route the write and read transactions to the Primary or Standby sites, we define two services for each application, one Read-Write and the other Read-Only. This is similar to the well known scheme described in various Oracle best practice white papers, where the Read-Write service runs on the Primary and the Read-Only service runs on the Standby. What is slightly different in our case is that the Read-Only service runs on both the Primary and Standby. This allows us to load balance the reads to both sites. Each application method, depending on whether it is reading or writing, will then use a database connection that is tied to the appropriate service.

Our approach was to use a Decorator design pattern to implement a wrapper layer for our application methods that read data. This layer contains some very simple logic of calculating the Standby lag to determine whether to use the Read-Write or Read-Only connection pool for a particular transaction and then delegates processing to the underlying method. It also uses application server specific APIs to determine the runtime transactional aspects of a method in order to support repeatable reads.

This approach allowed us to customize the logic for each method based on its latency requirements and its runtime transactional properties. Once this was done, the marginal cost of supporting new application methods was very small. The application methods that update data

do not need a wrapper layer as they always use the Read-Write connection pool. The same is true for reads that cannot tolerate any latency or for very short reads where the cost of calculating the latency would be as high as the read itself.

The pseudo code below describes the logic implemented in the wrapper layer:

```
latency_tolerant_read_method_wrapper() {
   if already participating in a transaction then
      call latency_tolerant_read_method (RW_connection)
   else
      if standby lag within acceptable limit then
         call latency_tolerant_read_method (RO_connection)
      else
         call latency_tolerant_read_method (RW_connection)
}
```

Determining Standby Lag

There are a couple of different ways to calculate the lag of the Standby. One is to convert the current System Change Number (SCN) of both databases to a timestamp and the other is to query the apply_lag column of the v$dataguard_stats table. On Oracle Database 11g Release1 we found that it is actually more efficient to check the current SCN, even if it is more indirect. On the other hand, on Oracle Database 11g Release 2, querying v$dataguard_stats is a better approach due to optimizations made. It also provides the capability to automatically detect if a query would return stale data by using the `STANDBY_MAX_DATA_DELAY` parameter. When using this parameter, a stale query would throw an exception and give the application the opportunity to retry it on the Primary.

Service Startup and Failover

The usual way of starting the Read-Write and Read-Only services on Oracle Database 11g Release 1 is by using a startup trigger. One problem we found with this method, when using it on Oracle RAC, is that some of our applications use singleton services, which run on only one Oracle RAC node in order to benefit from instance affinity. One example is services used for workflow management and JMS. Using startup triggers for singletons on Oracle RAC is

problematic, as a startup trigger cannot relocate a service when its instance fails. To solve this problem we had to resort to using Oracle Fast Application Notification (FAN) callouts.

Oracle Database 11g Release 2 supports role-based services, which eliminate the need for either startup triggers or FAN callouts. As the 11.2 srvctl is now Primary / Standby role-aware, we can use it to define and automatically start any type of service. One problem that still remains with the 11.2 role-based services is that during startup of an Oracle RAC database all singleton services tend to start on the node that comes up first. In our case, both workflow and JMS singletons would start on the same Oracle RAC node, instead of being evenly distributed.

Load Balancing Effectiveness

The load balancing logic described above is very effective but it also comes with some caveats. The main problem is having to do unnecessary query redirects to the Primary due to the inability to determine whether a particular table has actually changed when the Standby has fallen behind. For example, the Standby lag may be 30 seconds, but the actual table we need to query from has not changed during this time.

In addition, in Oracle Database 11g Release 1 the apply lag measurement precision is 3 seconds, which may not be granular enough for methods that require very low latency. Also, for very short reads the overhead of measuring the lag may be larger than the read itself, so for these we skip the calculation and always execute them on the Primary. This overhead has been reduced in Oracle Database 11g Release 2 and the lag measurement precision has been improved to 1 second, so we expect better load balancing results with this release.

Avoiding Split Brain

To keep the downtime to a minimum and avoid split brain in case of failover we are also using Data Guard Fast-Start Failover. Fast-Start Failover includes an Observer process that continuously monitors both primary and standby databases. The Observer will detect any failure of the primary database and automatically transition the standby database to the primary role.

One of the hardest dilemmas we faced was deciding where to place the Observer. The best practice is to place the Observer on a third site, but we were originally tempted to put the Observer on either the Primary or Standby site, to avoid having to manage a third site. We very quickly realized that doing so would be disadvantageous given the rules used by the Observer process to maintain quorum, protect against split brain, and execute automatic failover. While placing the Observer at the Primary or Standby location could be made to work, there would be

cases where we would have had to disable Fast-Start Failover and resort to manual procedures to effect role transitions, especially in the case of a network isolated Primary.

Given the lightweight requirements for a third site it wasn't difficult to identify a satisfactory site (no Oracle Database instance is required and the Observer host may be on a different hardware architecture/O.S. than the primary and standby database). We also use Enterprise Manager Grid Control to monitor the Observer and automatically restart it or fail it over to another host if required; this alleviated any administration concerns.

Client Failover

It is also important to consider how to efficiently stop the query load balancing when the Standby goes down, in order to avoid stalling due to connection timeouts. Oracle Notification Services (ONS) cannot be used to disconnect clients from the Standby since a db_down event is not posted in this case. Setting `SQLnetDef.TCP_CONNTIMEOUT_STR` to a low value would not be adequate either as it would still result in some stalling.

One option is to use a hardware traffic manager such as BigIP which can virtualize the IP address of the Read-Only service. Newer capabilities of application servers such as JBoss HA data sources or WebLogic multi data sources specifically address this problem, as they allow client connections to failover almost instantly in the case of a site going down. Additionally, they can allow query load balancing to automatically restart when the Standby comes up again by reestablishing client connections.

Example Implementation

MorphoTrak recently deployed this solution for the national police of one of its large European customers.

The network latency between their two datacenters can be up to 10ms and the peak database redo rate is 3MB/sec. Each datacenter is using a two-node Oracle RAC Database to provide local high availability and minimize the need for site failover.

The impact on the Primary due to the Maximum Availability Data Guard mode is between 5% and 10% depending on the latency and redo rate. We found this acceptable in return for the guarantee of zero data loss should the primary site fail. The apply lag on the Standby is less than 3 sec, depending on the redo rate. Due to the reliability of the network, the value for Data Guard

NET_TIMEOUT was set to 10 seconds to meet the Recovery Point Objective without affecting performance.

Our load balancing effectiveness using the techniques described above turned out to be nearly perfect, even with the previously mentioned caveats.

Complete site failover can be achieved in as low as two minutes. This is accomplished by setting the Fast-Start Failover threshold to one minute to allow for RAC node eviction and other transitory outages (this eliminates false failovers) and allowing for one more minute for the database and client failover to complete.

Conclusion

Assuming that the application changes described above are feasible, instantaneous failover is not needed, and clients experience similar latency when connecting to either Primary or Standby sites, Oracle Active Data Guard can be a very cost effective Disaster Recovery solution, as it allows offloading actual application query workloads to the Standby. Additionally, this solution can also be used with higher latency networks, by using Maximum Performance (asynchronous redo transport) Data Guard mode, if the performance impact of using Maximum Availability mode is too high and a small amount of data loss can be tolerated.

About the Author: Aris Prassinos is Chief Engineer for MorphoTrak, SAFRAN Group, where he has been instrumental in architecting innovative public safety solutions, deployed by MorphoTrak around the world. Aris is an Oracle ACE Director, a frequent speaker and panelist at database industry conferences, and was also recognized as the "Content Management Architect of the Year" for 2005 by Oracle Magazine.