Proc. 2010 Computer Measurement Group Conf., Dec. 6-10, 2010, Orlando, Florida.

Method and Model to Assess the Performance of Clustered Databases: The Oracle RAC Case

Kevin Molloy Daniel A. Menascé

Dept. of Computer Science, MS 4A5
The Volgenau School of IT & Engineering
George Mason University
Fairfax, VA 22030, USA
kmolloy1@gmu.edu menasce@gmu.edu

Abstract

Oracle allows for locking at the row level and uses a multi-version read consistency model. To provide scalability, Oracle embeds lock data into the data blocks within the database. For added availability and performance, Oracle provides Real Application Cluster (RAC), which has a shared cache and can operate on a shared Storage Area Network. This paper presents a methodology and analytic model aimed at assessing the performance impact of using a RAC versus running the database on a single server. Experiments are used to demonstrate the use of the methodology and model.

1 Introduction

Modern database systems deal with very high demands from application and system architects. Database systems are expected to provide fault tolerant and scalable solutions to serve increasing workload and data volumes. Oracle's Real Application Cluster (or RAC) option provides both high availability and scalability. Oracle promotes RAC as being the key component in their grid architecture solution. This framework is comprised of clusters of smaller servers instead of larger, more expensive SMP systems. A single node failure in a cluster does not compromise its ability to provide database services, and thus, increases availability.

Predicting the performance of existing applications in RAC is not trivial. A wide range of claims exist from near linear scalability to 10 to 20 percent overhead [9]. Clear examples exist that show very good scalability, for example the November 2009 Sun TPC-C benchmark achieved more than 7 million tpm-C on a 12 node RAC. In this situation, the application was specifically adjusted to optimize performance under RAC. Oracle's Real Application testing suite allows for workload capture with the ability to replay these transactions in RAC. Benchmarks and load testing evaluations lack the flexibility of analytical models, since they only offer metrics for specified workload intensities. Vendors such as BEZ Systems do offer tools based on analytical models that can predict RAC performance.

This paper investigates RAC's scalability by using an analytical model that captures the additional demands and constructs a queuing network (QN) model to predict performance under varying conditions. Previous work by Sasaki and Tanaka [8] establishes methods to estimate required inter-cluster coordination. Dempster also provides methods for predicting the performance of Oracle RAC and insight into the distributed locking protocols [2]. This paper expands on these ideas by providing a method to compute the service demands for transactions in Oracle, and adjusts the service demands to compensate for the RAC overhead. These service demands are then used as input to a QN model to predict response times under different workloads.

The rest of the paper is organized as follows. Section 2 presents some background information on RAC and provides some definitions. Section 3 presents the notation and modeling assumptions. Section 4 builds up the service demands and defines the QN used to predict RAC performance. Section 5 provides the results from experiments used to validate the model. Finally, Section 6 presents some concluding remarks and discusses future work.

2 Oracle RAC

Oracle RAC is a parallel database system that uses a share everything approach. A single physical copy of the database is manipulated by a group of nodes that form a cluster. Every node in the cluster is connected to a shared storage device (SAN), and thus, every node can access any part of the database. The nodes are connected via an isolated high speed interconnect. Commonly, this interconnect is simply a gigabit ethernet network, with data encoded in

the UDP protocol. However, very high performance clusters use vendor specific protocols and Infiniband technology. Figure 1 illustrates a typical RAC configuration.

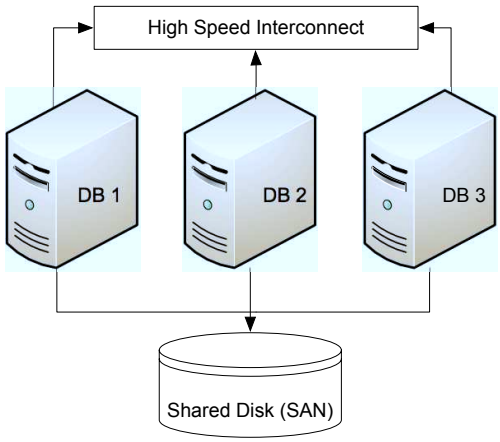

Figure 1: Oracle RAC.

2.1 Oracle Terminology

Oracle stores rows (tuples) in a $block$ (sometimes referred to as a page). The set of blocks that comprise a table (relation) is referred to as a $segment$. Segments exist for all physically stored objects in the database (indexes, partitions of tables, etc). An Oracle database refers to the files that contain the data within the database. An Oracle instance is a group of processes that support multi-user access to the database. In a RAC environment, multiple instances (running on separate nodes) access the same Oracle database. Each Oracle instance is comprised of a shared global area, or SGA. Each instance maintains a local buffer cache of data blocks within the SGA. The collection of Oracle instance's must adhere to a protocol to maintain coherence amongst the distributed caches. This protocol is known as cache fusion and is implemented by the global cache service processes (GCS) of each instance (see Section 2.4).

Oracle uses a multi-version read consistency model that provides a mechanism in which readers never wait for writers [12] (see Section 2.2). A monotonically increasing logical clock referred to as a $system\,change\,number$, or SCN, provides a logical ordering of events within the database instances. Each block (and sometimes each row depending on replication settings) is timestamped with the SCN when modified. Undo segments are responsible for maintaining the "before image" of data that is changed during the course of a transaction. In the event of a transaction failure, this data is used to rollback the transaction. The data within the undo segments is also used to support the read consistency model and several other Oracle features.

2.2 Current and Consistent Reads

Blocks are accessed in either current (CU) or consistent read (CR) mode. CR reads provide the mechanics to support the multi-version consistency model mentioned previously. At the beginning of each statement (or optionally at the transaction level), the current SCN value is associated with the statement. If a block is read that has been modified since the statement started executing (block SCN $>$ statement SCN) or if it contains uncommitted modifications, Oracle constructs a consistent image by making a copy of the block in the instance's cache and labeling it as a consistent image (versus the current state of the block). Changes to the consistent image copy of the block are rolled back (using data in the undo segments) until the SCN of the block precedes the statement's SCN. See [12] and [5] for more details on consistent reads.

CU mode guarantees that the data in the block is the most up to date copy across all instances (i.e., exclusive access). Data Manipulation Statements (DML) use CRs to perform data selection, and then blocks are acquired in current mode (CU) to perform the modification (see page 247 of [5]). For example, an $UPDATE$ statement with a $WHERE$ clause selects the data for modification using CR reads, and then performs CU reads before actually changing the data. Coordinating block access in RAC is discussed in Section 2.4.

2.3 Locking Mechanisms

Row locking information is embedded within the data blocks themselves in an area known as the interested transaction list (ITL) [5]. This eliminates the need to keep a global row lock table in memory. When a row is changed, a CU block access is performed. The ITL is then inspected to see if the row is locked by a currently executing transaction. If the row is not locked, the row is modified and the ITL is updated to lock the row. If the row is already locked, the requesting transaction places itself on the lock holder's enqueue. At transaction completion, the enqueue is posted and lock acquisition is attempted again. A process known as "block cleanout" consists of the removal of this locking information from the block.

RAC uses a distributed lock manager (DLM) to coordinate access to global resources (such as enqueues). Each resource is assigned a single node in the cluster as its coordinator, or master. The number of active nodes can change dynamically, causing reconfiguration of master assignments. GES (global enqueue services) manages access to all global resources except for blocks.

2.4 Cache Coherence Method

Cache fusion, sometimes referred to as parallel cache management (PCM) or global cache services (GCS), coordi-

nates access to blocks. Each instance's buffer cache is "fused" together and treated as a single logical unit. Each release of RAC has made improvements to this protocol to improve its scalability [1]. GCS coordinates shared and exclusive access to blocks while row level locks provide transactional consistency. Row level locks (embedded in the ITL) are preserved by ensuring mutually exclusive access to the block when required [4].

Each block is assigned a single instance which is responsible for coordinating and granting access. This instance is known as the *mastering instance* for the block. Segments are divided into groups of 128 blocks, which we define as a master block list, or MBL. Each MBL is mastered by a single instance (see Fig. 2), which is quickly determined via a hash function [3]. This distributes the responsibility for mastering a segment across all instances in the cluster. The exception to this rule are UNDO segments (which are mastered by the instance to which they are assigned). When a segment is accessed at a very high frequency from one instance, GCS can assign that node as the master of the entire segment. This is called dynamic remastering [4], and the motivation is to improve performance of large ETL/batch jobs executed from a single node by reducing the inter-node coordination requirements.

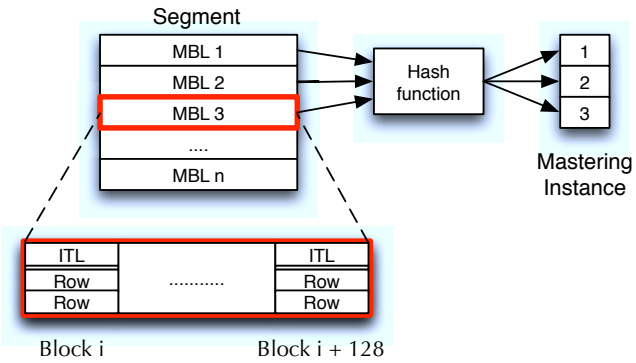

Figure 2: Each block within a segment belongs to a master block list (MBL), which is mastered by a single instance within RAC

GCS grants shared current (SCUR) or exclusive current (XCUR) access to blocks. When access to a block is granted to an instance, that instance becomes the current "owner" of the block. SCUR access can be granted to multiple instances simultaneously and guarantees that each instance has the most current version of the block (i.e., no one can change the block). XCUR access is granted exclusively to a single instance at a time, and is used to support update operations. Instance's that are granted XCUR access to a block also handle CR requests for the block from the other instances in the cluster. CR blocks are not locked or tracked by the master instance because they can only be used within the context of their SCN (never used for update).

Regardless of the number of instances within the cluster, each request involves at most three instances: the master, the requestor, and the current block owner(s). A single instance can assume more than one of these roles (the master and current owner could be the same). Cache fusion operations are thus categorized as either 2-hop or 3-hop (referring to the number of nodes involved in the request). This establishes an upper bound, which is independent to the number of nodes in the cluster, that is key to RAC's scalability. While the number of nodes that participate in a given global cache request may be limited to 3, the volume of global cache requests is dependent on the number of instances (see Section 4.4 and Fig. 6).

2.4.1 Current Mode Block Request

CU requests require exclusive access; a XCUR lock is obtained as shown in Fig. 3. Gaining XCUR access does not require any transactional lock coordination [4]. If the block is already in our cache in XCUR status (step 1), no further action is required. Otherwise, the block's mastering instance is determined. The mastering instance is responsible for coordinating access to the block and granting remote instances access when requested. If the block is locally mastered (step 3), the instance checks to see if it has granted access to this block to another instance. If this is true (step 4), then the mastering instance instructs the local instance to release its lock on this block and send a copy of the block via the interconnect (step 5). A redo log flush is required during lock release to guarantee that the updated data within the block is available in the event of an instance crash. Similar steps are required if shared locks have been granted to remote instance (steps 7 and 8). When the requesting instance is not the mastering instance, the mastering instance is contacted (step 6) and a similar protocol is followed.

Oracle DML statements consist of two parts: query the data (read the block in CR mode) and then make the necessary changes (CU requests) [11] [5]. For blocks with high contention, RAC may require a remotely constructed CR first, and then an XCUR lock. This effectively doubles the inter-node communications and adds significantly to the RAC overhead.

2.4.2 Consistent Read Block Request

CR block requests are satisfied as shown in Fig. 4. Building a CR consists of two steps: 1) obtaining a copy of the block and 2) making the block consistent as of the requested SCN. Figure 4 outlines the first step of this process, except for step 12 (which is discussed below). In the case of steps 5, 8, 13, 16, 17, and 18, making the block consistent to

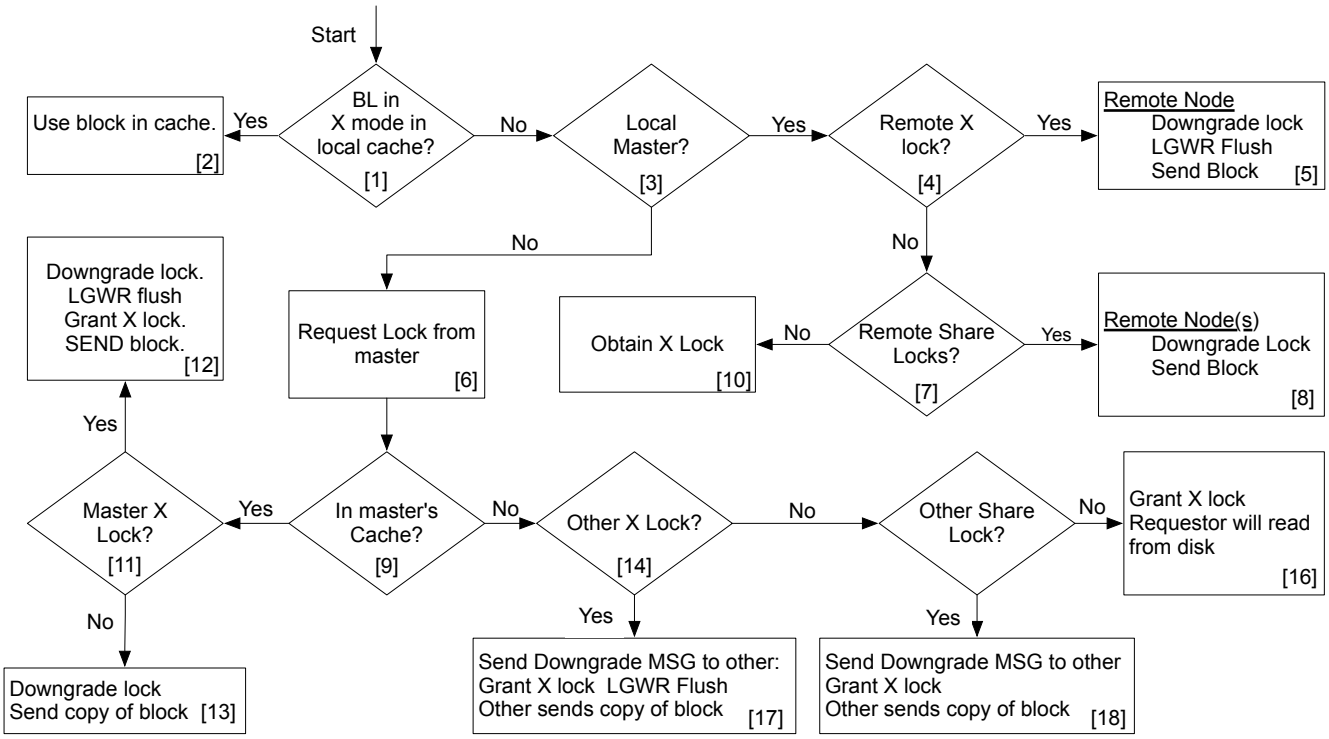

Figure 3: Sequence of actions for a current mode block request (BL=block; LGWR = log writer process; X=exclusive)

the requested SCN may require additional block accesses. These additional accesses follow this same process.

When the block is held in the local cache (step 1), no further action is required. If the requested block is not in any instance's buffer cache, a SCUR is granted to the requestor by the mastering instance and the block is read from disk (steps 5 and 16). If other instances hold the block in SCUR status, SCUR status is granted to the requestor and the block is transferred via the high speed interconnect (steps 13 and 18). This access can be significantly faster than reading the block from disk [1].

There are two methods for creating the CR when the block is held in XCUR status by another instance (steps 8 and 17) as shown in Fig. 5. The first one involves the CR image being assembled by the XCUR owner and the block is transferred via the interconnect (step 4). To prevent an instance from serving too many CR requests for the same block, a limit, called the fairness threshold, is established. This threshold is called "lock down factor" in [8] and defaults to four in Oracle. When an instance receives a CR request that causes it to exceed the fairness threshold, the XCUR lock is downgraded to a SCUR lock and the requestor is sent a copy of the block (step 2). The second method involves the requesting instance constructing the CR. This occurs after a XCUR lock is surrendered because of the fairness threshold being exceeded, and in some situations where the XCUR holder deems CR construction too expensive [4]. Recall that CR blocks do not require locks because they are only used for read. Blocks assigned a status of SCUR do require locks as GCS guarantees that this is the most up to date copy of the block.

3 Modeling Assumptions

The notation used in our model is summarized in Table 1. Our model makes the following simplifications and assumptions. A uniform access distribution model assumes that transactions access each block of a segment with equal probability as shown in Eq. (1). This type of model works well for UPDATE and DELETE statements which target individual row modifications in a uniform manner.

$$pr_t(i,j) = \frac{1}{T_i} \quad \forall\ t,j. \tag{1}$$

With respect to INSERT statements, experiments were performed and the global cache request metrics were captured. Automatic segment space management was enabled for the tablespace that contained this table's segment. This feature allows for increased concurrency by providing transactions separate lists of free blocks and dynamic instance affinity for RAC [12]. Table 2 shows the results, which clearly illustrate that even at high transaction rates, little

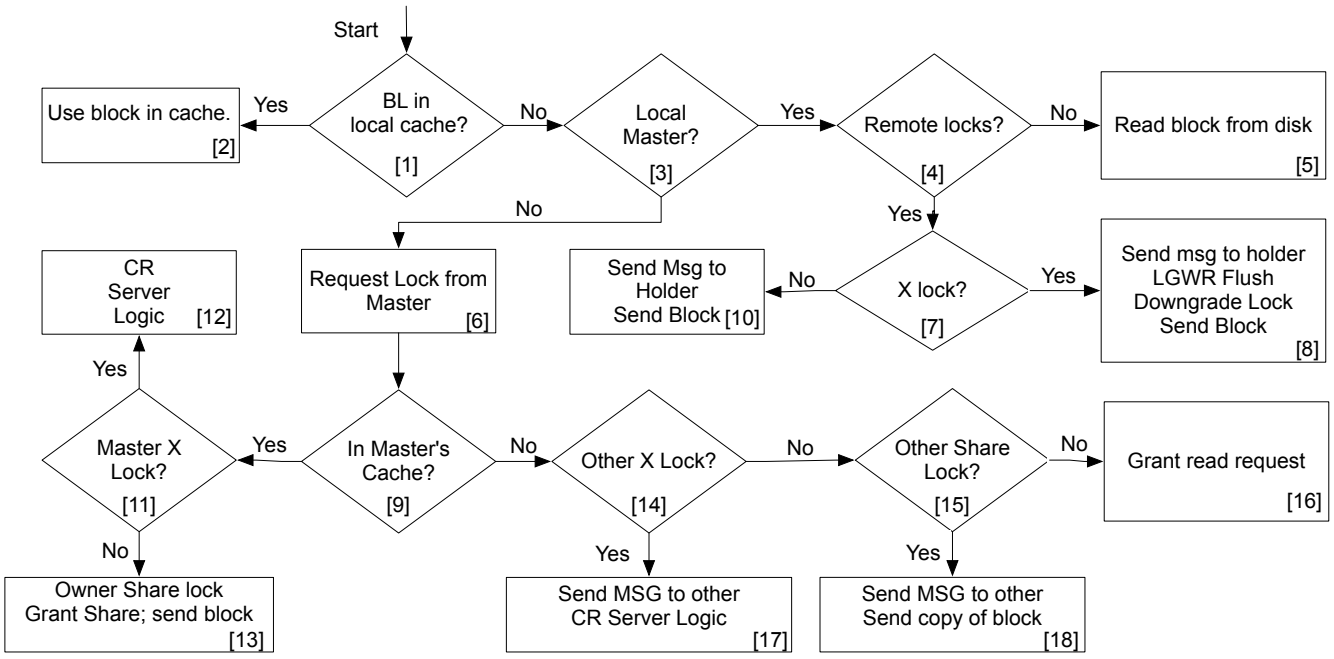

Figure 4: Actions for a CR block request (BL=block; CR = consistent read; LGWR = log writer process; X=exclusive)

B	number of blocks in a master block list
T_i:	number of blocks of segment i. Thus, the number of master block lists for segment i is $\lceil T_i/B \rceil$
$\mathcal{S}$:	set of all segments
$pr_t(i,j)$	probability that transaction t wants to access block i ($i = 1, \ldots, B$) in MBL j ($j = 1, \ldots, \lceil T_i/B \rceil$)
N	number of nodes in the RAC cluster
τ:	duration of the experiment
X_r	throughput of class r transactions
$w_{i,r}$	write ratio of segment i for class r requests where $w_i = \sum_{r=1}^{R} w_{i,r}$
bc_i	number of block changes to segment i ($bc_i = \sum_{r=1}^{R} bc_{i,r}$)
cr_i	number of consistent gets for segment i ($cr_i = \sum_{r=1}^{R} cr_{i,r}$)
cu_i	number of current gets for segment i ($cu_i = \sum_{r=1}^{R} cu_{i,r}$)
$D_{i,r}$	service demand of class r requests at device i.
$U_{i,r}$	utilization of device i by class r requests, where device utilization is given by $U_i = \sum_{r=1}^{R} U_{i,r}$
$GCCR^r_{j,N}$	global cache CR requests for all class r transactions on segment j for a cluster of size N
$GCCRT^r_{j,N}$	number of cache CR requests per class r transaction on segment j for a cluster of size N (equal to $GCCR^r_{j,N}/(X_r \times \tau)$)
$GCCU^r_{j,N}$	global cache CU requests for all class r transactions on segment j for a cluster of size N
$GCCUT^r_{j,N}$	number of cache CU requests per class r transactions on segment j for a cluster of size N (equal to $GCCU^r_{j,N}/(X_r \times \tau)$)
GC_N	for N nodes, number of global cache requests (equal to $\sum_{s\ \in \mathcal{S}, \forall\ r}(GCCR^r_{s,N} + GCCU^r_{s,N})$)
GCT^r_N	for N nodes, number of cache requests per class r transaction ($\sum_{s\ \in \mathcal{S}}(GCCRT^r_{s,N} + GCCUT^r_{s,N})$)
Z_r	Average think time for class r transactions
pr_{2hop}	probability of a 2-node global cache request (equal to $\frac{1}{N-1}$)
pr_{3hop}	probability of a 3-node global cache request (equal to $\frac{N-2}{N-1}$)
NT_{size}	Network time to transmit UDP message where size = msg (500 bytes) or database $block$ (8K)

Table 1: Notation.

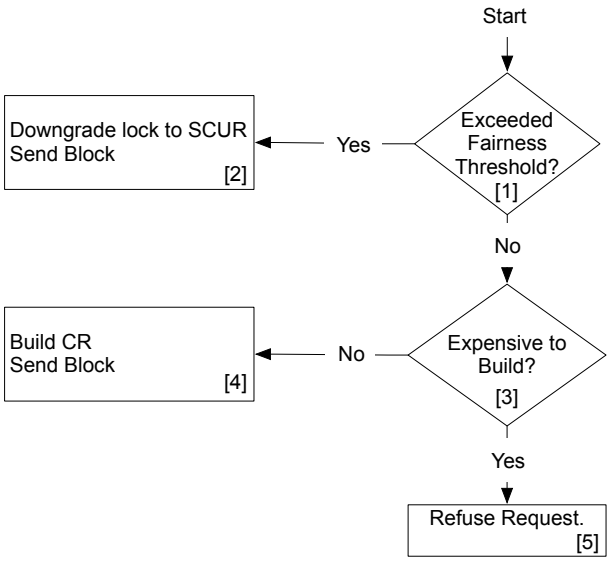

Figure 5: CR Block Construction.

to no global cache interaction is required.

The primary key segment of the table would also be impacted by INSERT statements. If the index keys are very close together (as might be the case with a sequence generated surrogate key), all inserts would target the same leaf blocks of the index. For more information on Oracle B-tree's, the reader is directed to [12]. To minimize the impact of RAC, we assume that an Oracle sequence with the $noorder$ attribute is used with a cache size equivalent or greater than the number of keys that will fit into a leaf block of the index. The $noorder$ attribute specifies that each instance will maintain their own cache of sequence values (versus a global cache shared amongst all instances). If sequence numbers are uniformly requested from each instance, the distance between any two cache numbers for any instances will be the size of the cache value. When this is used as the surrogate key of the table (and it is the leading key of the index), this effectively provides instance affinity for the leaf blocks of the index. Our model assumes this behavior, and thus, the RAC overhead for INSERT statements is considered negligible and is not modeled.

Cache fusion operations that result in a SCUR grant and block shipment via interconnect are not modeled. While this may seem like an oversimplification, consider the following. If frequent CR requests for a specific block are made, that block will most likely remain in the local buffer cache. If the segment is frequently changed, CU and CR requests will dominate the inter-node communications (and these are modeled). Our model assumes that the buffer cache is large enough to hold all blocks accessed during the test. Oracle performs multi-block read aheads during full

X_0	2 Node Config.		3 Node Config.	
	NRSIB	Msg/Row	NRSIB	Msg/Rows
12	100%	0.002	99%	0.01
23	100%	0.001	99%	0.01
57	100%	0.002	99%	0.01

Table 2: X_0 = Throughput (in tps); NRSIB = Percentage of blocks within the table segment that contain data from a single instance; Msg/Row = Number of inter-node messages divided by the number of rows inserted.

segment scans to improve I/O performance. In Oracle 11g, a new protocol for multiple CR block requests exists within cache fusion. Modeling OLTP systems is the primary focus of this model; thus, it is assumed that Oracle will not perform these types of operations. To ensure this behavior, the db_file_multiblock_read_count initialization variable is set to 1 for the experiments (see Section 5).

Oracle may have to access multiple blocks to complete a request when either a CR or CU request is processed by a remote node. Our model assigns equal weights to CR and CU requests with respect to their service demands. For CR construction, this is a function of query duration and block change rates. For simplicity, the model assumes that all transactions are relatively short, and therefore, CR construction can be estimated as a constant (for databases that support a mix of longer running report statements and OLTP against the same tables, this may be a significant simplification). CU requests can also experience a range of response times depending on whether redo or data blocks need to be flushed. A variety of workloads were reviewed using the Oracle 10gR2 and 11g R2 AWR RAC reports, and the response times for 95% of the CU and CR requests were within 1 millisecond. The waits that can cause variations in the service demands for these events are not explicitly modeled (e.g., buffer pins on remote nodes or index leaf node split), but are implicitly modeled by capturing the overall system workload characteristics.

CR and CU requests can also require a synchronous log flush. For a log flush to be required, the block of interest needs to be dirty in the owner's buffer cache. Our model captures this requirement implicitly. An interesting observation is that the performance of select statements (CR blocks) is now dependent on LGWR performance (a condition that does not exist in single instance Oracle) [7].

Dynamic remastering of segments to a single node and parallel SQL statements are not considered in the model. Our model focuses on the overhead of cache fusion, and does not consider other factors introduced by RAC (e.g., SCN synchronization, dictionary locks, and block fetches via interconnect versus disk). Software contention within

the database is not specifically modeled. Asynchronous inter-node requests are also excluded from the model (e.g., block removed from local cache, etc).

4 The Analytic Model

We now describe the analytic model we built to predict RAC performance. Using a single instance configuration, service times are calculated for transaction classes using the service demand law [6]. Global cache coherence requirements are estimated using a series of formulas based on the work by Sasaki and Tanaka [8]. A method for capturing the service demands associated with these coherence requirements is provided. A multi-class closed Queuing Network (QN) model is then developed to approximate response times at varying transaction throughputs.

The steps of our modeling approach are: 1) obtain service demands for an idle system (i.e., no transactions), 2) obtain service demands for a single-instance system, and 3) obtain service demands for processing global cache coherence requests (for the 2-hop and 3-hop cases).

4.1 Oracle Background Processes

Oracle uses several background processes to perform asynchronous tasks that improve performance. The two most important are the DBWR and LGWR processes [2]. Each of these processes perform lazy writing of blocks to disk, with the LGWR process requests sometimes being synchronous in nature. While some methods explicitly model these processes, our method captures their contribution to the system implicitly by capturing the service times in an isolated environment. This avoids having to apportion the global utilizations to individual classes. The service demand for a given transaction will be defined by its CPU (D_{cpu}), data IO (D_{data}), and redo log IO (D_{log}) service demands.

4.2 Service Demands for Idle System

System utilization is measured while Oracle is open and available, but the system is at an idle state with respect to user transactions. Single instance and RAC both utilize background processes that monitor system health and collect statistics. The processes are modeled in an $idle$ class. A delay device is used in the model so that the throughput of this class is one.

4.3 Service Demands for Single Instance Oracle

Each transaction to be modeled is run in an isolated environment in order to establish both the service demands for each device and the write ratios for each segment accessed by the transaction. The write ratio is the ratio of block changes to block reads (both CR and CU). The write ratio is used in calculating the global cache service demands (see Section 4.4). To allow for independent throughput rates per class, the write ratio for a segment is provided by Eq. (2). Block access metrics can be obtained from a variety of techniques that involve Oracle (e.g., AWR Reports, v$segment_statistics, and TKPROF reports). The single instance service demands, $D^{si}_{i,r}$, for each class r and device i are adjusted by subtracting the resource utilization from the idle class as shown in Eq. (3). This equation follows the Service Demand Law [6].

$$w_i = \frac{\sum_{r=1}^{R} bc_{i,r} \times X_r}{\sum_{r=1}^{R} (cr_{i,r} + cu_{i,r}) \times X_r} \tag{2}$$

$$D^{si}_{i,r} = \frac{U^t_i - U_{i,idle}}{X_r} \tag{3}$$

4.4 Global Cache Coherence Service Demand

Our model estimates RAC overhead as a function of the global cache synchronization requirements for each node. Sasaki and Tanaka [8] define a method to estimate the inter-node requirements for global cache coherence. Equation (4) is the multi-versioning coefficient for segment i when the cluster consists of N instances/nodes. The number of global cache CR requests for segment i is provided by Eq. (5) and the number of global cache CU requests for segment i is provided by Eq. (6). The fairness factor used in these equations is four (see section 2.4.2). The impact of cluster size on the number of global cache requests is shown in Fig. 6. As the number of nodes in the cluster increases, the total number of global cache requests increases while the requirement per node decreases.

$$m_{i,N} = \frac{(1 - w_i)(N - 1)}{N + w_i - 1} \tag{4}$$

$$GCCR_{i,N} = \frac{m_{i,N} * (1 - m^4_{i,N})}{1 - m_{i,N}} \times bc_i \tag{5}$$

$$GCCU_{i,N} = \frac{N-1}{N} \left((m_{i,N})^4 \ \frac{(1 - w_i)(N - 1)}{w_i \times N - w_i + 1} + 1 \right) \times bc_i \tag{6}$$

Global cache requests involve either two or three nodes (see Section 2.4). A two-node exchange will occur with probability $\frac{1}{N-1}$, otherwise a three-node exchange will take place. The coordination requirements are not a function of the segment size, but solely of the volume of block changes with respect to the write ratio (because of our uniform access assumption). The RAC service demand calculation, $D^{\mathrm{RAC}}_{i,r}$, for resource i and class r is shown in Eq. (7).

$$D^{\mathrm{RAC}}_{i,r} = \begin{cases} D^{si}_{i,r} & \text{for } N = 1 \\ D^{si}_{i,r} + D^{2hop}_{i,r} \times GCT^r_N & \text{for } N = 2 \\ D^{si}_{i,r} + (D^{2hop}_{i,r} + \alpha) \times GCT^r_N & \text{for } N > 2 \end{cases} \tag{7}$$

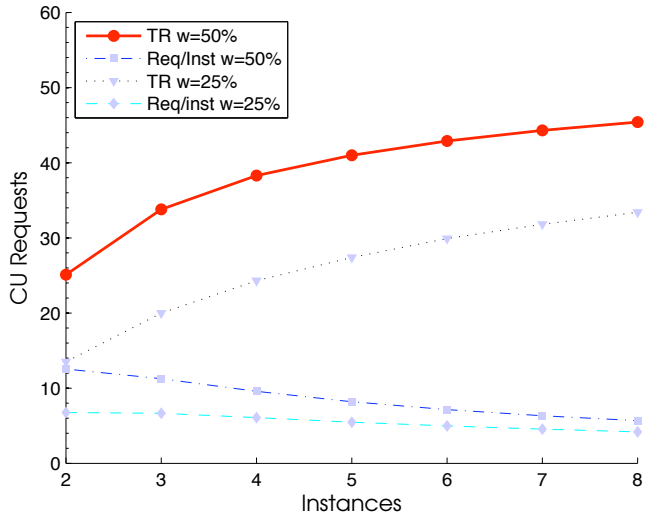

Figure 6: GCCU requests per 100 block changes. (TR = total requests for all nodes, Req/Inst = number of requests per instance.)

where $D_{i,r}^{2hop}$ is the service demand at device i and class r for each global cache request for the 2-hop case, $D_{i,r}^{3hop}$ is the equivalent service demand for the 3-hop case, and $\alpha = D_{i,r}^{3hop} \times pr_{3hop}$. Note that for $N > 2$, the $D_{i,r}^{2hop}$ term is always present regardless of whether a 2-hop or 3-hop is needed.

Equation (8) shows the calculation of the service demand $D_{cpu,r}^{2hop}$ at the CPU (similar calculations are performed for the data and log disk devices). All global cache requests in a two-node configuration are two-hop requests.

$$D_{cpu,r}^{2hop} = \frac{(2 \times U_{cpu}^{2}) - U_{cpu}^{si}}{GCT_{2}^{r}/\tau} \tag{8}$$

where U_{cpu}^{2} is the average CPU utilization of each node in the cluster in the 2-node case and U_{cpu}^{si} is the CPU utilization of the node in the single node case.

Equation (9) shows the service demand for a 3-hop case for the CPU device.

$$D_{cpu,r}^{3hop} = \frac{(N \times U_{cpu}^{3}) - (D_{cpu}^{2hop} \times GCT_{N}^{r}/\tau) - D_{cpu,r}^{si} \times X_{r}}{(pr_{3hop} \times GCT_{N}^{r})/\tau} \tag{9}$$

Each transaction class is run in isolation to capture per-class metrics. To approximate the behavior and service times for three-hop requests, each class is executed again under a 3-node configuration. A delay device is used in the QN model to represent the latency introduced via the network transfers between instances given that network contention is negligible.

4.5 QN Model

A multi-class closed QN model that uses MVA (mean value analysis) was used to predict performance metrics. More details on queueing networks can be found at [6].

5 Experimental Results

To validate the global cache coordination predictions and the QN model, several workloads were evaluated on a three-node RAC cluster running both Oracle 10gR2 and 11gR2. The database nodes were Dell T110 Poweredge servers with Xeon quad core X3430 2.4 GHz processors with 4 gigabytes of memory. Each database node used Oracle's Enterprise Linux Release 5 Update 5 as its operating system. To provide consistent measurements, the power management and turbo boost features of the Intel Xeon processor were disabled within the BIOS.

The storage server was a Dell Inspiron 530 with 4 gigabytes of memory. The storage server contains two disk drives: a 1 TB 7200 Western Digital caviar black SATA and a 160 GB Western Digital caviar blue SATA drive. The storage server used OpenFiler NSA 2.3 and hosted iSCSI separate targets for the Oracle clusterware (OCR and voting disks), datafiles, and redo log files. These targets are managed via automatic storage management (ASM). Each cluster node has two 1-gigabit network interface cards: one connects to the storage network and the other forms the cluster interconnect network. An Apple Macbook Pro with a 2.4 Ghz Intel core Duo processor was used to run the workload generator.

5.1 Database Description

The test database consists of six tables (see Fig. 7). An Oracle sequence is used to generate surrogate key values for the ORDERS table. This key forms a composite key in the ORDER_ITEMS table. A cache size of 2,000 is specified for the sequence, which guarantees adequate spacing to avoid index leaf block contention.

5.2 Transaction Description

Two transactions are modeled: an online order and an inventory report. The online order consists of read and write operations, while the inventory report is a read-only report. A mixed workload (i.e., both transactions) was used to validate the global cache request predictions. The online order class is the only one used for validating the QN model.

5.2.1 Online Order Transaction

The steps involved in this transaction are:

1. Look up customer information in the CUSTOMERS table using the supplied customer_id.
2. Retrieve a new sequence number and insert a new ORDERS row, using the sequence as the primary key.

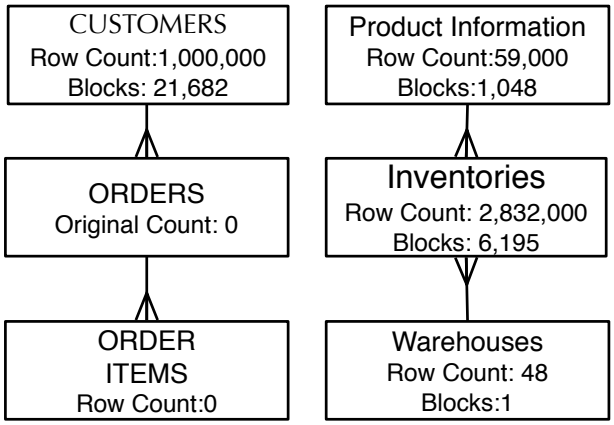

Figure 7: Database Entity-Relationship Diagram

3. Randomly select 25 products to add to this order. To avoid deadlocks, the product_ids are sorted by product_id. Then, for each product_id:

   - In a single SQL statement, verify there is adequate inventory in the warehouse assigned to this customer. If the inventory is adequate, update the inventory to reflect the number of items to be placed on this order.
   - Create an ORDER_ITEMS row to reflect the item being successfully added to the order

Once all 25 items are added to the order, a commit operation is performed. The following metrics were calculated using a TKPROF trace of the transaction: $bc_{inv,order} = 25$ and $cr_{inv,order} + cu_{inv,order} = 50$ (where inv = inventories). These results follow the behavior of UPDATE statements as described in Section 2.4.1. The INVENTORIES table only contains 6,195 blocks, which creates a high volume of contention in the RAC environment provided that uniform access is maintained.

The customer retrieval described above involves a full segment scan (no index is used). This choice allows the transaction to perform a normal volume of reads and more closely simulates a real world transaction with edit look ups, code validations, etc. During the testing phase, block contention within the CUSTOMERS segment was identified. To resolve this issue, three copies of the customer table were created in the database. During execution, each Pro*C load generator (see section 5.3) was assigned to one of the three tables. AWR reports confirmed that this eliminates the bottleneck while preserving the benefit of increasing the amount of work performed by each transaction.

5.2.2 Inventory Report

This transaction queries the INVENTORIES table for the quantity of all items within a given warehouse. The fact that this statement performs a full table scan to retrieve the data means that every block will be read exactly once. A TKPROF trace shows the following write ratios: $bc_{inv,invrpt} = 0$ and $cr_{inv,invrpt} = 6,289$.

5.3 Workload Generation

A combination of Pro*C, UNIX shell scripts, and PL/SQL packages were created to generate the workloads. A PL/SQL procedure performs all of the database work for the transaction. A Pro*C program calls the PL/SQL procedure using an exponentially distributed think time (Z_r) between transactions. A total of 54 Pro*C programs are started as part of each test run (48 for online order transactions and 6 for inventory report transactions). Connections are equally distributed amongst the active database instances (i.e., each instance has $54/N$ connections). Each transaction generator requests products from an unique warehouse, which eliminates row level contention within the INVENTORIES table. The inventories segment was tested (using a variety of SQL statements) to verify that each database block contains data for multiple warehouses so that the uniform block access assumption is maintained during testing. All the tests results shown were executed against an Oracle 10gR2 database.

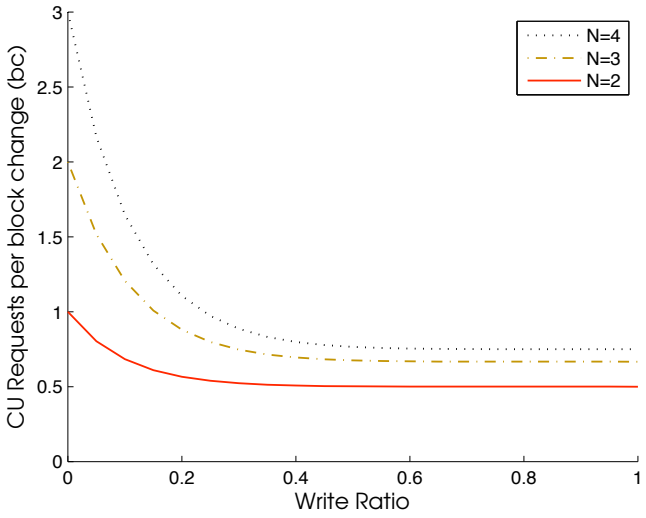

Figure 8: As the write ratio approaches 0, the number of CU requests per block change increases rapidly.

w_{inv}	N	X_{order}	X_{invreq}	$\frac{bc_{inv}}{\tau}$	$GCCU_{inv,N}$			$GCCR_{inv,N}$		
					Measured	Predicted	Relative % Error	Measured	Predicted	Relative % Error
0.50	2	12.0	0	308.24	151.41	154.76	2.21	149.87	152.22	1.57
0.50	2	23.3	0	596.16	294.96	299.31	1.47	289.93	294.4	1.54
0.50	3	11.98	0	310.24	202.21	209.47	3.59	199.92	201.53	0.81
0.50	3	23.14	0	595.68	393.78	402.20	2.14	386.87	386.95	0.02
0.41	2	12.0	0.02	310.35	152.11	157.06	3.25	209.88	212.67	1.33
0.43	2	23.3	0.03	600.32	295.20	303.13	2.69	378.78	385.98	1.90
0.41	3	11.98	0.02	311.09	202.24	214.76	6.19	281.26	276.97	1.53
0.42	3	23.11	0.03	609.31	392.01	419.09	6.91	508.16	523.31	2.98
0.33	2	12.0	0.05	311.23	153.00	160.88	5.15	295.63	299.93	1.45
0.29	2	23.3	0.14	632.80	302.47	332.45	9.91	646.72	704.61	8.95
0.33	3	11.96	0.05	309.25	205.66	225.22	9.51	398.61	378.67	5.00
0.29	3	23.13	0.13	635.89	409.85	479.33	16.95	855.5	879.22	2.77
0.19	2	12.0	0.16	320.03	171.16	183.94	7.47	504.13	540.84	7.28
0.20	2	23.2	0.29	666.40	321.04	378.20	17.80	900.79	1,080.47	19.95
0.19	3	11.99	0.16	324.19	250.79	290.55	15.86	635.05	640.48	0.85
0.20	3	23.11	0.28	666.91	471.47	584.01	23.87	1,170.62	1,271.45	8.61

Table 3: Global cache request predictions vs measurements for 2 and 3 node configs with varying write ratios. w_{inv} = write ratio for the inventory segment. bc_{inv} = # of block changes during the test period.

5.4 Global Cache Request Prediction Results

Tests were performed to measure the accuracy of the global cache message quantity predictions. The INVENTORIES segment will be reviewed, as it is the only segment for which the transaction generators provide uniform distributed access. The throughput of the online orders and inventory report transactions were varied to produce different write ratios (calculated by Eq. (2)). The volume of CR and CU global cache requests are predicted using Eqs. (4), (5), and (6). The results for 2-node and 3-node configurations are shown in Table 3. As the write ratio approaches zero, the relative error of the predictions increases. Figure 8 clearly shows that as the write ratio approach zero, the slope of the curve steepens. The number of nodes in the cluster amplifies this effect. The growth in the relative errors may be the result of Oracle optimizing some of its block writes as the transaction load increases, and thus changing the write ratio slightly.

5.5 Service Demand Calculations

A series of benchmark tests were performed to capture the service demands required by the model. These tests consist of running the online order transaction with an average think time of 4 seconds ($Z_{order} = 8$). First, a single instance test is performed (i.e., the other instances are shutdown). Measurements were taken using `iostat` and `vmstat`, as well as Oracle's Automatic Workload Repository reports. These measurements were used to compute the service demands for our single transaction class using Eq. (3). The test was repeated in a 2- and 3-node configuration with equations (8) and (9) providing the method of computing the service demands of individual global cache requests. Network timing was captured using the `ping` command to establish the latency between servers via the cluster interconnection network (see results in Table 4). The network service times are $NT_{block} + NT_{msg}$ for D_{net}^{2hop} and NT_{msg} for D_{net}^{3hop}. Each set of results were validated by comparing the measured response times from the load generator against the calculated response time provided by the Interactive Response Time Law [6]. In 93% of the test cases, the relative error between the measured and calculated response times was less than 10% (the exceptions had an absolute error less than 0.06 seconds).

5.6 QN Results

A multi-class closed QN model was built using the service demand calculations from Section 5.5. The model was solved for a series of workloads, varying the think time between transactions as well as the size of the database cluster (number of nodes). A few experiments were performed to validate the model's predictions. Table 5 compares the predictions of the model versus actual measurements in terms of throughput. The model does a good job of predicting the throughput of the system for think times of two or greater.

Event	D_{cpu}	D_{dd}	D_{logd}	D_{net}
D_i^{idle}	6	0.00	0	0
D_i^{si}	37	2	2	0
D_i^{2hop}	0.17	0	0	0.52
D_i^{3hop}	0.15	0	0	0.14

Table 4: Service Demands (in msec) measured with $Z_{order} = 8$

Z_{order}	N	Predicted X_r	Measured X_r
5	1	9.38	9.52
5	2	9.50	9.30
5	3	9.45	9.28
3	1	15.58	15.19
3	2	15.76	15.30
3	3	15.56	15.29
2	1	22.00	21.94
2	2	22.95	22.04
2	3	22.92	22.09

Table 5: Model versus measurements for varying think times and cluster sizes

For the single instance case, as the load on the system increases, the model understates the system throughput. The measurements show a 20% reduction in the service demand (0.0337 to 0.0266) for the CPU as the workload increased from 22 tps to 36 tps. One possible reason for this reduction is processing efficiencies within the Oracle kernel (e.g., combining block writes). A slight elevation in the service demand (from 0.0266 to 0.0289) was measured for values of X_r exceeding 36 tps. This is most likely a result of software contention within Oracle.

The RAC overhead calculated from Eqs. (8) and (9), shown in Table 4 were validated against an Oracle AWR report. The number of global cache requests were within 4% and the service times for GCCR and GCCU were within 0.25 milliseconds. However, the RAC service demand times rose as the workload was increased, suggesting that modeling this as load independent will cause our model to report better throughput and response time values under heavy workload conditions. Reviews of the AWR statistics suggest that the additional service time was spent waiting for "gc buffer busy" events.

The model was used to illustrate the scalability of RAC for our given workload. Figure 9 shows the throughput predictions for varying cluster sizes. Figure 10 shows the ratio of throughputs relative to the single instance case, i.e., $X_0(1)/X_0(N)$. If perfect linear speedup were achieved

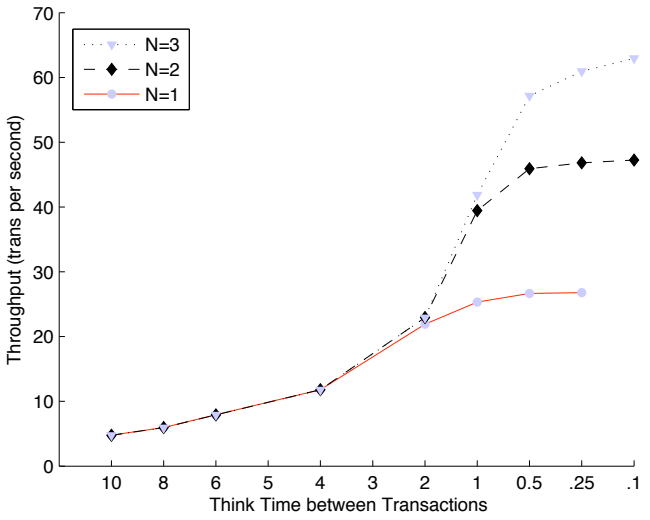

Figure 9: QN Model predictions for fixed number of workers (48) where N is the number of nodes.

(i.e., $X_0(N) = N\ X_0(1)$), this ratio would be $1/N$. These values, i.e., 0.5 and 0.33 for N=2 and N=3, respectively are shown by the dashed reference lines in the figure. As the figure indicates, the ratio $X_0(1)/X_0(N)$ tends to a value slightly higher than the linear speed up threshold due to the RAC overhead. While not the worst case scenario (which would be all instances contending for a single block within the segment), the workload used for testing was designed to promote global cache requests, and therefore, amplifies the RAC overhead.

6 Concluding Remarks

Oracle RAC and the concept of grid computing is redefining system architectures to achieve high availability and scalability. The goal of this research was to develop a model to accurately predict performance in a RAC environment, using workload characteristics from a single instance environment as input. RAC's overhead is directly related to block read/write ratio, block contention, and cache coherence messaging demands. The method proposed in this paper accurately predicts the volume of global cache requests for different configurations (cluster size and write ratios) for write ratios greater than 0.3. The QN model, built up from the global cache workload predictions, works well for light to moderate workloads in predicting system performance. However, the lack of modeling for block contention and other software bottlenecks result in the model overestimating throughput under heavy workload conditions.

Future work in this area includes reviewing the new Oracle 11g cache fusion protocol, which is currently not well-documented. Oracle's performance capture (AWR report)

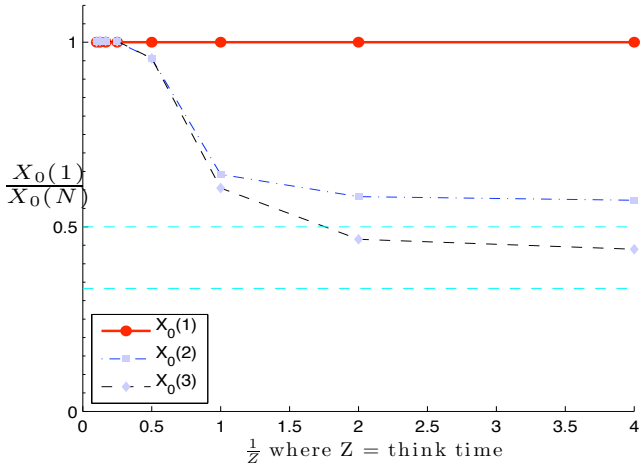

Figure 10: Ratio of the throughput of a single sever versus an N node configuration.

and workload replay capabilities (real application testing) allow for recording key metrics of a given workload. This information should provide a manner to formulate service demands and build QN models of more complex transactions. Techniques to enhance the workload characteristics so that block-based contention can be modeled will allow the model to be more accurate under heavy workload conditions. Priority-based modeling for the global cache service daemon (LMS) will be evaluated as a future enhancement.

References

[1] Mike Ault, Madhu Tumma. Oracle 10g Grid & Real Application Clusters. Oracle 10g Grid computing with RAC. Rampant Techpress 2004.

[2] E.W. Dempster et. al.,"Modeling Parallel Oracle for Performance Prediction," Distributed and Parallel Databases,Kluwer Academic Publishers, 13, pp. 251–269, 2003.

[3] Julian Dyke. Inside RAC presentation. www.juliandyke.com.

[4] Gopalakrishnan, K. Oracle Database 10g. Real Application Clusters Handbook. McGraw-Hill, New York. 2007

[5] Kyte, Thomas. Expert Oracle Database Architecture 9i and 10g Programming Techniques and Solutions. [New York]:Thomas Kyte, 2005.

[6] D.A. Menascé, V.A.F. Almeida, and L.W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*, Prentice Hall, Upper Saddle River, 2004.

[7] Riyaj Shamsudeen. Boot Camp: RAC Performance Tuning.

[8] S. Sasaki and A. Tanaka, "Analytical Model of Inter-Node Communication under Multi-Versioned Coherence Mechanism," 2009 IEEE International Symposium on Parallel &Distributed Processing, pp.1–8.

[9] Madhu Tumma,"Oracle RAC: Administrative and Performance Challenges" IOUG Collaborate07 conference, 2007.

[10] Cache Fusion: Extending Shared-Disk Clusters with Shared Caches. Proceedings of the 27th International Conference on Very Large Data Bases, pp. 683-686, 2001.

[11] "Oracle Database PL/SQL Language Reference 11g Release 2(11.2) E10472-05". September 2009.

[12] "Oracle Database Concepts 11g Release 2(11.2) E10713-04". September 2009.