International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

Real-Time Fault-Tolerant Analytics using Distributed Database in-Memory

Shaik Shoyab & V.Sriharsha
[1] *PG Scholar, Department of CSE, PACE Institute of Technology and Sciences,Vallur, Prakasam,, Andhrapradesh, India*
[2] *Associate Professor, Department of CSE, PACE Institute of Technology and Sciences,Vallur, Prakasam, Andhrapradesh, India*

***Abstract*—** *Modern data management systems are required to address new breeds of OLTAP applications. These applications demand real time analytical insights over massive data volumes not only on dedicated data warehouses but also on live mainstream production environments where data gets continuously ingested and modified. Oracle introduced the Database In-memory Option (DBIM) in 2014 as a unique dual row and column format architecture aimed to address the emerging space of mixed OLTAP applications along with traditional OLAP workloads. The architecture allows both the row format and the column format to be maintained simultaneously with strict transactional consistency. While the row format is persisted in underlying storage, the column format is maintained purely in-memory without incurring additional logging overheads in OLTP.*

Maintenance of columnar data purely in memory creates the need for distributed data management architectures. Performance of analytics incurs severe regressions in single server architectures during server failures as it takes non-trivial time to recover and rebuild terabytes of in-memory columnar format. A distributed and distribution aware architecture therefore becomes necessary to provide real time high availability of the columnar format for glitch-free in-memory analytic query execution across server failures and additions, besides providing scale out of capacity and compute to address real time throughput requirements over large volumes of inmemory data. In this paper, we will present the high availability aspects of the distributed architecture of Oracle DBIM that includes extremely scaled out application transparent column format duplication mechanism, distributed query execution on duplicated in-memory columnar format, and several scenarios of fault tolerant analytic query execution across the in-memory column format at various stages of redistribution of columnar data during cluster topology changes.

Keywords—real-time analytics, OLTAP, Oracle Database Inmemory, distributed architecture, high availability, distributed in-memory fault tolerant analytics

I.INTRODUCTION

Over the last few years, the capacity to price ratios of DRAM has been observed to increase manifolds [1]. It is therefore getting cheaper to place more and more data as close as possible to compute. In parallel, as data ingestion sources and volumes keep on exploding, new breeds of mixed OLTAP applications have emerged [2]. These new applications demand real time analytical insights over massive data volumes as data gets ingested and modified in live mainstream production environments. These applications and workloads are very different from traditional OLAP practices [3] where data has to be first curated into dedicated static data warehouses and analytics workloads are run on static data.

Oracle introduced the Database In-memory Option in 2014 as a unique dual format architecture aimed to address these emerging mixed OLTAP applications along with traditional OLAP workloads [4][5] . The unique architecture allows both row format and column format to be maintained at an Oracle object level (Fig. 1). Both formats are simultaneously active and are not mutually exclusive. Strict transactional consistency is guaranteed between the formats in real time. While the row format is persisted in underlying storage and gets logged for recovery purposes, the column format is maintained purely in memory without incurring logging overheads. The in-memory columnar format provides breakthrough performance for analytics while the row format handles OLTP workloads. Strict consistency between the formats alleviates the need to maintain and synchronize sets of auxiliary analytic indexes therefore improving OLTP as well.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

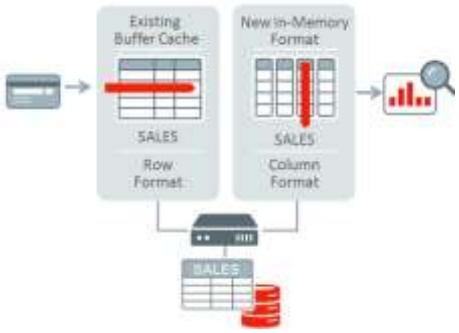

Fig. 1.Dual row and column format for OLAP and OLTAP workloads

compared to persistent storage. Main memory volumes are increasing but data ingestion volumes are increasing at much higher rates. Scale out in terms of both memory and compute resources becomes necessary to meet the required real time throughput requirements over large volumes of data in memory. It can be argued that scale up within a single server may be sufficient for most workloads [6], but single server based architectures neither provide real time availability of the main memory data nor provide fault tolerance in execution on the same. Ironically, real time availability and fault tolerant query execution are some of the most relevant requirements for pure no-logging main-memory architectures. Therefore a distributed high available architecture becomes a must.

The overall distributed architecture of Oracle Database in Memory and its scale-out aspects are highlighted in [5][7]. Besides providing distribution and query execution scale out across a cluster of servers, the distributed architecture has been designed to provide real-time duplication of the inmemory columnar format for high availability along with fault tolerant query execution across server failures. The architecture also ensures minimal recovery impact on the columnar format across cluster topology changes through efficient rebalancing mechanisms and glitch-free in-memory query execution during rebalancing. In this paper, we will primarily focus on these aspects of the architecture.

The main sections of the paper are organized as follows. Section II presents a quick overview of the distributed architecture. Section III introduces the set of duplication options provided by the architecture. Section IV presents a detailed overview of the column format duplication mechanism. Section V describes the fault tolerant query execution mechanism and details the mechanisms for providing glitch free query execution on several cluster topology change scenarios, such as exit of a server, during asynchronous scaled out redistribution of in-memory data across remaining servers, and subsequent redistribution of inmemory data when the server gets added back again in the cluster topology. Section VI presents a set performance evaluation experiments to validate the availability and fault tolerance aspects of the architecture.

II.DISTRIBUTED DBIM–A QUICKREVIEW

This section presents a quick review of the distributed architecture of the Oracle Database In-memory. The building block of the architecture is an In-memory Compression Unit (IMCU) [4], which serves as the smallest unit of distribution, duplication, and distributed access of the columnar format across a cluster of servers. Each IMCU is a columnarized representation 'populated' from a substantial set of rows of the RDBMS object persisted in Oracle Data Blocks [8] (Fig. 2). It contains contiguous runs of columns (Column Compression Units), where each run can be compressed using different compression levels. Columnar data within an IMCU is a readonly snapshot consistent as of a point in time; subsequent changes in the underlying data blocks are tracked by an accompanying Snapshot Metadata Unit (SMU) [4]. As more and more changes accumulate in the SMU, the IMCU undergoes heuristics based fully online repopulation mechanism that results in a new clean version.

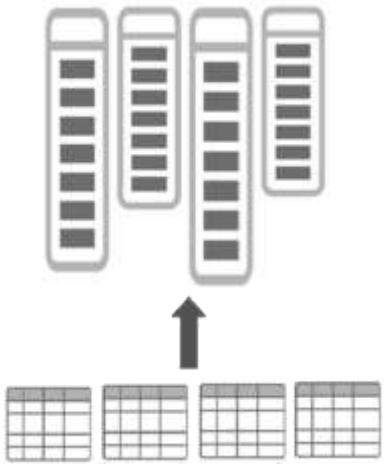

Fig. 2.In-memory Compression Unit (IMCU) populated from Oracle RDBMS Data Blocks

The IMCU format inherits all the compute and capacity utilization advantages of columnar format. Several data processing optimizations [4], such as vector processing operations, predicate evaluation push down, bloom filter push down, in-memory storage indexes for data pruning, etc. have been implemented on top of the IMCU format. These optimizations serve as the backbone for breakthrough analytics performance.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

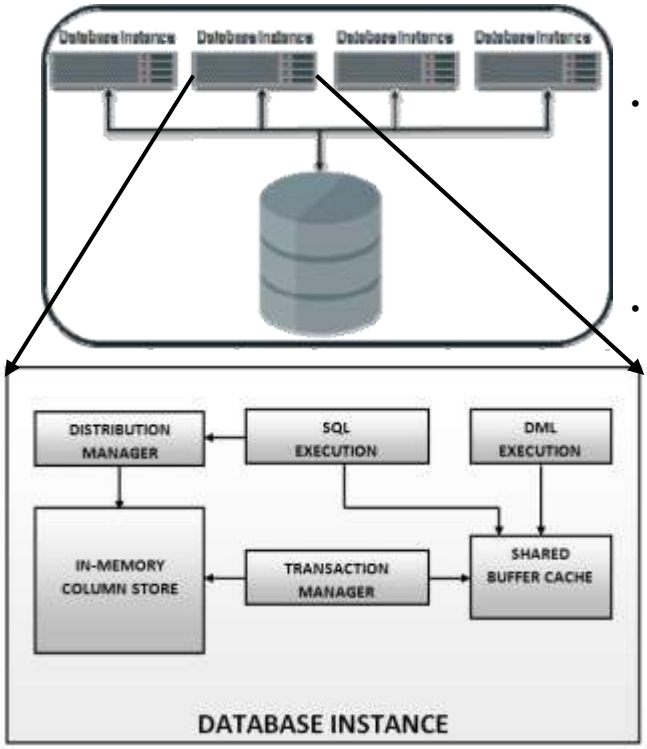

Fig. 3.Distributed Oracle Database In-memory Architecture

The distributed architecture employs Oracle Real Application Cluster (RAC) configurations [8] that allows a cluster of RDBMS servers (instances) to operate on an Oracle database persisted in underlying storage, abstracting the database as a single entity (Fig. 3). Fig. 3 illustrates the components most relevant to the distributed architecture.

- The shared buffer cache [9] is a collective cache of Oracle data blocks across the cluster. It is equipped with a Cache Fusion service that synchronizes access of the data blocks across the cluster.

- The In-memory Column Store (Fig. 4) [7] is the new inmemory area that hosts the IMCUs across the cluster. It is a shared-nothing container of IMCUs within an instance. An in-memory object is defined as a collection of IMCUs across all in-memory column stores within a cluster that have been populated from the data blocks of the same underlying RDBMS object. Therefore, there is one-to-one correspondence between an in-memory object and an RDBMS object (Oracle table/partition/subpartition) on which the in-memory option is enabled. Each instancespecific In-memory Column Store is equipped with a globally-consistent yet fully local In-memory Home Location service that provides for seamless interfacing of the column format with the row format based transaction manager and SQL execution engines.

- The transaction manager [7] ensures strict real-time consistency between the data blocks and their corresponding IMCUs. It also ensures that block modifications due to on-line transaction processing operations (OLTP) are propagated to the appropriate IMCUs across the cluster.

- The distribution manager [7], as the name suggests, handles all the distribution, duplication, availability, and the fault tolerance aspects of the architecture. It also provides access awareness to the transaction manager and the SQL execution engine.

The parallel SQL execution engine [10] employs the buffer cache for OLTP-style queries that benefit from index based accesses. The in-memory column store is employed for all query workloads except indexed based accesses.

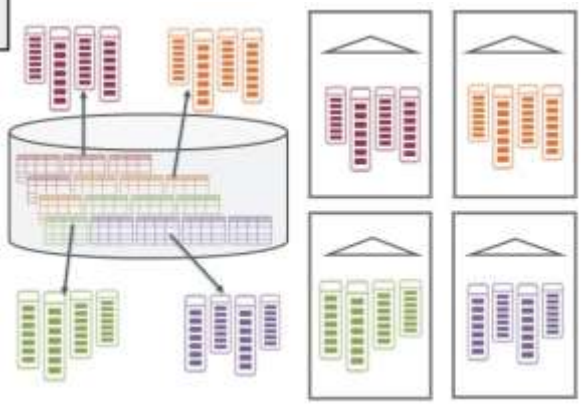

Fig. 4.In-memory Column Store with IMCUs distributed across 4 instances

III. DUPLICATIONDURINGDISTRIBUTION

This section presents a detailed overview of the schemes and mechanisms associated with in-memory object duplication across a cluster of servers. By default, Oracle DBIM does not duplicate IMCUs, i.e., there exists only one IMCU across the cluster for a given set of Oracle data blocks, as illustrated in Fig. 4. However, the architecture provides several duplication options to users for real-time high availability of in-memory objects and fault tolerant in-memory query execution.

A.Duplication Options

Oracle DBIM provides two duplication schemes, namely, DUPLICATE or 1-safe duplication, and DUPLICATE ALL. The user can specify one of these two options while enabling the in-memory option for an RDBMS object. In case of

DUPLICATE, the same set of underlying Oracle data blocks are represented by two IMCU copies distributed in two different instances in the cluster (Fig. 5).

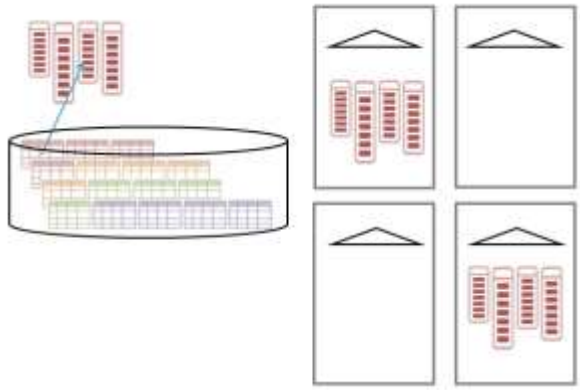

Fig. 5.IMCU distributed with DUPLICATE option in two instances

In case of DUPLICATE ALL, each instance in the cluster hosts the same copy of the IMCU populated from the same set of underlying Oracle data blocks (Fig. 6).

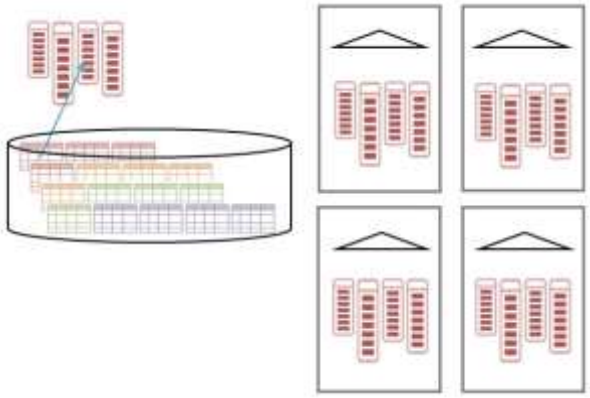

Fig. 6.IMCU distributed with DUPLICATE ALL option in all instances

For the remainder of this paper, we will just focus on the methods and mechanism related to the DUPLICATE option, using the same hypothetical RDBMS object and a cluster of 4 RDBMS server instances (Fig. 4).

IV.DUPLICATION MECHANISM

The duplication mechanism is initiated on detection of missing representation of data blocks of in-memory enabled Oracle RDBMS objects in the global column store across the cluster. The detection is performed either during a scan by a user-issued query or by a dedicated background monitor process executing on each database instance. The duplication mechanism is a three phase mechanism. It consists of a very brief phase of serialization and duplication context generation for consensus, followed by a fully scaled out, load balanced, and application transparent duplication of IMCUs across the cluster, and a final broadcast of duplication completion timestamp across the cluster.

A.Serialization and Consensus Broadcast

The serialization phase consists of the following steps, namely, 1) Master Selection, 2) Duplication Context Generation for Consensus, 3) Acknowledgement of Acceptance, and 4) Master Downgrade.

Master election becomes necessary to serialize the distribution and duplication of an RDBMS object across the cluster, thereby preventing concurrent duplication of the same object at the same time. Master selection is based on competing for a global object distribution lock exclusively in no-wait mode (Fig. 7).

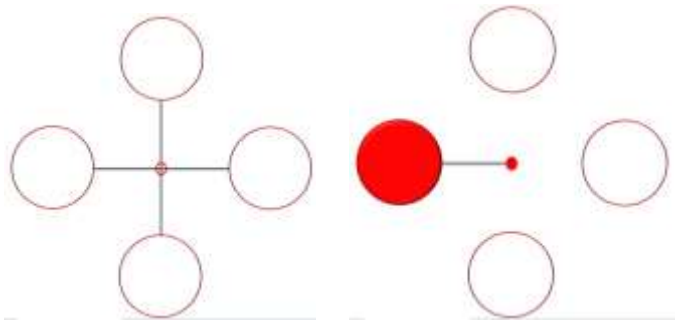

Fig. 7.Leader instance election

The instance acquiring this lock gets selected as the 'master' for coordinating the duplication of the object. The lock is a completely non-blocking lock, i.e., it does not block concurrent OLTP operations or concurrent queries on the object. The lock is taken at an object level, which implies that multiple 'masters' can co-exist in the cluster to coordinate concurrent duplication of multiple RDBMS objects.

Once the 'master' instance gets selected, it generates a very minimal distribution and duplication context for consensus on the set of data blocks that are required to be columnarized and duplicated across the global column store. The consensus context is a payload of a few hundred bytes, which remains constant irrespective of the size of the object to be duplicated or the size of the cluster. Once the context gets generated, the 'master' broadcasts the payload to the rest of the 'inactive' instances and waits for acknowledgement from each of these instances (Fig. 8). The constant size of the payload ensures minimal

cross-instance communication overheads across a large cluster of RDBMS instances.

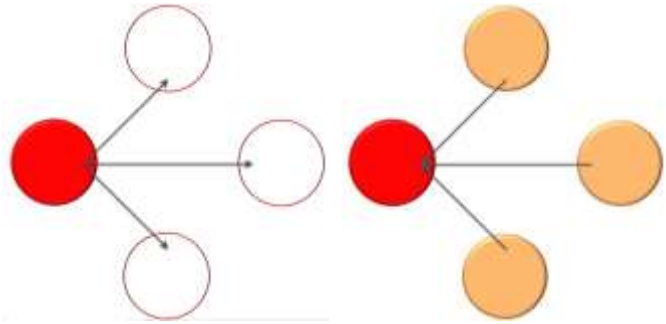

Fig. 8.Payload broadcast, acceptance, and acknowledgement

On receiving the payload from the 'master', the rest of the 'inactive' instances 'accept' the payload. Each of these instance en-queue requests for shared access on the same object distribution lock and send 'acknowledgements' back to the 'master'. At this point, these instances have to wait on the lock as the 'master' has exclusive access on the same. On receiving 'acknowledgements' from each of the 'inactive' instances, the 'master' atomically downgrades its access on the object distribution lock from exclusive to shared and attains the role of a 'follower' (Fig. 9).

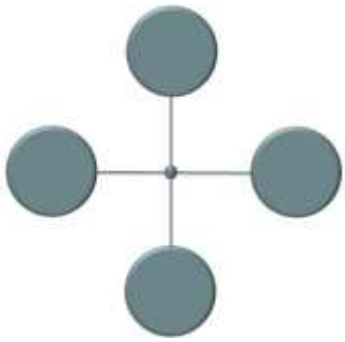

Fig. 9.Change of instance roles to 'followers'

Once the downgrade takes place, the rest of the 'inactive' instances acquire shared access on the lock and get upgraded as 'follower' instances. At this point, each of the 'follower' instances proceeds with the decentralized IMCU duplication phase independently of each other. Until all these instances release access on the shared lock, no instance can be assigned as the 'master' for duplicating the same object.

B.Distributed Duplication of IMCUs

Once the decentralized phase commences, each 'follower' instance executes the following steps, namely, 1) Attaining Distributed Consensus on IMCU Contexts, 2) Attaining Distributed Consensus on Multiple Home Locations for IMCU Contexts, and 3) IMCU Population and Home Location Directory Registration. These steps are executed in same order by each 'follower' instance, but with nplete asynchrony.

commencement of the 'decentralized' phase, the ower instance 'decodes' the minimal payload eived from the 'master' to generate a set of bally consistent IMCU contexts to be duplicated oss the cluster. A 'unique' mechanism is used by h 'follower' instance to determine the globally isistent set of IMCU contexts, irrespective of the istantly changing underlying RDBMS object due to OLTP activity. Each instance therefore achieves distributed agreement [11] on the set of IMCUs with its peer instances, but without incurring peer-to-peer communication (Fig. 10).

Fig. 10.4 IMCU contexts generated with distributed consensus without peerpeer communication

Once the set of IMCU contexts have been determined by a 'follower' instance, the instance needs to assign two globally consistent 'home location' instances for each IMCU context. The 'home location' instance for an IMCU context serves as the dedicated instance where the IMCU gets physically populated in the local in-memory column store. Similar to the first step, each instance has to come up with the same 'home location' instances for the same IMCU context, to achieve distributed agreement with its peers, but without incurring peer-to-peer communication overheads.

The globally consistent assignment of 'home locations' is based on a variant of consistent hashing mechanism called Highest Random Weight (HRW) hashing or Rendezvous hashing [12]. Given an object O and a set of 'n' active sites, the scheme assigns a random weight for each site based on a key derived from object O. The site with the highest weight gets chosen as the 'home location' for the object O. To achieve consistent multiple 'home locations', two sites per IMCU context are selected; the site with the highest weight is elected as the primary home location, and the site with the lowest weight is elected as the secondary home location.

On execution of the above steps, each instance comes up with the same IMCU context assignment matrix (Fig. 11). The strictly consistent IMCU context assignment matrix could have been

generated by a purely centralized approach where the 'master' would have generated the contexts and communicated the same with all instances or each instance would have communicated the contexts with one-another across the cluster. However, in a mainstream production environment with hundreds of gigabyte-to-terabyte sized objects being duplicated at the same time along with concurrent OLTP and OLAP activities, cross-instance communication of millions of IMCU contexts across the cluster would have choked the network causing scale-out throughput bottlenecks. On the other hand, instances could not have arrived with strictly consistent set of IMCU contexts if the mechanism had been purely decentralized.

IMCU Context	IMCU Boundaries	Home Locations Assignments
IMCU 1	<E1, E2'>	[Inst A] [Inst C]
IMCU 2	<E2'', E3'>	[Inst B] [Inst D]
IMCU 3	<E3'', E4'>	[Inst C] [Inst A]
IMCU 4	<E4''>	[Inst D] [Inst B]

Fig. 11.Home location assignment matrix generated with distributed consensus without peer-peer communication

Once the IMCU context assignment matrix has been generated by a 'follower' instance, the contexts are compiled into two disjoint sets. The first set of contexts includes the ones where either the primary or the secondary home locations match the id of the executing 'follower' instance. The second set includes the rest of the contexts where neither the primary nor the secondary home locations match the id of the executing instance. The IMCU contexts in the first set are executed in parallel by dedicated background processes. These IMCUs are physically populated from the underlying data blocks defined in the IMCU context into the local in-memory column store. Physical population includes storage I/O, data transformation, compression (if applicable), allocation of instance-local shared memory for the set of Columnar Compression Units within the IMCU, and memory-copy of the IMCU in the allocated memory from the local column store. Physical population of the IMCU is followed by registration of the IMCU context with the column-store home location service. A home location entry (HLE) gets created with the IMCU memory offset and other metadata (described in the subsequent section). Once the entry is inserted and committed in the home location service index, the IMCU context becomes visible to the transaction manager and the SQL execution engines.

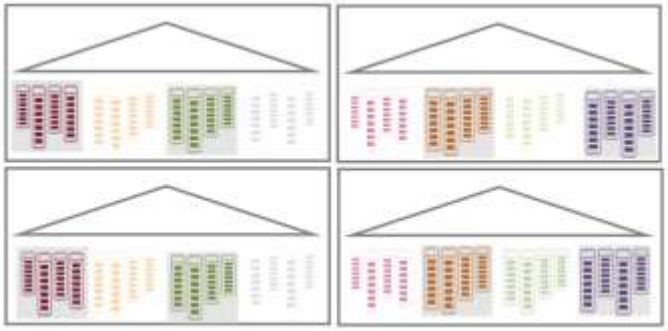

Fig. 12.Globally consistent home location service with duplicated IMCUs

The IMCU contexts in the second set are executed by a single background process as they are not required to be physically populated in the local column store. However, these IMCU contexts are still registered with the local in-memory home location service index where the corresponding HLEs store the remote primary and the secondary home locations of the IMCU contexts.

Once all 'follower' instances release shared access on the object distribution lock, each instance eventually generates a globally consistent view of the local home location service map with IMCUs duplicated on appropriate designated home locations (Fig. 12).

C. Finalization Broadcast

The instance that had been chosen as the 'master' in phase A en-queues waited request for exclusive access on the object distribution lock once it completes phase B. By the time it becomes the 'master' again by acquiring exclusive access on the lock, all 'follower' instances have completed the steps of the decentralized phase B. The 'master' takes a timestamp and broadcasts it to all instances. Each instance records the completion timestamp of the current duplication procedure at an object level.

V.Fault Tolerant Query Execution

The column store home location service serves as the fundamental data structure that seamlessly interfaces with the SQL parallel query execution engine and provides distributed fault tolerant completely local in-memory compression unit scans.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

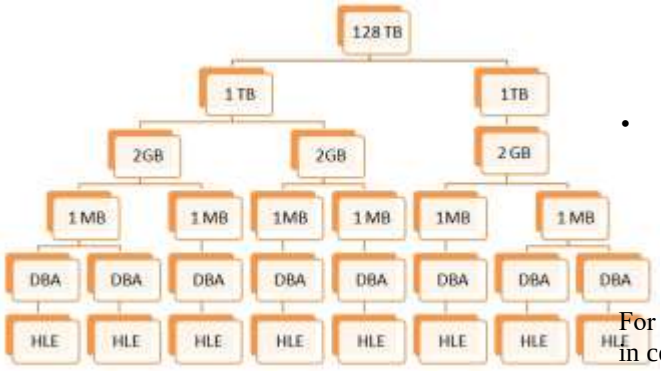

Fig. 13.In-memory home location service: collection of home location indexes

A. In-memory Home Location Service

The in-memory home location service serves as the fundamental data structure enabling distributed fault tolerant query execution across all column stores in the cluster. It serves as a data block address based lookup service for fast access of home location entries or HLEs, where each HLE holds a set of information related to a single IMCU context. The service divides the entire on-disk database block address space into regions of 128 TB of linear address space, where each individual 128 TB of data block address space is represented by an index (Fig. 13). Given the data block boundaries of the IMCU context in an HLE, the intermediate branch and leaf nodes are created on demand before inserting the HLE. The home location service provides a basic application programming interface that takes in a set of data block addresses and returns the set of IMCU contexts for the IMCUs that cover these data blocks, when applicable. The service is used by the transaction manager as well as the SQL execution engine to detect whether columnar representation exists for a data block of an RDBMS object either in the local column store or a remote column store.

The relevant set of information held by each HLE is enumerated as follows:

- IMCU memory offset ○NULL is remote IMCU ○Non-NULL if IMCU populated in local column store
- IMCU home location information for Copy 1
  - Instance Home Location ○ Timestamp of Registering this Home Location
- IMCU home location information for Copy 2
  - Instance Home Location ○ Timestamp of Registering this Home Location
- IMCU boundaries
  - Set of data block address runs
    - ③ Start data block address
    - ③ Contiguous run of data blocks

For example, the HLEs for IMCU context 'IMCU 1' in column store of instance A and instance B contain the same values except the IMCU memory offset; non-zero physical offset in case of instance A where the IMCU is physically populated (Fig. 14).

Instance	IMCU Context	IMCU Boundaries	Home Locations	IMCU Offset
A	IMCU 1	<E1, E2'>	[A: T1A] [C: T1A]	Non-NULL
B	IMCU 1	<E1,E2'>	[A: T1B] [C: T1B]	NULL

Fig. 14.IMCU context IMCU1 with globally consistent boundaries and home locations

B. Parallel Query Execuiion in a Stable Cluster

Let's consider the scenario where an analytic query is executed on an RDBMS object in a stable cluster on completion of duplicated distribution of the corresponding inmemory object. The fact that the in-memory home location service is globally consistent across the column store enables a database client to initiate query workloads from any database instance in the cluster. We select instance D as the query coordinating instance for ease of explanation of the rest of section V.

Once the query is coordinated, the SQL query optimizer [13] calculates the degree of parallelism based on the cost of accessing the in-memory object and ensures that at least one parallel execution scanner process gets allocated on the instances populated with the IMCUs for the in-memory object.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

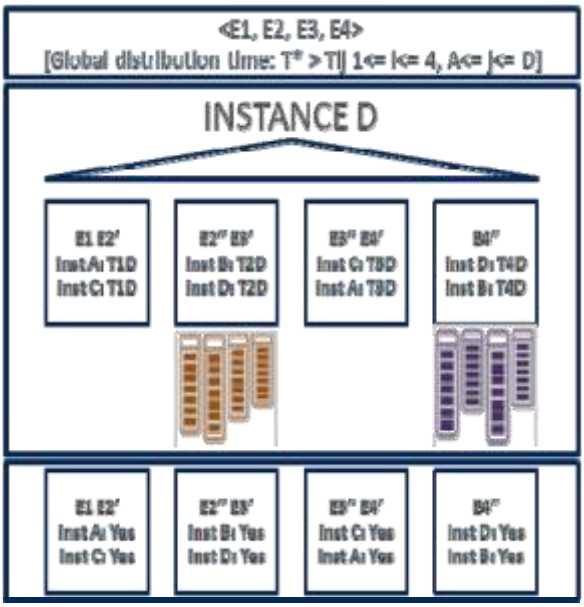

Fig. 15.Query coordinated on instance D in a stable cluster after initial duplication

The coordinator process first retrieves the set of Oracle data blocks relevant for the query. It then consults its instancelocal home location service to generate 'IMCU-aligned' worksets or 'granules' that are 'designated' to be represented inmemory either local or remote (Fig. 15). The target locations of granules missing representation in the column store are flagged as UNDEFINED. For the rest of the 'IMCU-aligned' granule, the following mechanism is employed by the coordinator to assign the target location for the granule.

1. The status of both home location instances is checked using Oracle RAC cluster membership table. An instance home location is considered active for the in-memory object if its most recent start up timestamp is lesser or earlier than the most recent global object distribution timestamp. An instance is considered dead for an inmemory object if it is actually dead or the recent start up timestamp is later than the most recent global object distribution timestamp. If one of the locations is not active, the other location is selected towards step 2. If both are active, step 2 gets employed for both home locations. If both are inactive, both locations are unsuitable candidates for target location of the granule, and the target location of the granule is flagged as UNDEFINED.
2. The registration timestamp of the chosen home location is compared with the global object distribution completion timestamp. If the registration timestamp is lesser than the distribution timestamp, the location serves as a candidate for target location for the granule.
3. At this point, if a single home location stays as the target location candidate for the granule, the granule is assigned the home location. If both home locations are unsuitable,

   the granule is tagged with an UNDEFINED target location. If both locations are suitable, either the primary home location or the secondary home location gets selected as the target granule location based on a hash function on the coordinator process id. This allows for both execution scale out as well as utilization of both copies by multiple concurrent queries in a stable cluster.

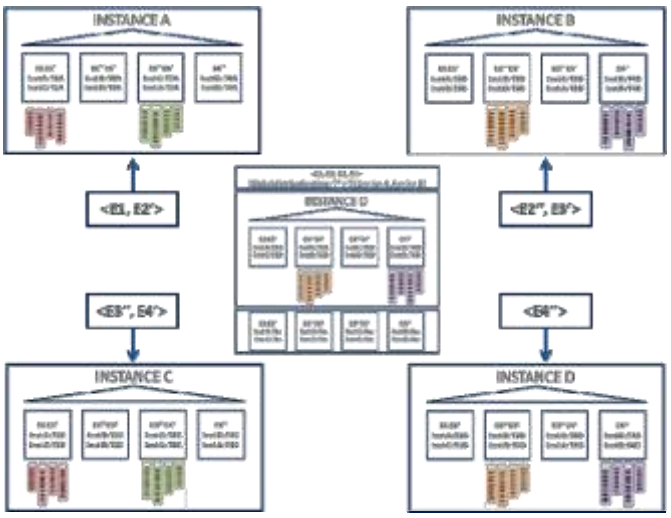

Fig. 16.IMCU-aligned granules generated based on primary IMCU home locations ensuring full in-memory scans

The coordinator allocates (N+1) granule distributors, one for each of the N instances in the cluster, and one for nonaffined granules. Once the target locations have been assigned to all granules, the granules get compiled into the instancespecific distributors. Granules that are tagged with UNDEFINED locations as well as granules that are not represented in-memory as per the local home location service are compiled into the non-affined distributor.

On completion of granule distribution, each parallel execution scanner process starts de-queuing granules from its respective distributor (Fig. 16). For each granule, it consults its local home location service to ensure completely local inmemory access of the IMCUs. As evident from Fig. 15 and Fig. 16, in a stable cluster, the registration timestamps of both home locations for all IMCU contexts are lesser than the global duplication timestamp. Therefore, either of the home locations for an IMCU context can serve as target location candidates for 'IMCU-aligned' granules. The coordinator selects the set of either all primary locations for all IMCU contexts or the set of

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

all secondary home locations for all IMCU contexts to set up granule contexts. This allows for efficient scale-out for analytic scans as well as load balancing across duplicate copies of the IMCUs across the cluster on concurrent query workloads, evident from Fig. 16 and Fig. 17.

Besides distributed execution in a stable cluster, the mechanism described in this section ensures glitch-free faulttolerant query execution during rebalancing of in-memory objects on cluster topology changes. The rest of the section will demonstrate the fault tolerance aspects of distributed query execution through a set of cluster topology change scenarios.

Fig. 17.IMCU-aligned granules generated based on secondary IMCU home locations ensuring full in-memory scans

C. Parallel Execution on Server Failure

When analytic queries are executed on a duplicated inmemory object after failure of a single instance (let's consider instance B), the coordinator granule generation mechanism ensures that parallel execution processes still undergo completely local IMCU scans (Fig. 18 and Fig. 19). Both home locations for IMCU contexts IMCU1 and IMCU3 remain suitable as target location for the granules <E1, E2'> and <E3'', E4'>.

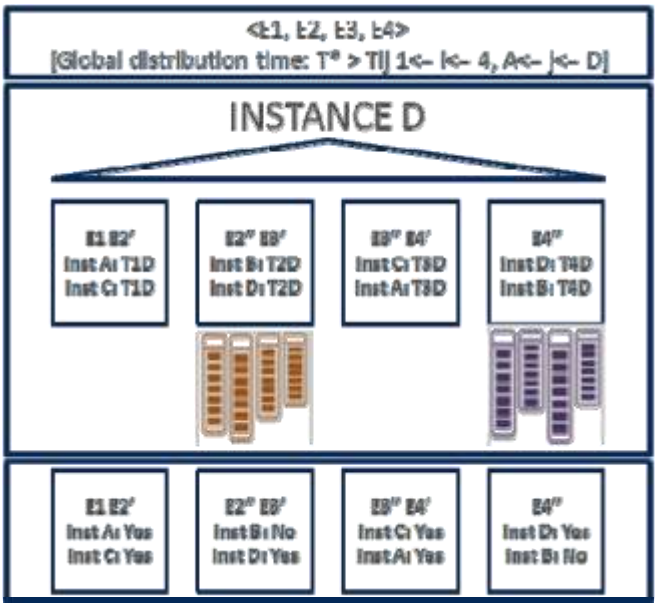

Fig. 18.Query coordinated on instance D after death of instance B

However, for granules <E2'', E3'> and <E4''>, the coordinator selects instance D as the target home locations because the cluster membership table has updated the status of the instance B to be non-active. Therefore, granules <E1, E2'> and <E3'', E4'> are executed in instances A and C while granules <E2'', E3'> and <E4''> are executed in instance D, resulting in fully local IMCU accesses.

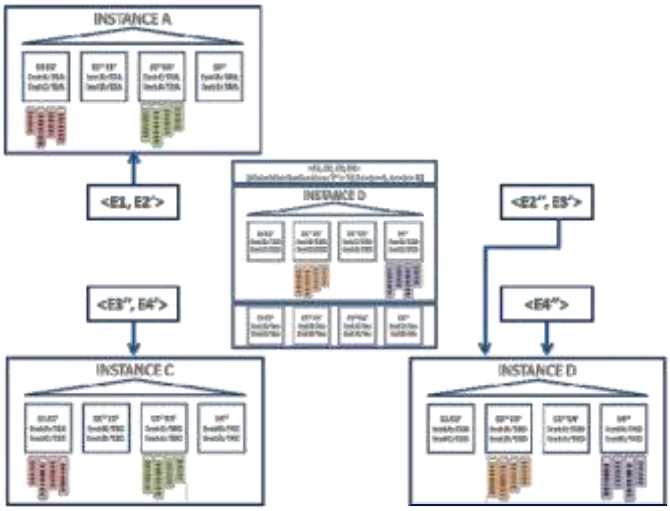

Fig. 19.Glitch free full in-memory parallel query execution on a server failure utlizing duplicate copies of IMCU contexts IMCU2 and IMCU4

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

D. Parallel Execution during Concurrent Redistribution across Remaining Servers

The fact that IMCU contexts IMCU2 and IMCU4 are left with a single copy across the global column store triggers the redistribution of these IMCU contexts in the remaining active instances in the cluster. The redistribution mechanism is the same as the one described in section IV, but uses the same object layout snapshot as of the previous duplication such that the same IMCU contexts are generated. Since the set of participating instances exclude instance B, one of the home locations for the IMCU contexts IMCU 2 and IMCU 4 get modified (Fig. 20), while they remain the same for rest of the IMCU contexts.

IMCU Context	IMCU Boundaries	Old Home Locations Assignments	New Home Location Assignments
IMCU 1	<E1, E2'>	[Inst A] [Inst C]	[Inst A] [Inst C]
IMCU 2	<E2'', E3'>	[Inst B] [Inst D]	[Inst A] [Inst D]
IMCU 3	<E3'', E4'>	[Inst C] [Inst A]	[Inst C] [Inst A]
IMCU 4	<E4''>	[Inst D] [Inst B]	[Inst D] [Inst C]

Fig. 20.Change in home loction assignments for IMCU contexts IMCU2 and IMCU4 due to eviction of instance B

The fact that the actual IMCU duplication is performed by each of the instances asynchronously without peer-peer communication implies that the home location changes get updated out-of-sync on each column store home location service. However, even though they get updated asynchronously, the updates modify the registration times with timestamps greater than the previous global duplication timestamp. Therefore, during the time when the duplication of IMCU contexts IMCU2 and IMCU4 takes place, query coordinators of concurrent queries select the stable home location (instance D) for both IMCU2 and IMCU4 contexts and therefore for granules <E2'', E3'> and <E4''> (Fig. 21). With the completion of the duplication, once the global object duplication timestamp gets updated, both home locations for IMCU contexts IMCU2 and IMCU4 become suitable target locations for granules <E2'', E3'> and <E4''> (Fig. 22).

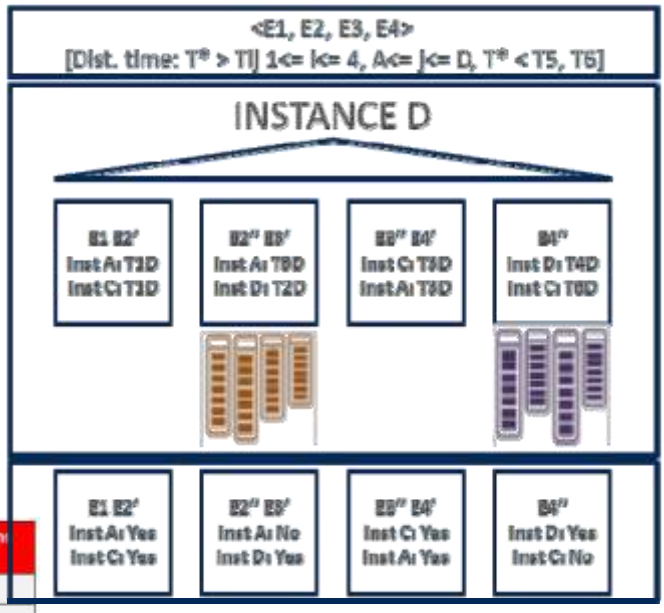

Fig. 21.Query coordinated on instance D during redistribution of copies of IMCU contexts IMCU2 and IMCU4 in the cluster of 3 instances

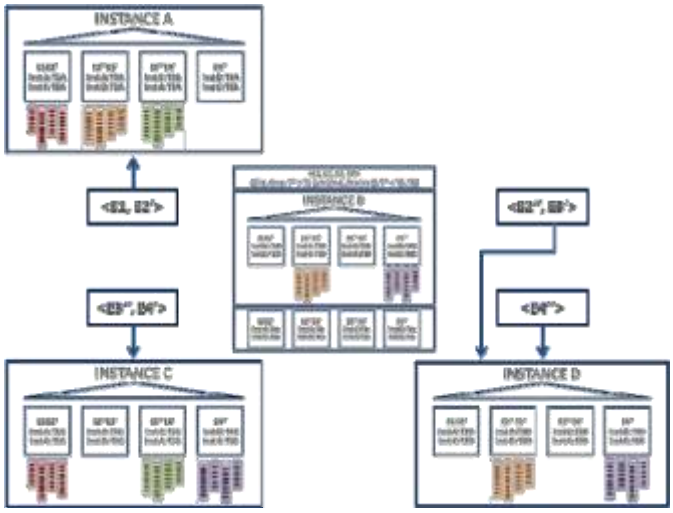

Fig. 22.Glitch free full in-memory parallel query execution on stable copies of IMCU contexts while duplicate copies of IMCU contexts IMCU2 and IMCU4get created in new home locations

E.Parallel Execution during Redistribution on Server Addition

Once instance B comes up and becomes active again, redistribution of the in-memory object gets triggered to rebalance the duplicate copies across the cluster. The duplication mechanism is the same as before and again it uses the same object layout snapshot as of the previous duplication such that the same IMCU contexts are generated. Since the set of

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

participating instances now includes instance B, the Rendezvous Hashing scheme generates home locations that are consistent with the ones generated during the initial duplication of the in-memory object. The home locations of the IMCU contexts IMCU1 and IMCU3 remain the same, while the home locations for the IMCU contexts IMCU 2 and IMCU 4 get modified to the original values (Fig. 23).

IMCU Context	IMCU Boundaries	Old Home Locations Assignments	New Home Locations Assignments
IMCU 1	<E1, E2'>	[Inst A] [Inst C]	[Inst A] [Inst C]
IMCU 2	<E2'', E3'>	[Inst A] [Inst D]	[Inst B] [Inst D]
IMCU 3	<E3'', E4'>	[Inst C] [Inst A]	[Inst C] [Inst A]
IMCU 4	<E4''>	[Inst D] [Inst C]	[Inst D] [Inst B]

Fig. 23.Original home loction assignments for IMCU contexts IMCU2 and IMCU4 due to addition of instance B

During the redistribution of the copies of IMCU contexts IMCU2 and IMCU4 back into instance B, the asynchronous peerless cluster-wide duplication mechanism may result in updating home location entries for IMCU2 and IMCU4 in instances A, C, and D with instance B as one of the home locations even when the actual population of the contexts have not taken place in instance B.

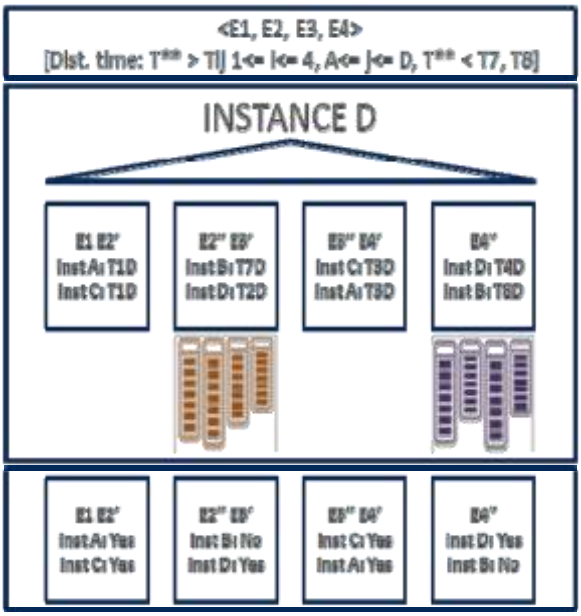

Fig. 24.Query coordinated on instance D during redistribution of copies of IMCU contexts IMCU2 and IMCU4 in the cluster of 3 instances

However, the updates modify the registration times with timestamps greater than the previous global duplication timestamp. Therefore, during the time when the population of IMCU contexts IMCU2 and IMCU4 takes place in instance B, query coordinators of concurrent queries select the stable home location (instance D) for both IMCU2 and IMCU4 contexts and therefore for granules <E2'', E3'> and <E4''>. (Fig. 24) With the completion of the duplication, once the global object duplication timestamp gets updated, both home locations for IMCU contexts IMCU2 and IMCU4 become suitable target locations for granules <E2'', E3'> and <E4''> (Fig. 25).

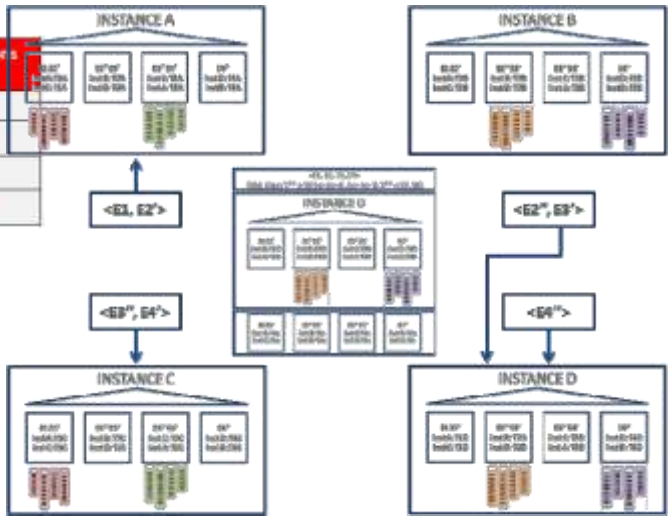

Fig. 25.Glitch free full in-memory parallel query execution on stable copies of IMCU contexts while duplicate copies of IMCU contexts IMCU2 and IMCU4get redistributed in their original home location (instance B)

The following section demonstrates the impact of duplication of the columnar format on performance of analytic query workloads across these cluster topology change scenarios through a series of validation experiments.

VI. Performance evaluation of high availability architecture

Since its release in 2014, the scale-out performance of the distributed architecture of Oracle Database In-memory gets exhaustively evaluated continuously through real-world enterprise workloads in the field. However, real-world evaluation of the high availability aspects of the architecture is not a routine affair and remains untested until deployments incur cluster outages. In this section, we present a preliminary evaluation primarily to validate the high availability and fault tolerance aspects of the distributed architecture through a set of experiments simulating cluster topology changes. The experiments have been designed to demonstrate and verify the capabilities of the architecture that include comparison of query performance across a server failure on duplicated inmemory and non-duplicated in-memory objects a) across a server failure, b) during redistribution of IMCUs across

remaining servers after server failure, and c) during redistribution of IMCUs once the failed server is added back to the cluster. We conclude the section demonstrating sustained query throughput on a duplicated RDBMS table inmemory in an experimental setup on an 8-instance cluster undergoing constant topology changes, where queries are coordinated on a dedicated instance that is kept stable, while the rest of the instances get automatically aborted and restarted one-at-a-time.

A.Hardware and Schema Setup

All four experiments are conducted on Oracle Exadata Database Machine version X4-2 [14], a state-of-the-art database multi-processor multi-core server and storage cluster system introduced in 2013. The X4-2 database machine allows a RAC configuration of 8 RDBMS server instances, each equipped with 2 12-core Intel Xeon processors and 256GB DRAM, and 14 shared storage servers amounting to 200TB total storage capacity, over a state-of-the-art Direct-to-Wire 2 x 36 port QDR (40 Gb/sec) InfiniBand interconnect.

We use an in-house 'ATOMICS' table created with 1 billion rows and 13 columns resulting in an on-disk storage size of 84.62GB for the experiments. Two more versions of the table 'ATOMICS2SF' and 'ATOMICS4SF' are further created on the base atomics table with a scale factor of 2 and 4 respectively. The size of the in-memory column store is set to 128GB on all database instances.

B.Experiments

1)Single Server Failure Exeriment

Three different RAC configurations (2 instance cluster, 4 instance cluster, and full 8 instance cluster) are used for this experiment to demonstrate the impact of loss of columnar format on query performance after server failure when the 'ATOMICS' table gets distributed without duplication. For the 2-instance scenario, the original 84.62GB ATOMICS table is used, while for the 4-instance and 8-instance scenarios, the 2x and 4x scale factor tables are used. The tables are first distributed with NO DUPLICATE option resulting in single copy of the IMCUs of the corresponding in-memory tables. Fifty iterations of three different analytic queries are executed on stable clusters. Then one of the instances is aborted manually, and the twenty iterations of the same query set are executed. A manual parameter is set to prevent redistribution of the in-memory object while the queries take place. The above experiment is repeated with same tables but distributed with DUPLICATE option.

Fig. 26 demonstrates regression in average query elapsed times on non-duplicated tables on server failures. The percentage of regression is observed to be most significant on the 2-instance cluster test case, as around half of the inmemory columnar data gets lost due to the loss of an instance, while on the 8-instance cluster test case, around one-eighth of the data get lost due to the loss of an instance. On the other hand, no visible performance regression of average query elapsed times is observed on duplicated in-memory tables on a single server failure irrespective of the size of the cluster.

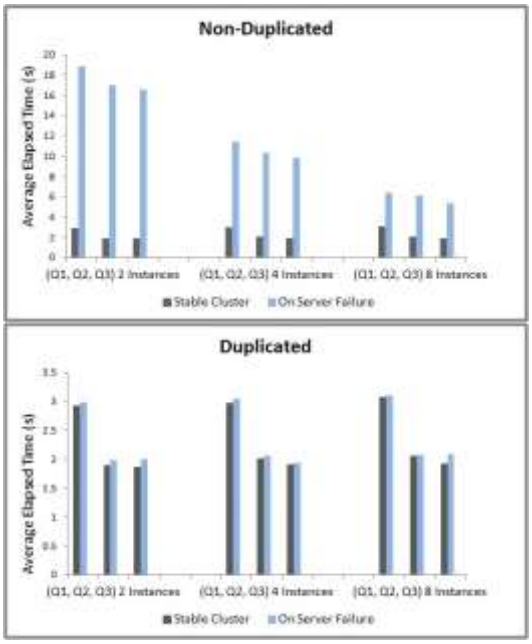

Fig. 26.Average elapsed times of query sets executed on non-duplicated and duplicated 'ATOMICS', 'ATOMICS2SF', and 'ATOMICS4SF' tables on a single server failure

2)Redistribution Across Remaining Servers

The same experiment setup employed in subsection B is reused for this experiment, without explicitly setting a manual parameter to prevent redistribution. Therefore, all queries get executed while redistribution of the lost IMCUs takes place across the cluster. Fig. 27 illustrates the elapsed time performance of the query set executions. For non-duplicated tables, performance regression is still observed when compared to the elapsed times observed in stable cluster configurations. However, the average elapsed time decreases due to the fact that the lost IMCUs get populated in the remaining instances over the duration of the experiment. For duplicated tables, no visible differences are observed when compared to results from stable cluster configurations.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

3)Redistribution After Server Addition

The same experiment setup employed in subsections B and C is reused for this experiment, with the aborted instances in the above test cases restarted and added back to the cluster. The experiment forces all queries to get executed once the redistribution phases of the IMCUs get initiated. Fig. 28 illustrates the elapsed time performance of the query set executions. The redistribution process moves duplicate copies of relevant IMCUs back to their original home location of the restarted instance. Therefore, even for non-duplicated tables, no visible performance regression is observed when compared to the elapsed times observed in stable cluster configurations as IMCU copies are available in instances due to redistribution of IMCUs lost from the aborted instance that takes place in the test scenarios described in subsection C. As expected, no visible differences are observed for duplicated tables when compared to results from stable cluster configurations.

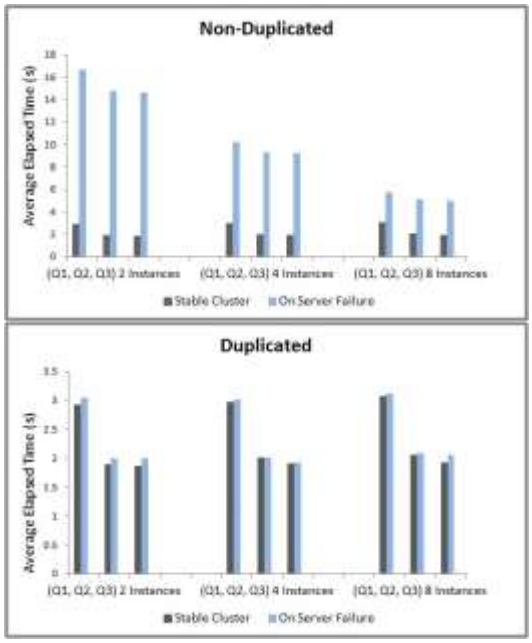

Fig. 27.Average elapsed times of query sets executed on non-duplicated and duplicated 'ATOMICS', 'ATOMICS2SF', and 'ATOMICS4SF' tables during reduplication of IMCUs across remaining servers

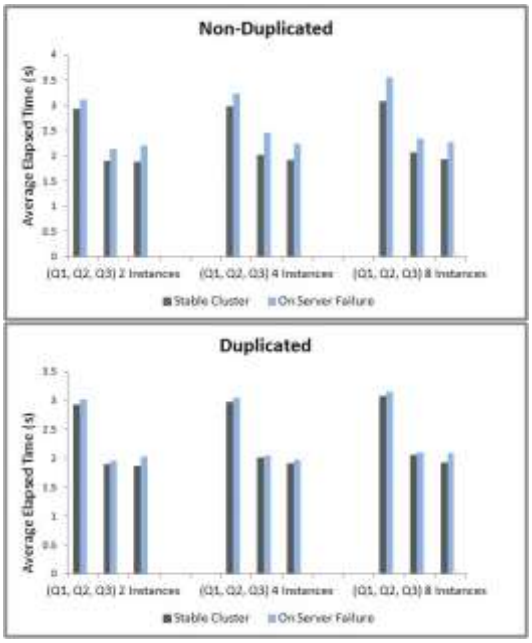

Fig. 28.Average elapsed times of query sets executed on non-duplicated and duplicated 'ATOMICS', 'ATOMICS2SF', and 'ATOMICS4SF' tables during redistribution of duplicated IMCUs in restarted server instance

4)Sustained Query Execution on Continuous Cluster Topology Change

The final set of experiments in this evaluation exercise combines all the scenarios described in subsections 2, 3, and 4 by configuring sustained query execution over a period of three hour while the cluster undergoes topology changes. For this experiment, the 2x scale factor 'ATOMICS2X' is distributed in duplicate mode in a cluster configured with 4 instances. The test case is configured such that instance A always remains active while rest of the instances (B, C, and D) are aborted and restarted (after 3 minutes of abortion) one after the other in a round robin fashion in 5 minute intervals. All queries are executed from the stable instance A. Fig. 29 illustrates average elapsed time collected every 60 seconds across the three hour period. Sustained elapsed times are observed due to glitch free fault tolerant distributed query executions undergoing fully local in-memory columnar format scans across all active servers at any given snapshot within the three hour execution window.

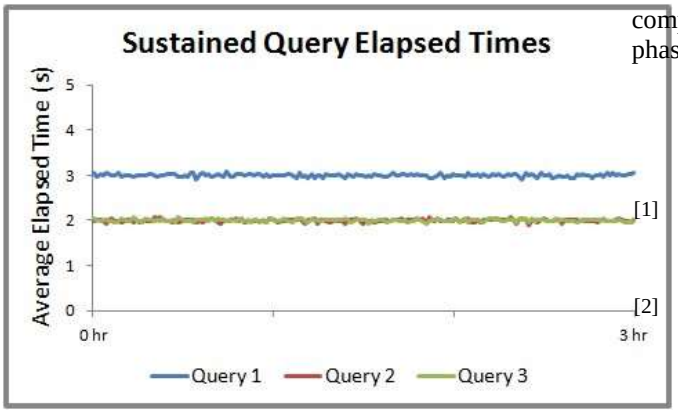

Fig. 29.Fault tolerant in-memory parallel query execution on 'ATOMICS2SF' sustained for a duration of 3 hours under constant cluster topology changes due to removal and addition of a server at regular intervals in a cluster

The preliminary evaluation provides a compact yet complete demonstration of the high availability and fault tolerance capabilities of Oracle DBIM. The results do demonstrate that Oracle DBIM ensures seamless distributed query execution on duplicated in-memory objects across a cluster of servers undergoing continuous topology changes, as long as long as there exist one stable copy of the underlying in-memory compression units

VII.Conclusion

A new breed of mixed OLTAP applications have emerged that require real-time analytic insights on massive volumes of data in live mainstream production environments as well as traditional data warehouse ones. Oracle introduced the Database In-memory Option (DBIM) in 2014 as the industryfirst dual format in-memory RDBMS highly optimized to break performance barriers in analytic query workloads without compromising or even improving performance of regular transactional workloads. Since the new columnar format is maintained purely in-memory without additional logging overheads, the new in-memory option has been implemented as a distributed architecture to provide maximal availability of the columnar format supporting fault-tolerant in-memory query execution across cluster topology changes, besides scaling out main memory capacity and query execution throughput. This paper primarily presents the maximal availability architecture of Oracle DBIM. The architecture provides real-time duplication of the in-memory columnar format for high availability along with fault tolerant query execution across server failures. The architecture also ensures minimal recovery impact on the columnar format across cluster topology changes by guaranteeing fully glitchfree in-memory query execution mechanisms during completely asynchronous efficient rebalancing phases.

VIII.References

[1] N. Elmqvist, P. Irani, "Ubiquitous analytics: interacting with Big Data anywhere, anytime," Computer, 46, 4, April 2013.

[2] A. Goel et. al., "Towards scalable real-time analytics: an architecture for scale-out of OLxP workloads," Proceedings of the 41st International Conference on Very Large Data Bases, vol. 8, pp. 1716-1727, August 2015.

[3] D. Burdick, P. M. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan, "OLAP over uncertain and imprecise data," The International Journal on Very Large Data Bases , vol. 16, pp. 123-144, April 2007.

[4] "Oracle database in-memory, an Oracle white paper," Oracle Openworld, October 2014.

[5] T. Lahiri et al., "Oracle database in-memory: a dual format in-memory database," Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, pp. 1253- 1258, April 2015.

[6] M. Michael, "Scale-up x Scale-out: A Case Study using Nutch/Lucene," Proceedings of the IEEE International Symposium on Parallel and Distributed Processing. IPDPS'07. IEEE, pp. 1–8, 2007.

[7] N. Mukherjee et al., "Distributed architecture of Oracle database inmemory," Proceedings of the 41st International Conference on Very Large Data Bases, vol. 8, pp. 1630-1641, August 2015.

[8] "Oracle12c Concepts Release 1 (12.0.1)," Oracle Corporation, 2013.

[9] W. Bridge, A. Joshi, M. Keihl, T. Lahiri, J. Loaiza, and N. MacNaughton, "The Oracle universal server buffer manager", in *Proceedings of VLDB '97*, pp. 590-594, 1997.

[10] "Parallel Execution with Oracle 12c Fundamentals," An Oracle White Paper, Oracle Openworld, 2014.

[11] N. Lynch, "Distributed Algorithms," Morgan Kaufmann Publishers. ISBN 978-1-55860-348-6.

[12] J. C. Laprie, "Dependable computing and fault tolerance: concepts and terminology," Proceedings of 15th International Symposium on FaultTolerant Computing (FTSC-15), pp. 2–11, 1985.

[13] D. Das et. al., "Query optimization in Oracle 12c database in-memory," Proceedings of the

41st International Conference on Very Large Data Bases, vol. 8, pp. 1770-1781, August 2015.

[14] R. Greenwal, M. Bhuller, R. Stackowiak, and M. Alam, "Achieving extreme performance with Oracle Exadata," McGraw-Hill, 2011.

Mr. Shaik Shoyab pursuing M. Tech in Computer Science and Engineering from PACE Institute Of Technology and Sciences affiliated to the Jawaharlal Nehru Technological University ,Kakinada.

Mr.V.Sri Harsha has received his B.Tech and M.Tech PG. He is Dedicated to Teaching Field from the last 6 Years. He has Guided 10 P.G Students and 20 U.G Students. At Present He is Working as Asst.Professor in PACE Institute Of Technology and Sciences, Vallur, Prakasam(Dt), AP, India.He is Highly Passionate and Enthusiastic about his Teaching and Believes that Inspiring Students to Give of his best in order to Discover what he Already knows is better than Simply Teaching.