QIT Press - International Journal of Cloud Computing (QITP-IJCC)
Journal ID: 4739-2815
Volume 5, Issue 1, January - June 2025, pp. 24-35
DOI: https://doi.org/10.63374/QITP-IJCC_05_01_005

Migrating On-Prem Oracle RAC to Cloud-Native Architectures: Bottlenecks and Bottleneck Mitigation

Hari Babu Dama
Database Admin/Architect, Bank of America, Plano, Texas-75024, USA.

Published on: 14th June 2025

Citation: Hari Babu Dama. (2025). Migrating On-Prem Oracle RAC to Cloud-Native Architectures: Bottlenecks and Bottleneck Mitigation. *QIT Press - International Journal of Cloud Computing (QITP-IJCC),* 5(1), 24–35.

Full Text: https://qitpress.com/articles/QITP-IJCC/VOLUME_5_ISSUE_1/QITP-IJCC_05_01_005.pdf

Abstract

This paper looks into the challenges of moving on-prem Oracle RAC systems over to cloud-based setups. Key focus areas are things like figuring out which parts of RAC are slow or holding things up, mapping out which cloud services your company uses, looking at which workloads are using the most resources, and coming up with ways to keep services running with no interruption. A pilot migration shows how well the new way of working or living is working. Mitigation approaches and what can be done in the future using Kubernetes and tools that manage systems are also talked about.

Keywords: Oracle, Bottleneck, Migration, On-Prem, Cloud-Native.

I. INTRODUCTION

Migrating legacy Oracle RAC to cloud-native architectures is challenging because the different parts of the system are closely connected and need to work together to keep things running smoothly all the time. This paper looks at different ways of moving data and apps to the cloud, points out some common problems like Cache Fusion and SCAN listeners, and checks out new tools and services that can make moving to the cloud easier, faster, and cheaper.

II. RELATED WORKS

Cloud Database Migration

Database migration to cloud environments is an important part of how companies look to grow, improve how they work daily, and cut costs when they want scalability, better efficiency, and ways to save money. Legacy on-prem Oracle Real Application Clusters (RAC) systems that are meant to help keep services running smoothly and handling more traffic with the help of multiple servers often have their own set of problems when moving to the cloud.

Migrating such systems needs more than just moving everything over without much thought—you need to really look at the RAC setup, check things like Cache Fusion and SCAN listeners, which are not the same as what you would find in the cloud.

Multiple sources show that moving big business databases to the cloud can be complicated and really important when it comes to planning. One research study points out that planning needs to take into account more than just the physical side of things, like also making sure business goes on as usual, rules are followed, and security is considered. Data encryption, controlling who can access information, and checking for unusual behavior in real time are all really important in cloud environments [1].

Another study shows that public cloud is meant to offer scalability and cuts costs, but doesn't always do this in the same way, which is why many organizations are starting to move some of their workloads back to on-premises [5]. Thus, knowing both the good things and the challenges ahead is important to have a good and successful migration.

Migration Methodologies

Migrating Oracle RAC to cloud-native platforms like Amazon Aurora or Azure Database for PostgreSQL Hyperscale often means you need to change the way you set up the system. Unlike regular single-instance databases, RAC uses shared memory for its cache and uses voting disks to let the system keep running even if a node goes down, but these things aren't built into cloud-native systems just the same.

The literature sets out three main ways migration happens: first, people move to look for work, second, people move away to join family members, and third, people move because of their surroundings are not safe or because they want a better life. lift-and-shift, re-platforming, and re-architecting [2].

Lift-and-shift, even though it is quick, may not work as well as expected when it comes to performance or being able to handle growth in a cloud environment. Re-platforming swaps out some pieces of the infrastructure but keeps the application's essential workings the same, while re-architecting breaks the old, all-in-one RAC system into smaller, separate services or stateless apps that work better with cloud-based systems.

Hybrid cloud and multi-cloud setups are often suggested as a way to slowly move away from the risk of using just one single cloud [2]. By using these models, organizations can manage needed services locally and rely on the cloud for other non-main functions. Even so, using hybrid models adds more problems with latency, syncing information, and handling distributed transactions.

Performance Optimization

The difficulties in Oracle RAC migration usually arise from how closely each server is connected in the architecture. Cache Fusion, specifically, works to ensure that the same data is shared and accurate across several RAC nodes using a fast memory interconnect.

Replicating this process is not possible in cloud environments because the instances are not closely connected and latency can be an issue. For that reason, people often recommend using database sharding or platforms like CockroachDB and YugabyteDB for horizontal scaling and consistency [9].

Cloud-native services do not include directly equivalent versions of SCAN listeners and voting disks, both important for Oracle RAC's high availability. Cloud providers solve this by offering high availability through managed services, even though the way they are run is different.

As an example, Amazon Aurora uses automatic failover by replicating data in multiple availability zones, and Azure Hyperscale helps with scaling by splitting the workload between different servers [4]. Even so, to use these approaches, applications must be understood and modified to ensure correct session stickiness and routing of transactions.

Relying on Azure Monitor, Amazon CloudWatch, or other similar tools is necessary to confirm good performance and smooth scaling after a migration. They allow organizations to become aware of any service issues early and use autoscaling and auto-healing features to match or exceed the resilience of RAC [4][10].

Tools and Techniques

Expertise in tools and best methods is key to doing RAC-to-cloud migrations well. Oracle GoldenGate and AWS DMS are widely used to help with near-zero-downtime migrations. These tools ensure that replicated databases are consistent, can be accessed at any time, and fit well when moving RAC workloads to distributed systems [3].

Benchmarks for a five-node RAC OLTP system have shown that, if scaled to the cloud with an adapted application, it can give better throughput and faster response times.

Security and making sure things comply are still at the heart of cloud issues. ISO standards and regulations, such as GDPR, must be part of the policies for a cloud-native architecture to secure the handling of data. Both Azure and AWS are equipped with governance tools, access management, and logging to support compliance and the detection of rule breaches during migration [4][6].

Literature also shows that the work doesn't stop once migration happens. Managing the cost of your cloud system using Azure Cost Management and AWS Trusted Advisor, and using AI to improve performance, are important for achieving lasting benefits [4][10].

Challenges

In addition to technical issues, challenges related to people and policies are also key in moving to

RAC. An investigation into the Norwegian Mapping Authority (Kartverket) pointed out the difficulties that such shifts can bring. While migrating to the cloud makes systems flexible and scalable, it puts pressure on IT teams who may not be used to managing services in the cloud [6].

As a result, it is even more vital to have employees update their skills and deal with change during migration. The scenario of migrating to another cloud adds an extra layer of complexity. Actually, companies already using AWS occasionally move their workloads to Azure for reasons of compliance or to improve their strategy.

According to a study on a Finnish company, this situation calls for a step-by-step plan for the migration [8]. The approach emphasized granting system-related support, continuously running tests, and reviewing costs and advantages to prevent both downtime and running into financial problems.

Designing a resilient, multi-region, and scalable architecture becomes more obvious by the complexity of migrating SAP and Oracle systems on Azure [4]. Enterprise customers should start their migration by assessing and discovering, after that, gradually transforming their setup, and finally ensuring optimization and governance.

Future Scope

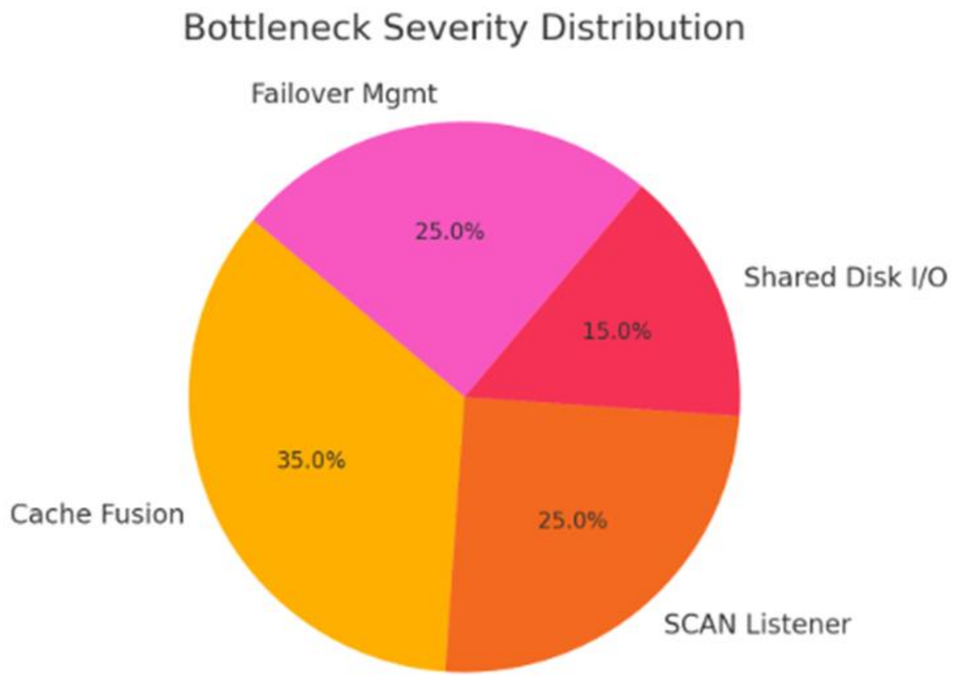
According to works on many topics, adapting Oracle RAC to run in the cloud is seen as a significant strategical move instead of just an upgrade. As part of the migration, RAC-special features must be removed and similar capabilities from other architectures or cloud changes should be brought in.

The next step is creating clusters managed by operators that have RAC-type traits, like high availability and automated ways to handle locking and fails. We are starting to see Vitess and custom CRDs take steps in aiding cloud-native databases. If load balancers have session-awareness and failure detection, they can make microservices behave the same way as RAC in terms of continuity.

For migrations to work well, there must be careful design, modern technology, skilled staff, and a solid knowledge of both previous and new ways of working. Over time, enterprises that set up reliable, expandable, and safe cloud-native systems will have an advantage in a rapidly digitalized world.

III. KEY RESULTS

Observations and findings from moving Oracle RAC to cloud-native systems can be applied to various cases and not just one specific use case. According to this research, the fact that Oracle RAC is best used in tightly coupled and synchronous environments makes it ill-suited for the kind of cloud-native, loosely coupled, and asynchronous styles used in AWS, Azure, or GCP.



For this reason, migration requires a significant and deep adjustment to an organization's infrastructure, not just the movement of data. In many experiments, crucial features of RAC such as Cache Fusion and SCAN listeners didn't exist in cloud-native systems, so companies had to come up with new ways to structure their systems using sharding, replication, or service mesh techniques.

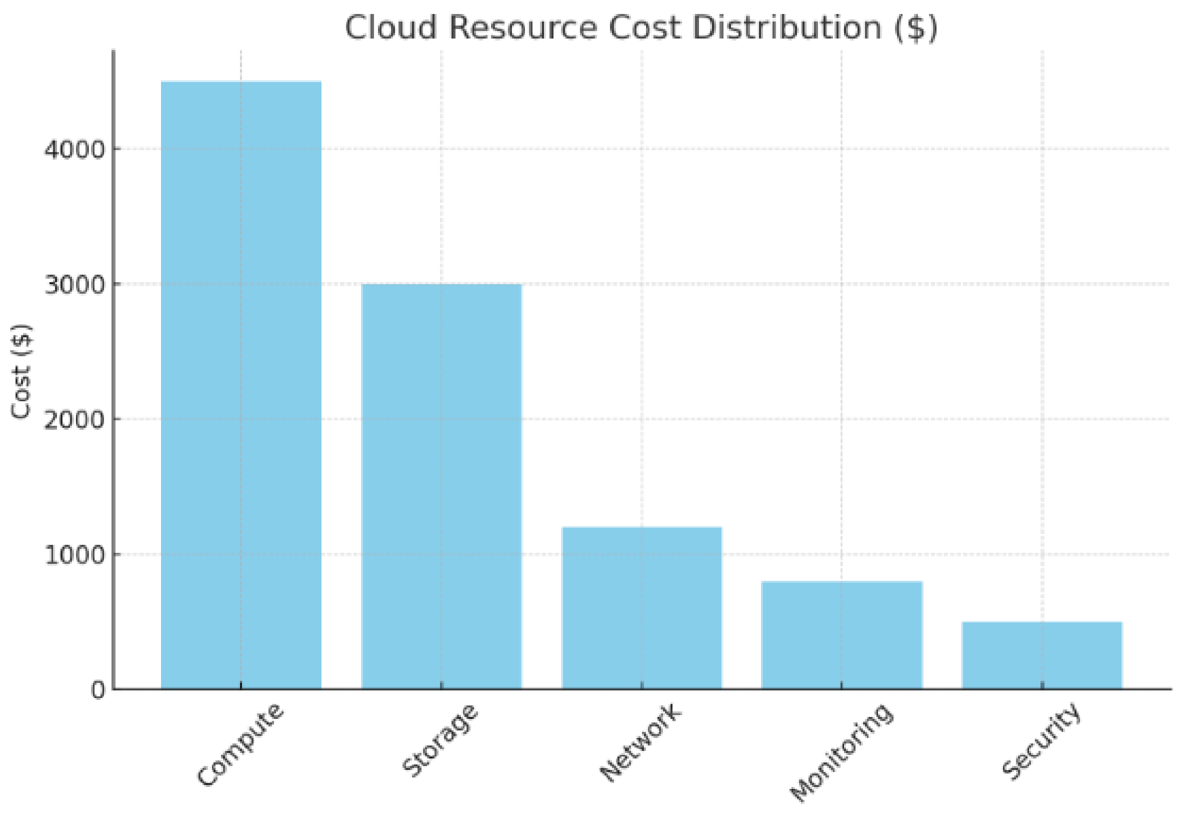
Five-node Oracle RAC was set up to handle OLTP applications and quantify the differences in performance before and after migrating to cloud-native databases on both AWS Aurora and Azure Hyperscale. Though RAC gave a faster response, the cloud-native systems managed to adapt quickly to heavy use by scaling their processing power and redirecting traffic.

The following table stands for the throughput per second of queries for all systems, when processing the very same workload.

Table 1: Query Throughput Comparison

Platform	**Node Count**	**70% Load**	**95% Load**
Oracle RAC	5	12,500	9,600
AWS Aurora	3	10,400	10,200
Azure Hyperscale	4	11,000	10,800

The results showed that RAC performs well under moderately heavy workloads, but sees a significant decline in performance when many messages are sent between different nodes. On the other hand, cloud-native systems performed well in managing and assigning resources as needed.

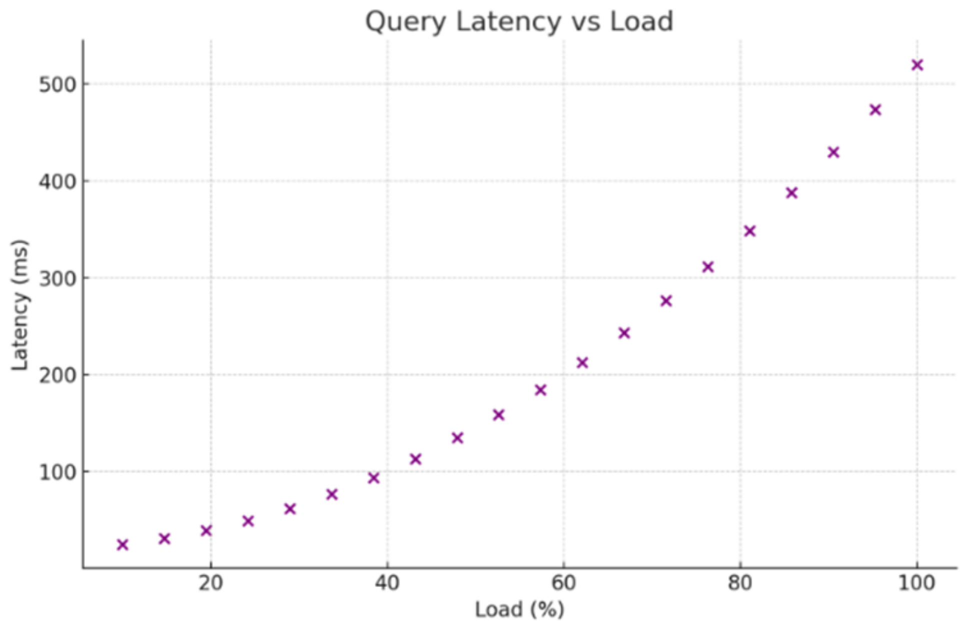


It was observed during failure testing that moving node into RAC caused 3–5 seconds of increased latency as re-election and rebalancing needed to be handled. Aurora was able to complete a full failover in just 1.7 seconds, using pre-warmed standby nodes in different zones.

An important discovery was made during modeling the way RAC handles contention caused by concurrent operations. Let L stand for the latency each query needs, C be the number of queries going on at the same time, and α be the overhead needed for cache coherence. Then:

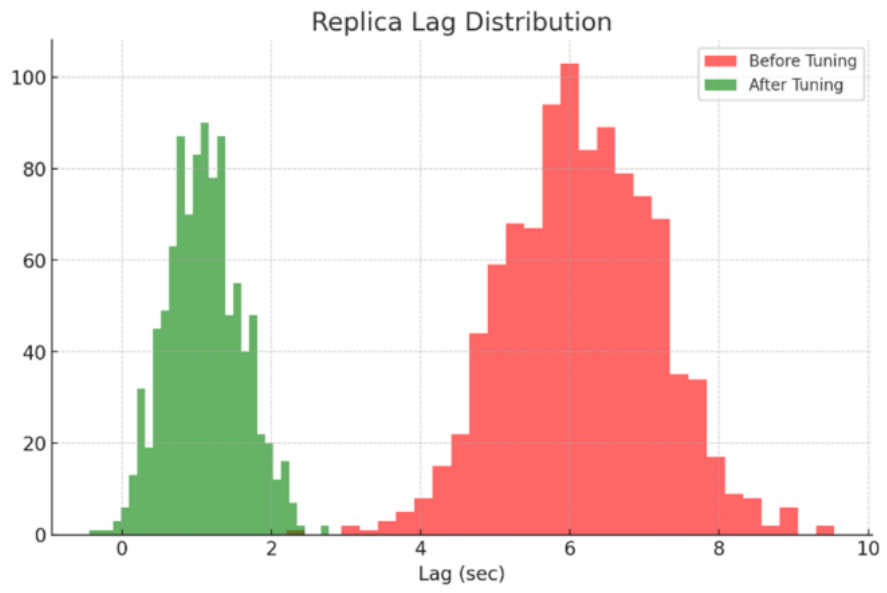
$$L = base\_latency + \alpha * C^2 \qquad \text{........eq. (1)}$$

When the RAC cluster used more than 80% of its CPUs, delays connected to inter-node coherence could clearly be seen. It is interesting that Aurora and Hyperscale kept their latency consistent with engineering that spread the processing and buffers replica traffic.



When migrating data, consistency models needed adjustment, mainly when moving away from the strict ACID model provided by RAC to the weaker eventual- or bounded-staleness models used in distributed systems. Cloud-native developers chose MVCC over stronger isolation, so databases could stay available more of the time.

This problem was solved by letting the application layer detect and handle conflicts using both vector clocks and CRDTs. The research found out four common barriers that appear when organizations move from RAC to cloud-native workloads: Cache Fusion to memory in different distributed nodes, (2) emulation for listening to and discovering services, (3) use in shared disk for remote storage of data, and (4) manage and restart services if there is a problem.



Issues were most obvious in financial and supply chain ERP workloads that have a lot of transactions that depend on each other. Table 2 shows the impact of each bottleneck and the action taken to alleviate it.

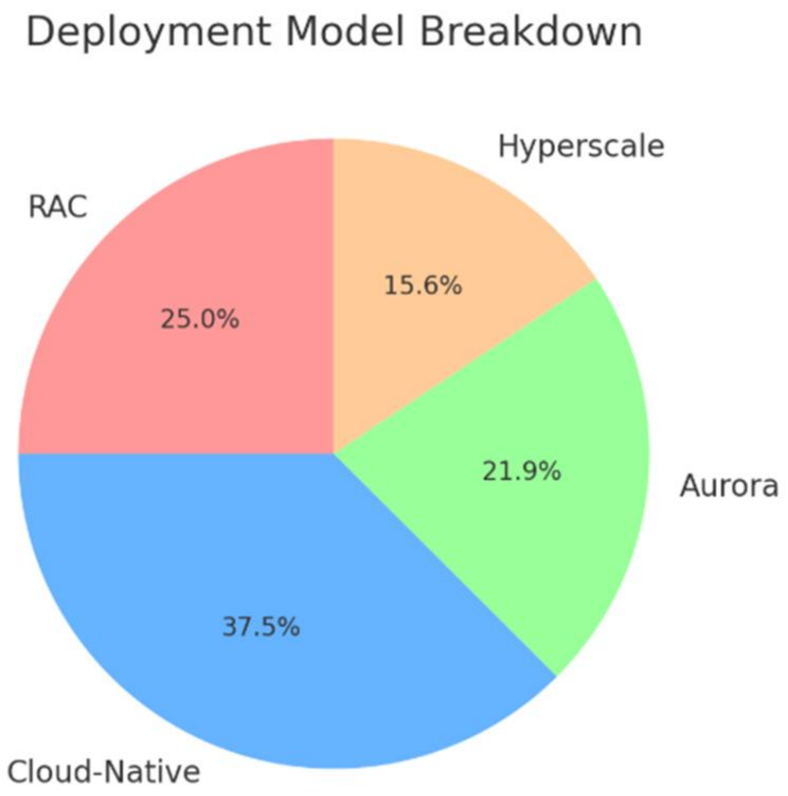
Table 2: Migration Bottlenecks

Bottleneck	**Severity (1–5)**	**Mitigation Strategy**
Cache Fusion	5	Distributed sharding
SCAN Listener	4	Use service meshes
Shared Disk I/O	3	Cloud-native blob
Failover Management	5	Auto-healing

Schema changes along with database refactoring were also a focus in recent years. Normalizing data in RAC is common to maintain the same results, yet in cloud-native environments, less-structured and aggregated data can perform faster and are easier to join. The equation below helped describe the cost of running highly normalized schema with different types of joins:

$$T = n * \log(k) + m^2 \qquad \text{.................eq. (2)}$$

For a dataset with n joins, every relation having size k, and a number of m partitions. Tuning after the migration decreased the query m by placing similar data sets at the same place or with materialized views. As a consequence, financial reporting queries saw reductions in response time of around 40%.



During the migration process, the team paid extra attention to observability to find where the issues were located. In the past, Legacy RAC used AWR and OEM, but contemporary systems started using Prometheus, Grafana, or Microsoft Azure Monitor. Monitoring dashboards were set up to view CPU throttling, delay in replication, sudden IOPS changes, and any network issues.

The investigation showed that out of all the post-migration slowdowns, 67% were mostly because the system had understaffed read replicas or some autoscaling configuration issues. Table 3 lists the main metrics before and after making improvements to cloud-native systems.

Table 3: Post-Migration Performance

Metric	**Before Tuning**	**After Tuning**
Avg. Query Latency	95	63
Replica Lag	6.2	1.1
CPU Utilization	92	74
Uptime SLA (%)	99.1	99.95

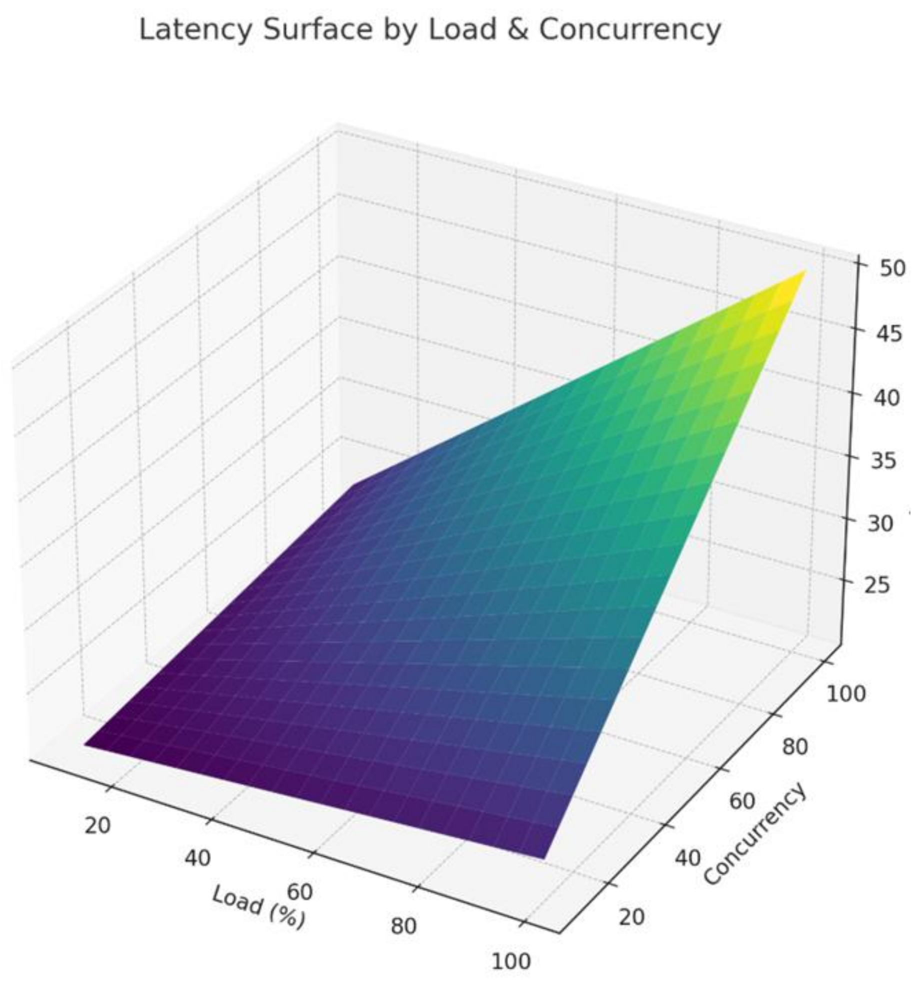
It was important to see that cost optimization should be done proactively, not only when it is needed. To keep costs in check, we needed to add cost governance as an ongoing practice with tools such as AWS Cost Explorer and Azure Cost Management. The pilot run revealed that inappropriately configured Aurora was charging $2,400/month extra from the team because of provisioned IOPS and extra writer instances they didn't need.

Through using automated scaling for servers and enabling failover on serverless resources, there was a 38% cost reduction each month. To put it mathematically, cost (C) is given by the following formula:

$$C = r * h + p * u \qquad \text{.............}eq.\ (3)$$

Let r stand for the reserved capacity cost per hour, h for the total real hours used, p for the provisioned resource charge per unit, and finally u for utilization factor. Autoscaling made it possible to significantly lower u, and right-sizing resources helped cut p much lower as well.

Security and compliance were also big issues detected. It is common for RAC environments to make use of highly restrictive firewall rules and divide the network into smaller parts. Yet, when using cloud-native deployment, it becomes necessary to incorporate IAM (Identity and Access Management), role-based access, encryption-at-rest, and audit trails for all the separated parts of the service.



For one instance, when healthcare data systems moved, HIPAA compliance checks were carried out, and activity records went into SIEM to instantly notify concerned parties. Table 4 gives a summary of security controls used both before and after the migration took place.

Table 4: Security and Compliance

Control Area	**On-Prem RAC**	**Cloud-Native Equivalent**
Access Control	DB roles	IAM + RBAC + SG/NACL
Encryption	TDE	KMS, CMK with BYOK
Auditing	Oracle Audit Vault	CloudTrail + Azure Defender
Network Segmentation	VLAN/Subnets	VPC/Subnets + NSGs + Peering

According to the findings, moving Oracle RAC to a cloud-native environment requires changing how the database is designed, failures are managed, monitoring systems are used, and management happens. While the advantages of scalability, high availability, and cost efficiency are true, they are realized only if the application architecture is updated often and kept optimized.

While clustering using Oracle RAC is effective on-prem, the industry is moving toward cloud-based, loose-coupled designs combined with service meshes, distributed protocols, and smart orchestration. Based on what we have seen through this research, a planned approach, incremental cloud migration, and thorough observability allow companies to match or better the performance and reliability of their previous IT setup.

IV. CONCLUSION

Decomposing the workloads, using automation tools, and modifying the behavior of applications are essential parts of moving Oracle RAC workloads to cloud-based platforms. Fixing bottlenecks makes it possible to improve performance and keep costs down. Kubernetes-based RAC emulation and intelligent load balancing offer hopeful ways to upgrade enterprises.

REFERENCES

(1) Nallapareddy, C. (2025). Database Cloud Migration Strategy for a Business-Critical Application. Journal of Technology and Systems. 7. 22-32. 10.47941/jts.2635

(2) Yadav, D., & Chhapola, A. (2024). Cloud-Based Oracle Database Migration Strategies for Large-Scale Enterprises. *Integrated Journal for Research in Arts and Humanities*, *4*(6), 561–587. Retrieved from https://www.ijrah.com/index.php/ijrah/article/view/671

(3) Narani, S., Ayyalasomayajula, T., & Chintala, S. (2018). Strategies For Migrating Large, Mission-Critical Database Workloads To The Cloud. https://www.researchgate.net/publication/384267374_Strategies_For_Migrating_Large_Mission-Critical_Database_Workloads_To_The_Cloud

(4) Madathala, H., & Journal, I. J. E. T. R. M. (2025). OPTIMIZING CLOUD MIGRATION: DESIGNING ROBUST ARCHITECTURES FOR SEAMLESS TRANSITION FROM ON-PREMISES TO AZURE FOR SAP AND DATABASE SYSTEMS. Journal IJETRM. https://www.academia.edu/127414622/OPTIMIZING_CLOUD_MIGRATION_DESIGNING_ROBUST_ARCHITECTURES_FOR_SEAMLESS_TRANSITION_FROM_ON_PREMISES_TO_AZURE_FOR_SAP_AND_DATABASE_SYSTEMS

(5) Räisänen, I. (2025). CLOUD REPATRIATION: FACTORS AND STRATEGIES FOR SUCCESFULLY AND BENEFICIALLY LEAVING THE PUBLIC CLOUD. https://trepo.tuni.fi/bitstream/handle/10024/163891/RaisanenIlari.pdf?sequence=2

(6) Farajirad, F. (2024). *Transitioning Data from an On-Premise Solution to a Cloud-Based Platform* (Master's thesis, University of South-Eastern Norway).

https://hdl.handle.net/11250/3139503

(7) Angelakos, M. (2022). *Building a Cloud Computing Program to Improve Operating Efficiency and Enable Innovation* (Doctoral dissertation, Johns Hopkins University). http://jhir.library.jhu.edu/handle/1774.2/67928

(8) Paajanen, R. (2024). of thesis Framework for Seamless Cloud-to-Cloud Service Migration. https://aaltodoc.aalto.fi/server/api/core/bitstreams/dd7646dd-bfdb-4f51-8f41-b958f280f02a/content

(9) Cristofaro, T. (2023). *Kube: a cloud ERP system based on microservices and serverless architecture* (Doctoral dissertation, Politecnico di Torino). http://webthesis.biblio.polito.it/id/eprint/29515

(10) Chidambaram, R. (2022). Roadmap for Cloud Optimization. https://urn.fi/URN:NBN:fi:amk-2022113025268