2020 IEEE 36th International Conference on Data Engineering (ICDE)

Oracle Database In-Memory on Active Data Guard: Real-time Analytics on a Standby Database

Sukhada Pendse[#1], Vasudha Krishnaswamy[#2], Kartik Kulkarni[#3], Yunrui Li[#4], Tirthankar Lahiri[#5], Vivekanandhan Raja[#6],
Jing Zheng[#7], Mahesh Girkar[#8], Akshay Kulkarni[#9]

[#] *Database Server Technologies, Oracle America*
400 Oracle Parkway, Redwood Shores, CA 94065, USA
[1] sukhada.pendse@oracle.com
[2] vasudha.krishnaswamy@oracle.com
[3] kartik.kulkarni@oracle.com
[4] yunrui.li@oracle.com
[5] tirthankar.lahiri@oracle.com
[6] vivekanandhan.raja@oracle.com
[7] jing.zheng@oracle.come
[8] mahesh.girkar@oracle.com
[9] akshay.s.kulkarni@oracle.com

***Abstract*— Oracle Database In-Memory (DBIM) provides orders of magnitude speedup for analytic queries with its highly compressed, transactionally consistent, memory-optimized Column Store. Customers can use Oracle DBIM for making real-time decisions by analyzing vast amounts of data at blazingly fast speeds. Active Data Guard (ADG) is Oracle's comprehensive solution for high-availability and disaster recovery for the Oracle Database. Oracle ADG eliminates the high cost of idle redundancy by allowing reporting applications, ad-hoc queries and data extracts to be offloaded to the synchronized, physical Standby database replicated using Oracle ADG. In Oracle 12.2, we extended the DBIM advantage to Oracle ADG architecture. DBIM-on-ADG significantly boosts the performance of analytic, read-only workloads running on the physical Standby database, while the Primary database continues to process high-speed OLTP workloads. Customers can partition their data across the In-Memory Column Stores on the Primary and Standby databases based on access patterns, and reap the benefits of fault-tolerance as well as workload isolation without compromising on critical performance SLAs. In this paper, we explore and address the key challenges involved in building the DBIM-on-ADG infrastructure, including synchronized maintenance of the In-Memory Column Store on the Standby database, with high-speed OLTP activity continuously modifying data on the Primary database.**

Keywords—OLAP, Standby database, In-Memory Analytics

I. Introduction

Oracle Active Data Guard (ADG) [6] provides active-standby replication for the Oracle Database. The Standby database is a synchronized, physical copy of the Primary database, typically hosted at a remote site. The Primary communicates with the Standby database over a network protocol like TCP/IP. Since the Standby database constitutes an important part of disaster recovery, it typically lags the Primary database by sub-second delays, thereby providing almost real-time, read-only access to the database objects. Sub-second delays are tolerable for a number of reporting applications processing large datasets, like Big Data analytics. Offloading such read-only workloads to the Standby database not only frees up CPU on the Primary (Production) database for OLTP, but also isolates batched reporting overheads from online processing [3]. Figure 1 shows the high-level architecture of Oracle Database deployed with Oracle ADG.

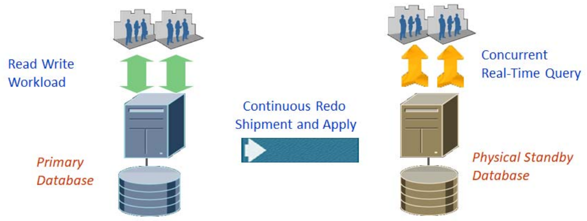


Fig. 1. Oracle Database deployed with Oracle ADG Standby

This paper describes how we extended the DBIM functionality [12] to the Oracle ADG architecture so that analytic workloads that are offloaded to the Standby database can run significantly faster. DBIM-on-ADG architecture is designed to allow for independent scale-out of the Primary as well as Standby databases using Oracle Real Application Clusters (RAC) [9], providing enterprise-scale customers full autonomy in scaling up their OLTP or read-only workloads.

Capacity Expansion Capability: When the In-Memory Column Store (IMCS) is configured for both Primary and Standby databases, the data populated in the IMCS on the two databases could be a completely different set of objects. This technique effectively increases the size of the IMCS. In a typical configuration, customers can create three *services*: Standby-only, Primary-only, and Primary-and-Standby using Oracle's Services Infrastructure [7]. As shown in Figure 2, the latest month of the SALES fact table data is populated in the Primary instance's IMCS, but the entire year's SALES data is populated on the Standby instance for running analytics. The dimension tables can be populated on both instances for efficient join processing. For each partition of SALES data, the customer therefore specifies either the standby or primary service, and for each dimension table, the customer specifies a service that

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00139

includes both Primary and Standby database instances. Customers can thus achieve workload isolation using the IMCS *capacity expansion capability* provided by the DBIM-on-ADG infrastructure, and reap the benefits of faster analytics for workloads run on both instances.

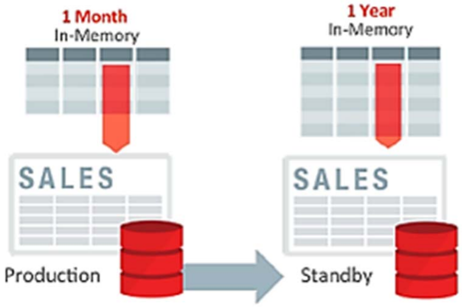


Fig. 2. Example Deployment of DBIM on Primary and Standby instances

Distinction from DBIM infrastructure on the Primary database: The unique replication architecture of Oracle ADG necessitated a redesign of key components of the Oracle DBIM infrastructure, specially the components that populate data in the IMCS and maintain its transactional consistency. A major challenge while designing the DBIM-on-ADG architecture was to avoid compromising the key benefit of ADG – its disaster recoverability. Disaster recoverability is a function of how quickly the Standby database can sync up with the redo logs being pushed by the Primary database. Thus, the DBIM-on-ADG infrastructure is designed to add extremely thin layers of overhead on the ADG architecture. The design performs minimal processing in critical paths and piggybacks on the massive parallelism employed by the Oracle ADG architecture.

The rest of this paper is organized as follows – Section II provides an overview of Oracle ADG and Database In-Memory, Section III describes the DBIM-on-ADG infrastructure and the design of its various components, Section IV provides performance results and Section V concludes with future work.

II. Overview of Oracle ADG and DBIM Architectures

A. *Overview of Oracle ADG architecture*

Oracle ADG provides active-standby replication for the Oracle Database. The replica, known as the Standby database, is a synchronized, physical copy of the Primary database, maintained via Redo Apply (also called Media Recovery) [6].

Redo logs shipped from the Primary database contain redo records, potentially generated by multiple Oracle database instances (with Oracle RAC [9]). A redo record can contain multiple Redo Change Vectors (CVs), with each CV being applicable to a single database block that is identified using the Database Block Address (DBA). All CVs in a redo record are considered to have been generated at the same SCN (System Change Number), which indicates the *database time* at which the changes were made to the database blocks. Each CV is tagged with a transaction identifier. It is key to note here that the SCN associated with a redo record may not represent the database time at which the transaction commits (commitSCN). The commitSCN is, in fact, the SCN associated with a 'commit CV' which is applied to a special block. On the Standby instance, a *Log Merger* process orders the redo records based on their SCN. The SCN-ordered logs can be applied to corresponding database blocks to create a physical copy of the underlying datafiles on the Standby database. Once all CVs up to an SCN value have been applied, the Standby database is considered to have caught up with the Primary database up to that SCN. This is the simplest version of redo apply.

Logs on the Primary database could be generated by multiple concurrent transactions. Serializing the application of the logs to the Standby database can heavily slowdown redo apply which, in turn, increases the lag between the Primary and the Standby databases. This is detrimental to the Standby Database's main purpose – disaster recovery. Hence, redo apply is massively parallelized for Oracle ADG by distributing the SCN-ordered set of CVs amongst *recovery worker* processes based on a hashing scheme. Figure 3 shows the high-level architecture of Parallelized Redo Apply. Each DBA is hashed to a particular recovery worker identifier, so a recovery worker process can independently process the CVs it has been assigned, and apply the CVs to database blocks in the SCN order.

Although parallelization speeds up redo apply, it introduces a potential transactional inconsistency problem. Since recovery workers could be applying the CVs at different rates, it is possible to break the transactional order of changes on the Standby database. In Figure 3, for instance, recovery worker 1 is still applying the CV from SCN 100, while recovery worker N has raced ahead and is applying the CV from SCN 110, which contains a change to the database at a later point in time. In a case where the changes to DBA 7 and DBA 150 are part of the same transaction, the change to DBA 150 should not be visible to queries before the change to DBA 7 is also visible (due to atomicity property of transactions). A centralized coordinator, therefore, needs to establish visibility of the applied changes on the Standby database, which brings us to the concept of QuerySCN on ADG.

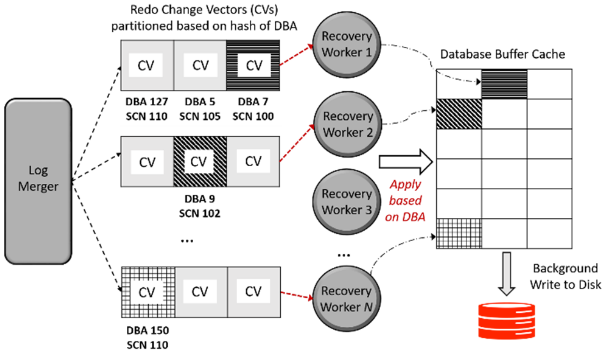


Fig. 3. Parallel Redo Apply/Media Recovery on Oracle ADG

A *recovery coordinator* process tracks the progress of all the recovery worker processes and establishes a consistency point up to which all workers have completed redo apply. This consistency point is exposed as the 'QuerySCN' on ADG. The QuerySCN serves as the Consistent Read (CR) [13] snapshot for queries executed against the Standby database, until a newer consistency point (i.e. a higher QuerySCN) is established by the recovery coordinator. Since the recovery worker processes apply redo at different rates, the QuerySCN on ADG typically leapfrogs, instead of being a stream of consecutive SCNs.

Specific components of the DBIM-on-ADG architecture are positioned strategically in an endeavor to establish the same consistency point for the IMCS. This enables queries run at the QuerySCN to take advantage of the IMCS on the Standby Database. Section III explores these components and their placement in detail.

B. *Overview of Oracle Database In-Memory*

Row-stores are ideal for OLTP workloads, wherein transactions access a small number of rows but several columns in each row, while column stores [4] are suited for analytic workloads that typically access a large number of rows, but only a few columns in each row. Oracle DBIM [12] introduced a dual-format architecture that maintains two copies of the same data – the traditional row-format on-disk and a columnar format in the IMCS. The DBIM Transaction Manager keeps the data in the column store consistent with ongoing transactional activity on the row-store. Oracle DBIM is, thus, equipped to speed up mixed-OLTP workloads that run transaction processing as well as analytic queries.

The data in the IMCS comprises of read-only In-Memory Columnar Units (IMCUs). IMCUs employ techniques like data compression and encoding to efficiently pack the IMCS. The In-Memory Scan Engine [8] takes advantage of techniques like SIMD vector processing, in-memory storage indexes, optimized predicate evaluation and aggregation [11] to speed up analytic queries. Data loading in the IMCS, also known as *Population*, is typically performed as a background activity, and does not affect ongoing transactions and queries. Population establishes a snapshot SCN for each IMCU, and the IMCU is loaded with data consistent as of the snapshot SCN based on Oracle's Consistent Read (CR) model.

Once loaded, data in the IMCUs is synchronized with ongoing transaction processing using specialized techniques. A Snapshot Metadata Unit (SMU) accompanies each IMCU and tracks the validity of the data populated in its corresponding IMCU at various levels of granularity – block level, row level and column level. The In-Memory Scan Engine reconciles the IMCU data with the SMU to ensure that *invalid* or *stale* data is not delivered from the IMCS, but delivered from the database buffer cache (i.e. the row-store). As transactions keep modifying the underlying row-store, higher and higher percentage of the data in an IMCU becomes invalid over time. To offer the best performance for queries scanning the IMCS, the technique of *repopulation* is employed to refresh the data in an IMCU as of a newer snapshot SCN. Like population, repopulation is completely online, transparent to queries and transactions accessing the IMCU and is accomplished as a background activity. A set of heuristics are used to trigger repopulation and tune the repopulation frequency of each IMCU.

In addition to providing consistency guarantees for the data loaded in IMCUs, SMUs provide concurrency control and synchronize operations like repopulation, scans and drop of IMCUs. Figure 4 illustrates a high-level picture of the DBIM architecture with access patterns for different components.

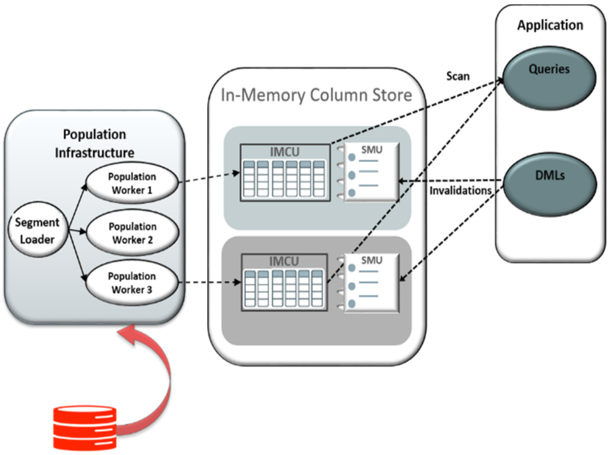


Fig. 4. Oracle DBIM architecture on the Primary database

III. DBIM-ON-ADG INFRASTRUCTURE

The Standby database performs continuous redo apply and establishes consistency points at which queries are guaranteed to yield consistent results. DBIM-on-ADG infrastructure employs specialized components to populate the IMCS and maintain its transactional consistency at these exact consistency points. The major components of the DBIM-on-ADG infrastructure are shown in Figure 5.

DBIM-on-ADG infrastructure interacts with the QuerySCN advancement on the Standby database to capture a *consistent* snapshot SCN to populate the IMCS. Once populated, transactional consistency of the IMCS is maintained by strategically positioned components:

- The *Mining Component* piggybacks on the recovery workers to identify modifications to objects in the IMCS
- The metadata mined by the Mining Component is buffered in the *In-Memory ADG (IM-ADG) Journal*
- The *Invalidation Flush Component* flushes this metadata to SMUs during QuerySCN advancement, thus marking modified data in the IMCUs *invalid*

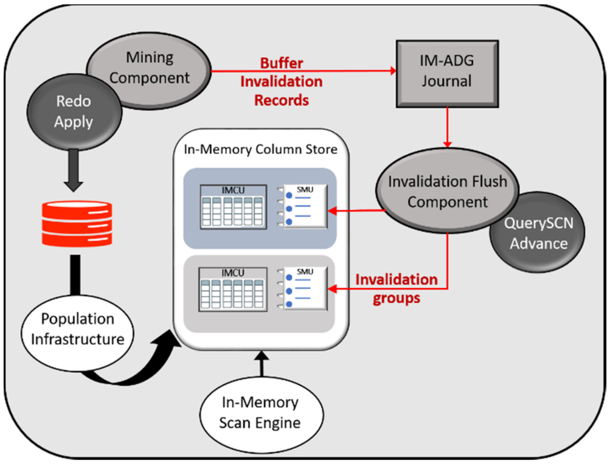


Fig. 5. Components of the DBIM-on-ADG infrastructure

Subtle enhancements to the aforementioned components allow DBIM-on-ADG infrastructure to scale seamlessly across ADG-RAC as well as handle schema changes. In addition, *specialized redo generation* can be employed on the Primary database to ensure consistency of IMCS across ADG instance restart. The following subsections describe the role and design of these components in detail.

A. *Population of the IMCS on ADG*

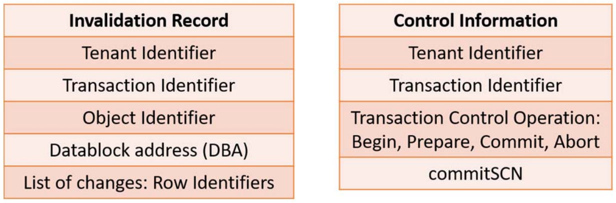
Population of the IMCS on the Standby database, just like on the Primary database, is designed to be completely online and does not block ongoing queries on the IMCS. A *segment loader* process chunks up an object into ranges of data blocks and background population worker processes construct IMCUs for the DBA ranges. Queries and redo apply on the Standby do not stop while population of the IMCS is in progress.

Unlike the Primary database, the Standby database publishes discrete consistency points corresponding to the QuerySCNs. Hence, the snapshot SCN of an IMCU is always the QuerySCN established at the time. This is essential because the population infrastructure may observe an in-flux state of the database, if it picks a snapshot that is not a consistency point. Synchronization is therefore needed between the recovery coordinator publishing a new QuerySCN and the population infrastructure capturing the snapshot SCN. This is achieved through the 'Quiesce Period' on the Standby Database. When the recovery coordinator is about to publish a new QuerySCN, it obtains the 'Quiesce lock' to indicate that *Quiesce Period* has started on the instance. Population infrastructure is not allowed to capture the snapshot SCN for IMCUs during the Quiesce Period. Once the new QuerySCN has been published, the Quiesce Period ends and the population infrastructure can proceed to obtain the snapshot SCN for IMCUs. Background processes in the population infrastructure check whether the Quiesce Period has ended and continue holding the Quiesce lock while capturing the snapshot SCN for an IMCU.

Once the IMCUs are populated on the Standby database instance, the query engine running on the Standby database, which is the same as that on the Primary database, can take advantage of all the optimizations and techniques developed by the In-Memory Scan Engine to scan the IMCS and provide extremely fast query response. The next major task, therefore, is to keep the IMCS on the Standby database consistent as of the consistency point or QuerySCN being published, so that queries see the most up-to-date, consistent results.

B. *Mining Component*

Recovery workers on the Standby database (ADG) apply Redo Change Vectors (CVs) to the underlying data blocks. The DBIM-on-ADG Mining Component piggybacks on the recovery workers to 'sniff' each CV. If the CV modifies an object that is specified to be loaded in the IMCS on the Standby database, a tuple consisting of the Object Identifier, Data Block Identifier (DBA) and the list of changed rows in the data block is noted down in the IM-ADG Journal. Since DBIM-on-ADG works with Oracle Database supporting multi-tenant applications, the tuple also includes tenant information. Each tuple mined by sniffing a CV is termed an 'Invalidation Record' (see Figure 6). Since changes made to the data blocks are guaranteed to be atomic at transaction boundaries, this tuple is tagged with its transaction identifier that is used to fulfil this guarantee.



Invalidation Record
Tenant Identifier
Transaction Identifier
Object Identifier
Datablock address (DBA)
List of changes: Row Identifiers

Control Information
Tenant Identifier
Transaction Identifier
Transaction Control Operation: Begin, Prepare, Commit, Abort
commitSCN

Fig. 6. Information mined by the Mining Component

In addition to mining changes to the data in the IMCS, DBIM-on-ADG protocols need to mine certain control information. Each transaction has a unique commit point, or commitSCN, at which the changes of a transaction are considered atomic, durable and visible to queries per Oracle's Consistent Read model. IMCS on the Standby database needs to adhere to these guarantees as well. Hence, the DBIM-on-ADG Mining Component mines control information about transactions – viz. transaction state changes like Transaction Begin, Prepare, Commit and Abort and the commitSCN associated with each transaction. Invalidation records are associated with this control information using the Transaction Identifier.

It is natural to now ask the question – '*Why can't an invalidation record be flushed to the SMU immediately after it has been constructed*?' After all, the SMU needs to record this information for transactional consistency of the IMCUs. The reason for delaying the flush is two-fold. Firstly, since the population of an IMCU is performed as a background activity, separate from the redo apply, it is possible that the relevant SMU has not been created yet. Secondly, even if the SMU is present, prematurely flushing the invalidation records means exposing a transaction's changes earlier than its commitSCN. While that may seem like erring on the side of caution, special protocols are required to guarantee that the SMU continues to exist and holds on to this invalidation information till the time the QuerySCN on Standby reaches the transaction's commitSCN. Since population and repopulation happens in the background in a completely online manner, it is very difficult to provide such guarantees.

To prevent these cases, DBIM-on-ADG protocols buffer invalidation records in an 'IM-ADG Journal', and only flush them to the SMUs at an optimal point.

C. *IM-ADG Journal to buffer the invalidation records*

The IM-ADG Journal facilitates journaling and buffering of the invalidation records mined by the DBIM-on-ADG Mining Component. The IM-ADG Journal is designed to work in synergy with the massively parallel redo apply, while maintaining the invariant that changes need to be atomic at transaction boundaries.

The core structure of the IM-ADG Journal contains an in-memory hash table mapping a transaction identifier to its invalidation records. The hash table is sized based on the degree of parallelism employed by the ADG architecture, to ensure minimal contention between the recovery worker processes.

However, with very high throughput of transactions on the Primary database, it is possible to see some chaining in the hashbuckets. The resulting hash-chains are protected using a ‘bucket latch’, to synchronize between multiple recovery worker processes operating on the same hashbucket. Figure 7 shows the high-level design of the IM-ADG Journal.

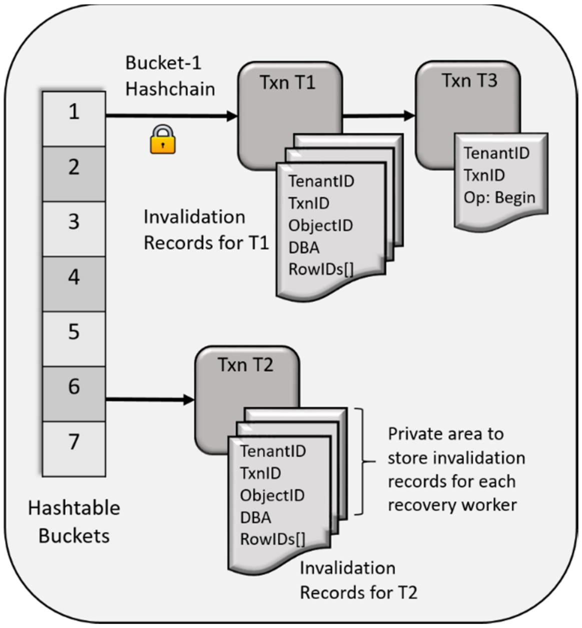


Fig. 7. High-level design of the IM-ADG Journal

Each hashbucket contains hashtable nodes which serve as the anchor for invalidation records from a transaction. This is essential to maintain transaction atomicity guarantees – either all changes of a transaction should be visible to a query, or none should. Once an anchor node is created for a transaction, each recovery worker is provided its own area in the anchor node to buffer the invalidation records it mines. This gets rid of all synchronization needed between multiple recovery workers mining invalidation records for a transaction – which is a common case. Figure 7 shows how mined invalidation records for transactions T1 and T2 are stored in the IM-ADG Journal. T3 currently has no invalidation records, but the anchor node would have been created when the corresponding ‘transaction begin’ (control operation) was mined by some recovery worker.

D. *QuerySCN advancement and Invalidation flush to SMU*

When the Standby database is ready to advance the QuerySCN in order to establish a newer consistency point, the invalidation records gathered in the IM-ADG Journal need to be flushed to the SMUs – if and only if – the transaction that made those changes has commitSCN less than or equal to the new QuerySCN. Since the IM-ADG Journal stores the invalidation records for each transaction separately, this is a simple operation. However, a transaction could have made changes that modify data in different IMCUs, making it essential to map invalidation records to the corresponding SMUs for ensuring a relatively cheap flush operation. The Invalidation Flush Component achieves this by organizing the invalidation records into ‘Invalidation groups’. The recovery coordinator advancing the QuerySCN flushes the invalidation groups to relevant SMUs before publishing the new QuerySCN. Any queries running at the new QuerySCN, thus, find the corresponding data in the IMCU invalid.

While this seems like a straightforward operation, it can introduce significant latency in publishing the new QuerySCN if the recovery coordinator performs this operation alone, in a serial manner. As mentioned in Section IIA, since the Primary database generates logs in a multi-threaded manner, committing thousands of transactions per second, the SCN on the Primary database advances very fast. Hence, the Standby database needs to be able to quickly advance the consistency point to higher and higher QuerySCNs. Any latency in establishing the QuerySCN runs the risk of making the Standby database lag, putting its failover capabilities at risk. Invalidation Flush is, thus, on the critical path and optimizing this operation is of paramount importance.

DBIM-on-ADG infrastructure employs two key techniques to reduce the latency of Invalidation Flush during QuerySCN advancement. First, a helper structure called the ‘IM-ADG Commit Table’ is created to provide quick lookups into the IM-ADG Journal. Second, the recovery workers are repurposed to perform a highly parallelized, cooperative flush operation.

1) IM-ADG Commit Table:

DBIM-on-ADG Mining Component maintains an in-memory, sorted linked list of transaction identifiers and their commitSCN in the IM-ADG Commit Table. When certain control information about a transaction is mined – viz. transaction commit or transaction prepare, a ‘Commit Table node’ is created. The Commit Table node contains the transaction identifier and its commitSCN, and is inserted in the linked list, which is sorted on the commitSCN. In addition, the Commit Table node contains a direct reference to the anchor node in the IM-ADG Journal which hosts the transaction’s invalidation records.

When a new consistency point needs to be established, DBIM-on-ADG Invalidation Flush Component rides on the recovery coordinator process to chop off the Commit Table and create a *Worklink* (see Figure 8). All nodes in the worklink carry transaction identifiers of transactions whose changes need to be ‘flushed’ to the SMUs before the new consistency point can be published. The Invalidation Flush Component achieves this by obtaining one-step access to the IM-ADG Journal anchor node through the worklink. It gathers all invalidation records for each transaction, chunks them up into invalidation groups based on the DBA ranges for IMCUs and flushes them to the respective SMUs.

To address the bottleneck of insertion into a single, sorted linked list by the Mining Component, the IM-ADG Commit Table can be partitioned to create multiple sorted linked lists. A worklink is created for each such sorted list during QuerySCN advancement.

2) Cooperative Flush:

It is easy to see that once the worklink has been created, the flush of invalidation records for different transactions in the worklink can be parallelized. DBIM-on-ADG Invalidation Flush Component uses the recovery workers to aid this process, performing ‘Cooperative Flush’. Recovery workers, in addition to performing the task of redo apply, periodically check if a worklink has been created. If there is a worklink, the recovery workers help the recovery coordinator flush a batch of nodes from the worklink before continuing redo apply.

The recovery coordinator creates the worklink, tracks its progress and publishes the target QuerySCN as the new consistency point once the worklink has been emptied.

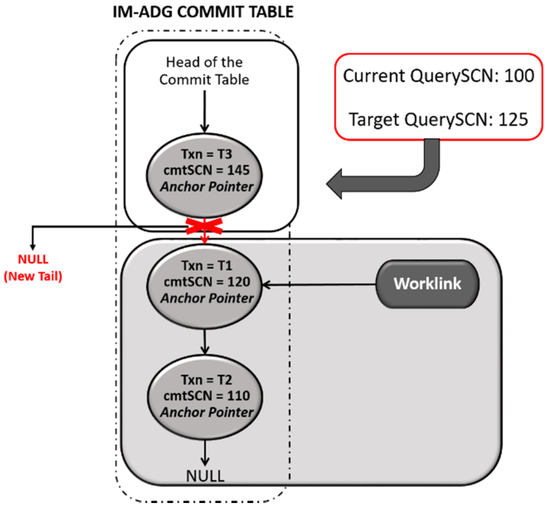


Fig. 8. High-level design of the IM-ADG Commit Table

E. Specialized Redo Generation on the Primary Database

The Primary database is mostly agnostic to the fact that the IMCS even exists on the Standby database and therefore, no overheads are introduced when a transaction generates CVs to modify data on the Primary database. However, there is an exception to this. Special redo generation may be performed on the Primary database in the form of annotating the *Commit Record* of a transaction with a flag indicating whether the transaction modified any object enabled for population into the IMCS. This subsection describes how DBIM-on-ADG infrastructure utilizes this flag.

Redo apply on ADG is completely decoupled from the Primary database. The database administrator can turn on or turn off redo apply on ADG, shut down and restart the Standby database instances at will. ADG protocols maintain enough state to resume recovery in such cases. However, since the IMCS has no persistent footprint other than the underlying row-store objects, DBIM-on-ADG components lose all their state in case of instance restart. It is therefore possible for a transaction to be partially mined in a recovery session, the Standby database instance then shut down and restarted, and the transaction commit information being mined in a later session. If the transaction's commitSCN is beyond the snapshot SCN of an IMCU, the transaction's invalidation records need to be flushed to the respective SMU. The Commit Record of the transaction, therefore, carries a flag to indicate whether any invalidation records are expected for this transaction. If they are, and the IM-ADG Journal has none or only a partial set of invalidation records (which is discovered by a missing 'transaction begin' control information record), the Invalidation Flush Component uses a coarse invalidation procedure to mark all IMCUs for the particular tenant as 'invalid'. Marking an IMCU invalid stops queries from accessing it, till it is repopulated.

While coarse invalidation introduces significant latency, it only occurs when the Standby database instance restarts. Hence, if population of the IMCS is postponed for a short duration after instance restart, we do not expect coarse invalidation at all. It is worth noting that special redo generation is not absolutely essential. DBIM-on-ADG can *pessimistically* assume that each transaction modified some object in the IMCS and trigger coarse invalidation, if a missing 'transaction begin' is discovered. However, it is in the interest of optimum query performance to not trigger coarse invalidation.

F. DBIM-on-ADG with Real Application Cluster (RAC)

Primary and Standby databases can be scaled independently using Oracle Real Application Clusters (RAC). Oracle Database In-Memory scales seamlessly across RAC, with IMCUs distributed across the IMCS on multiple Oracle RAC instances based on a hashing scheme. The mapping of IMCUs to instances is stored in a home-location map [5].

Redo apply on the Standby database is typically limited to a single master instance, known as Single Instance Redo Apply or SIRA. A non-master instance does not perform Redo apply, but hosts a local recovery coordinator process which receives the QuerySCN from the master recovery coordinator and exposes it to queries served by that instance. Hence, the IM-ADG Journal and IM-ADG Commit Table are created only on the master instance. During QuerySCN advancement, DBIM-on-ADG Invalidation Flush Component queries the home-location map and transmits the 'invalidation groups' to the desired instance. The local recovery coordinator on the receiving instance flushes the invalidation groups to SMUs on that instance and acknowledges the same to the master. Since messaging over the network can become a bottleneck, DBIM-on-ADG infrastructure employs batching and pipelined transmission of invalidation groups to reduce the impact of network latency on QuerySCN advancement.

G. Interaction of IMCS with Schema Changes

Oracle Database supports several DDLs at the table, partition, sub-partition and column levels. DDL operations typically modify underlying schema objects. Certain DDL operations in Oracle are only applied at the data dictionary-level and hence, perform no changes to the underlying data blocks for the object. Database In-Memory on Primary database is tightly integrated with these DDL operations. For instance, dropping a column in a table that is populated in the IMCS drops the corresponding column from all IMCUs for the table so that the column cannot be accessed by queries.

DBIM-on-ADG does not enjoy this privilege. DDLs are replayed via redo apply on ADG. DBIM-on-ADG infrastructure therefore introduces *redo markers* in the redo logs in response to DDL operations. Redo markers are similar to redo records but are used to indicate changes to non-persistent objects (which the IMCUs in the IMCS are). Redo markers are mined by the DBIM-on-ADG Mining Component and the information therein buffered in a separate DDL Information Table, similar to the IM-ADG Commit Table. At the time of advancing the QuerySCN, IMCUs for the particular object are dropped, if the definition of the object has changed.

IV. Performance Evaluation

In this section, we will present the key benefits of enabling DBIM on Oracle ADG. The performance benefits of DBIM for OLAP have been extensively evaluated in real-world enterprise workloads. Our performance evaluation experiments focus on demonstrating the advantages of having Database In-Memory with Oracle ADG, while satisfying the important goal of not degrading the redo apply and catch-up capabilities of Oracle ADG which are key for disaster recovery. We will, therefore, look at two aspects of performance – 1) Speeding up analytic workloads on Oracle ADG in the presence of OLTP on the Primary database and 2) Performance of Redo Apply on a DBIM-enabled Standby database with high throughput OLTP running on the Primary database (running in multi-tenant mode on 2-node Oracle RAC).

A. *Analytics workloads on Standby with and without DBIM-on-ADG infrastructure*

This section evaluates the speed-up of analytic workloads on Oracle ADG with OLTP on the Primary Database. In modern business organizations, the ability to combine transactional processing with super-fast on-demand analytics on real time operational data is the key to making the right business decisions. Oracle DBIM is an industry-first dual format database that provides fast in-memory analytic performance, while improving transactional processing. DBIM-on-ADG takes it to the next level by providing isolation and workload partitioning, while speeding up reporting workloads.

We present a synthetic workload running in different modes and the resulting gain in scan response times for ad-hoc queries running full table scans. All the experiments were carried out on Oracle Exadata Database Machine [10] which is a state-of-the-art SMP server and storage cluster system.

The setup includes a synthetic OLTAP workload that simulates an insert/update workload interspersed with queries. The test consists of a wide table with 6M rows, and 101 columns (1 identity column, 50 number columns and 50 varchar2 columns) with an index on the identity column. The hardware setup was a 2x Intel Xeon E5-2690 @ 2.90GHz, 8-core processor with 256GB of DRAM, of which only 60GB was used for the in-memory pool. The test was run for 1 hour with a target throughput of 4000 ops/sec. The percentage of DMLs and analytic queries in the workload was tunable. We demonstrate performance improvements for ad-hoc queries using full-table scans run on the Standby database while the Primary continues to process a workload with different mixes of DML operations. We use metrics such as query response time and CPU usage to show the capabilities of the DBIM-on-ADG infrastructure. An important part of the setup is ensuring that the Oracle database buffer cache is sized appropriately to avoid any physical I/O.

Table 1 shows two example queries being executed on the Standby database. These queries are forced to go to the IMCS by not constructing analytic indexes on any column. The queries thus showcase raw performance of IMCS and the In-Memory Scan Engine with optimized predicate evaluation and without any added aggregation benefits. In all these workloads, DBIM-on-ADG infrastructure ensures that the IMCS is maintained transactionally consistent as the QuerySCN advances.

Table 1. Sample queries in the analytics workload

ID	Description	SQL
Q1	Scan, filter a numeric column that may have been updated	SELECT * FROM C101_6P1M_HASH WHERE n1 = :1
Q2	Scan, filter a varchar column that may have been updated	SELECT * FROM C101_6P1M_HASH WHERE c1 = :2

It is key to note that the desired throughput of 4000 ops/s cannot be sustained without DBIM. There is significant backpressure since the setup uses the same set of threads for issuing DMLs on the Primary and queries on the Standby database. This causes the throughput to fall. Dedicated threads can instead be used to maintain the throughput for DMLs.

1) *Update-only workload*

Update-only workload in the synthetic OLTAP configuration introduces 4000 ops/s with 1% scan ops/s (40 scans/sec) running on the Standby database while 70% updates (2800 updates/sec) and 29% fetch operations via the index are being executed on the Primary Database instance. We compare the response time of the queries Q1, Q2 on the Standby Database with and without DBIM-on-ADG. Figure 9 shows that the response time has improved by almost 100x for the sample queries.

With faster scans, the Standby not only becomes a viable alternative to isolate the workload, but also reduces CPU usage on the Primary. With Update-only workloads, the CPU usage on the Primary Database reduces from 11.7% if all operations are run on the Primary to 4.7% when scans are offloaded to the Standby Database. The Standby Database CPU increases from 2% to 17% and the asymmetric increase is due to its architectural difference from the Primary Database.

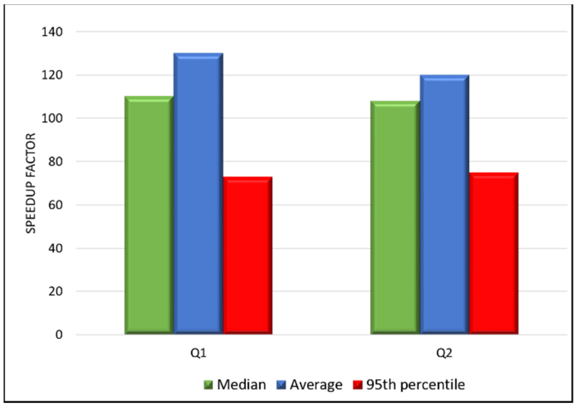


Fig. 9. Speedup in median, average and 95th%ile of query response times of Q1, Q2 with Update-only workload

2) *Update+Insert workload*

Update + Insert workload maintains table-scans at 1% on the Standby Database and the throughput at 4000 ops/s. It executes

25% inserts, 40% updates against the Primary database, with the remaining operations being index-based fetch. Figure 10 compares the response times for Q1, Q2 on the Standby database without and with DBIM-on-ADG.

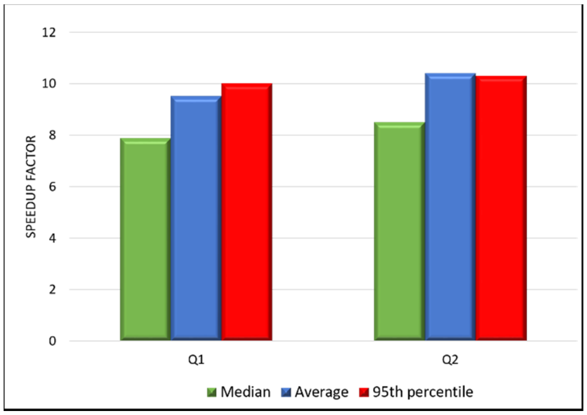


Fig. 10. Speedup in median, average and 95th%ile of query response times of Q1, Q2 with Update+Insert workload

With the introduction of DBIM-on-ADG, the Standby database performs significantly better, as expected, and the response time goes down by almost 10x. With inserts, the size of the table increases, but the query response times are only for the initial table data. Note that with inserts, population infrastructure has to utilize much more CPU in order to populate the newly inserted data into the IMCS. Highly concurrent invalidation and population activity on the *edge IMCU* corresponding to the new inserts leads to a limited performance benefit of the IMCS.

B. Comparison of Read-only Analytic Workload on Primary and Standby Databases

This experiment highlights that the Primary and the Standby databases perform equally well when a scan-only workload – i.e. a workload with no DMLs, is run separately on the Primary and the Standby database. This implies that scans for a subset of data (e.g. a partition) that has no DML activity can be seamlessly offloaded to the Standby, completely transparent to the end-user.

The scan-only workload uses the same synthetic OLTAP setup as subsection IVA, but performs 4000 ops/sec with 25% ad-hoc queries running full-table scans (1000 scans/sec) and 75% fetch queries that access the index. Table 2 compares the response time for Q1 on the Primary and Standby database with DBIM enabled on both.

TABLE 2. RESPONSE TIME FOR Q1 WITH SCAN-ONLY WORKLOAD ON PRIMARY AND STANDBY DATABASES WITH DBIM

	Median (ms)	Average (ms)	95th percentile (ms)
Primary	4.25	4.31	4.55
Standby	4.30	4.36	4.6

Furthermore, there is a direct transfer of CPU usage from the Primary to the Standby database instance – while Primary's CPU usage reduces from 8% to 0.5%, the Standby CPU increases from 0.3% to 7.9% when the scans are executed against the Standby database.

C. Performance of Redo Apply on the Standby Database

In this experiment, we show that the DBIM-on-ADG feature does not significantly affect Redo Apply on the Standby database. The rate of QuerySCN advancement is only slightly affected due to the Invalidation Flush as discussed in Section III. The workload used is a high-throughput transactions workload containing short, medium and long-running transaction mix run on the Primary database running with Oracle multi-tenant. The Primary and Standby databases are configured with DRAM of size 120 GB.

The plots in Figure 11 show the progress of the redo log being archived on the Primary database running with two Oracle RAC Instances (pri_log, pri_log2 in the figure) over a period of two hours. The archived redo is shipped to the Standby database and the progress of the redo log apply on the Standby database RAC instances 1 and 2 with the DBIM-on-ADG feature enabled is shown in the figure (std_log1, std_log2). It is clear that the log catchup is almost instantaneous and the Standby database has minimal lag, even in the presence of the overheads introduced by the DBIM-on-ADG infrastructure.

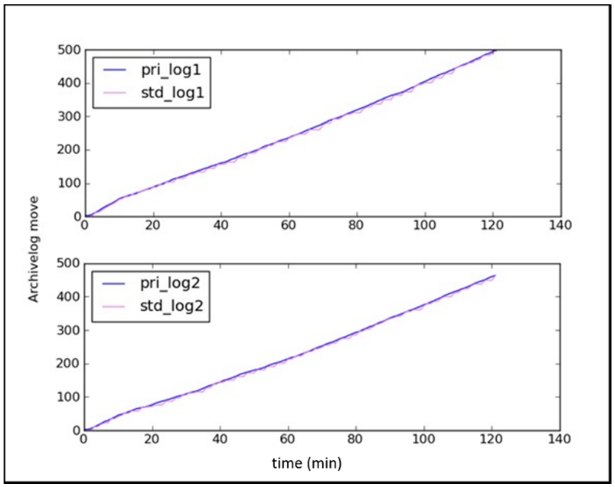


Fig. 11. Log advancement on Primary and Standby instances with Oracle ADG RAC

V. CONCLUSIONS AND FUTURE WORK

Oracle Active Data Guard has a unique architecture that provides for query execution on a Standby database, while serving as a disaster recovery solution. DBIM-on-ADG infrastructure enables the queries executed on the Standby Database to avail the benefits of DBIM, thus improving the response time of certain queries by orders of magnitude. DBIM-on-ADG leverages the highly parallelized infrastructure of ADG Recovery to synchronize the In-Memory Column Store on the Standby database with ongoing transactional activity on the Primary database, while ensuring that the Standby database remains committed to its goal of disaster recoverability. With Database In-Memory functionality extended to the Standby database, customers can get the best of both worlds by isolating their read-write and read-only workloads on the Primary and Standby Databases while continuing to perform faster analytics on both workloads.

Enabling DBIM on the Standby database has opened it up to a plethora of features introduced by DBIM. In-Memory Expressions [1] are now supported on the Standby database and provide even faster performance for complex, analytical expressions used in reporting queries, including JSON processing. In-Memory Join Groups can also be created for the Standby database to make join processing faster. Data from external sources like Hadoop can be enabled for population in the IMCS using the In-Memory External Tables feature [7]. Novel formats and techniques used by DBIM like in-memory storage indexes, aggregation push-down are extended seamlessly to ADG, thus, truly empowering the Standby database for real-time analytics processing.

DBIM, introduced in 2014, has grown into a large ecosystem of its own. Supporting the key features introduced by Database In-Memory with the DBIM-on-ADG infrastructure continues to be an active area of investigation for our team. With Multi Instance Redo Apply (MIRA) [2], ADG can scale-out redo apply to multiple instances with Oracle RAC, providing faster log advancement on the Standby Database. Enhancing the DBIM-on-ADG infrastructure to support MIRA is very important in order to avail the performance benefits for reporting queries on the Standby Database without compromising on the goals of MIRA.

Acknowledgment

We acknowledge the guidance, support and commitment of the In-Memory and Transactions teams, Recovery team, Functional and Stress testing teams in making this project successful. We thank the Performance testing team for their help with the performance evaluation of the DBIM-on-ADG architecture.

References

[1] Aurosish Mishra, et al. "Accelerating analytics with dynamic in-memory expressions," In Proceedings of the VLDB Endowment. 9, 13 (September 2016), 1437-1448.

[2] Best Practices for Data Guard and Active Data Guard Redo Apply, an Oracle White Paper, 2018

[3] Hasso Plattner, "The impact of columnar in-memory databases on enterprise systems: implications of eliminating transaction-maintained aggregates," In Proceedings of the VLDB Endowment. 7, 13 (August 2014), 1722-1729.

[4] Mike Stonebraker, et al. "C-store: a column-oriented DBMS," In Proceedings of the 31st international conference on Very large data bases (VLDB '05). VLDB Endowment 553-564.

[5] Niloy Mukherjee, et al. "Distributed architecture of Oracle database in-memory" In Proceedings of the VLDB Endowment. 8, 12 (August 2015), 1630-1641.

[6] Oracle Data Guard 11g Data Protection and Availability for Oracle Database Technical White Paper, October 2011

[7] Oracle Database In-memory with Oracle Database 19c, an Oracle White Paper, February 2019

[8] Oracle Database 12c: When to Use Oracle Database In-Memory, an Oracle White Paper, March 2015

[9] Oracle Database 12c Release 2, Oracle Real Application Clusters White Paper, March 2017

[10] R. Greenwald, M. Bhuller, R. Stackowiak, and M. Alam, "Achieving Extreme Performance with Oracle Exadata," McGraw-Hill, 2011

[11] Shasank Chavan, Albert Hopeman, Sangho Lee, Dennis Lui, Aijt Mylavarapu and Ekrem Soylemez. "Accelerating Joins and Aggregations on the Oracle In-Memory Database." 2018 IEEE 34th International Conference on Data Engineering (ICDE) (2018): 1441-1452.

[12] T. Lahiri et al., "Oracle Database In-Memory: A dual format in-memory database," 2015 IEEE 31st International Conference on Data Engineering, Seoul, 2015, pp. 1253-1258.

[13] W. Bridge, A. Joshi, M. Keihl, T. Lahiri, J. Loaiza, and N. Macnaughton, "The Oracle Universal Server Buffer Manager," In Proceedings of the VLDB '97, pp. 590-594, 1997.