WCRG WORLD COMET RESEARCH GROUP

World Conference on Modern Engineering, AI, and Innovation
WCMEAI25-Conference Proceeding | IJETMR
Doi: 10.67228/30715636/WCMEAI-2025P108
PP. 119-137

Original Article

Diagnosing and Resolving High CPU Utilization in Cloud Databases

Shiva Santosh Allenki

AWS Cloud Support Engineer at Amazon Web Services, USA.

ABSTRACT

High CPU use in cloud-based databases is a usual performance problem that affects how rapidly applications respond, how well they can grow & how well the entire architecture works. This study looks at the main causes, distinguishing tools & perfect approaches for lowering extravagant CPU utilization in virtualized database inauguration. The study depicts various contributing factors via observational analysis & controlled reproduction, encompassing inefficient query implementation plans, absent or outdated indexing, heightened transaction concurrency, improperly adjusted their resource allocation & suboptimal framework of caching or connection pooling mechanisms. The diagnostic process uses tools for monitoring their performance, profiling queries & analyzing workload patterns to find performance hotspots with great accuracy. Continuous monitoring application indications like CPU load averages, query latency & input/output production is more important for finding many problems early on. The suggested system significantly strengthens these resource efficiency via the use of adaptive workload management, query optimization & the automated scaling approaches, all while preserving data integrity & throughput. The experimental findings from the case study indicate that intelligent query optimization & load balancing may reduce CPU use by over 40%. Dynamic scaling and caching its improvements, on the other hand, make performance more steady when there is a lot of demand. The research emphasizes the need of conveying their database setups with workload factors instead of depending on their static stipulations The findings show that cautious diagnosis & systematized their enhancement of cloud databases may greatly increase their performance consistency, cost-effectiveness & the scalability. By keeping an eye on their database systems all the time, optimizing them based on their information, and automating processes in the best way, businesses can keep them functioning very smoothly. This manner, the systems can simply deal with many changes in workload. This research offers significant insights for database administrators, cloud architects & DevOps teams focused on creating resilient, resource-efficient & more scalable cloud infrastructures.

KEYWORDS

Kernel Profiling, Cloud Infrastructure, AI-Powered Diagnosis, Performance Bottlenecks, Kernel Instrumentation, Anomaly Detection, Distributed Systems, Observability, KernelSight-AI.

WCRG
Global Research
ISSN: 3071-5636
PP: 119-137.

I. INTRODUCTION

Cloud databases have become the backbone of modern digital infrastructure, enabling anything from simple web services to huge, data-driven commercial systems. Companies that want to be more agile & save money should consider them since they are more scalable, flexible & managed. Still, this ease of use makes it very hard to manage and improve the performance of databases in a shared, virtualized & very dynamic environment. High CPU use is a huge problem in this case since it might cause latency spikes, lower throughput & very less efficient operations.

In traditional on-premises setups, database administrators (DBAs) have full control over the hardware & workload, which makes it easier to find many resource bottlenecks. On the other hand, cloud solutions hide actual infrastructure, which makes things even more complicated. Getting the best performance is still a moving target since these workloads change all the time & different tenants use the same computer resources. When CPU use is very high, whether it's steady or intermittent, it often means that there are many problems with the way queries are structured, indexed, or managed. To solve this challenge, we need a deliberate, proactive approach that uses their automation and smart monitoring instead of just fixing things when they go wrong.

This paper investigates the growing problem of high CPU use in these cloud-based databases, focusing on its causes, effects & efficient methods for identification & the resolution. It calls for the creation of a systematic framework that can automatically find, evaluate & fix CPU bottlenecks in actual time, which would reduce the need for human involvement & save expenses.

1.1. Challenges

Managing performance in cloud databases is becoming more complicated because of a number of problems that are all related. The first & most obvious reason is that workloads change. Cloud databases are different from static on-premises systems because they are affected by how users change their behavior over time. When there are a lot of queries, the load might quickly rise, which means that more computing power is needed. Cloud elasticity makes it easier to scale these resources, but this process doesn't happen right away. During these changes, the CPU may get too full, which might cause their queries to take longer and throughput to go down.

Virtualization overhead is another problem that makes things harder. Cloud databases usually run on their virtualized hardware or in containerized settings. Virtualization makes it easier to keep things separate & flexible, but it also puts an additional layer between the software and the hardware. This uses CPU resources that don't immediately help with task execution. When several other virtual machines or containers are fighting for the same physical cores, it might make CPU use worse.

Another common reason is inefficient query patterns. Poorly set up SQL queries, no indexing & random joins may all greatly increase CPU use. A single query that scans an entire table on a huge dataset can use up all of the CPU resources, which would slow down many other queries that are

ISSN: 3071-5636
PP: 119-137.

running at the same time. When many comparable searches run at the same time, they may quickly use up all of the computer resources that are available.

The fact that cloud systems share resources makes these difficulties more worse. Databases in multi-tenant environments share hardware resources with other applications. Even though resource isolation limits, noisy neighbors—applications that consume a lot of CPU—can nevertheless have an effect on their database performance. It's harder to figure out why the CPU is being used when there are shared dependencies, since performance metrics are influenced by both internal database activities & things outside of the shared environment.

From a business point of view, these tech problems cause a clear drop in their performance. Applications show longer transaction queues, higher query latency & lower throughput. This has a direct impact on the experience of the end user, and in business settings, it might lead to lost sales or unhappy customers. Also, cost inefficiency is becoming an issue. Most cloud providers charge based on how much computing power you use, so if your CPU use is high for a long time, your operational expenses go up. Frequent, although temporary, pierces might lead to huge cost increases.

In short, to get the most out of the CPU in cloud databases, you need to find a balance between a lot of modifying elements, such as workload changes, virtualization overhead, query efficiency & shared restrictions. Traditional monitoring techniques could show high CPU metrics, but they don't always find the actual problems that are causing them. This shows that we need advanced symptomatic tools that can not only find performance problems but also understand & react to them in actual time.

1.2. Problem Statement

This article looks at the problem of cloud-hosted databases that have constant or sporadic spikes in CPU use, which makes them very less responsive, causes query response times to vary & raises operational expense. There are several other ways that high CPU use might show up: a processor that is always full under normal workloads, or sudden spikes in CPU activity caused by their certain requests, background processes, or system maintenance tasks. No matter how it's set up, the end consequence is the same: slower response times, delayed transactions & higher costs.

Most of the time, fixing CPU issues means manually optimizing these databases & setting fixed resource limits. These tactics work well in circumstances that are easy to foresee, but they don't perform well with modern, changing workloads. DBAs may change framework settings, add more indexes, or change queries based on how the system is performing in the static setups. Workloads vary often in cloud systems & this is typically due to how users act, changes to applications, or the latest ways of taking in their information. What worked as a perfect yesterday may not work in the present day.

Also, static resource allocation solutions, such as setting aside a certain number of CPU cores or RAM, are typically a waste of these resources. When workloads are low, resources aren't utilized as

ISSN: 3071-5636
PP: 119-137.

much as they may be, which costs money. When workloads suddenly increase, there aren't enough resources, which causes throttling & delays. As a result, businesses face a paradox: too much provisioning wastes money, while too little provisioning hurts performance.

Things become worse since cloud systems aren't really obvious. It's tougher to link CPU data to particular reasons since we don't fully grasp how the architecture works. Is excessive CPU use driven by inefficient internal queries or by an outside activity that is competing for shared computer resources? Standard monitoring dashboards may indicate symptoms of problems, but they often don't have the depth or expertise to figure out what's really going on.

Because of this, we need a flexible and adaptable plan to discover and repair excessive CPU utilization in these cloud databases right once. This plan should go beyond just diagnosing the problem and involve continuing optimization.

1.3. Motivation

Because maintaining cloud-native databases is continually changing, we need a systematic technique to find out why the CPU is running high and what to do about it. The cost of being inefficient increases a lot when companies utilize more and more technologies that require a lot of data. When the CPU use gets greater, it not only makes the program slower, but it also starts a chain of additional problems, such slowness, higher costs, and maybe breaching service level agreements (SLAs).

With an anticipatory framework, teams would have access to real-time information about performance issues so they can detect and address them before they become worse. Smart monitoring systems might search for early warning indicators, including queries taking longer to process or CPU-to-I/O ratios that aren't typical, instead of waiting for CPU saturation to affect users. Moving from reactive to proactive management is a big step forward for operational resilience.

Also, adaptive optimization is quite too important. Because cloud workloads are always changing, performance hone must also change. Machine predictive strategies may look at previous performance data to figure out when CPU demand will go up and adjust setups, query processing plans, or caching systems on their own. This resilience means that these database systems can remain more efficient in a number of conditions without needing aid from people.

This idea becomes more effective when machines are used.. Managing performance by hand is time-consuming, prone to mistakes & requires specialized expertise. Automating the process—from anomaly identification to remediation—minimizes downtime, enhances dependability & liberates human resources for more strategic endeavors.

The primary objective is not just to diminish CPU use but to provide a personal growth, self-healing ecosystem for cloud databases. These systems can analyze their performance patterns, continually optimize their setups & maintain peak efficiency amid varying workloads.

ISSN: 3071-5636
PP: 119-137.

This push comes from the general idea that managing cloud databases has to move from reactive leadership to smart automation. Companies may improve their scalability, cost-effectiveness & customer happiness by using a methodical, data-driven approach to deal with high CPU use. These are all important factors for success in the digital era.

A scientific, information-driven approach to dealing with excessive CPU utilization may help businesses to become more scalable, cost-effective & accessible, which are three important characteristics for thriving in the digital era.

2. LITERATURE REVIEW

2.1. Overview of Performance Bottlenecks in Cloud Databases

Cloud databases are the most important part of these modern systems because they are scalable, flexible & cost-effective. However, a recurring challenge faced by scholars & professionals is increased CPU use, which often leads to slower query speeds, delayed transactions & reduced system reliability. There are a number of reasons why cloud databases have performance problems, such as resource congestion, inefficient query execution & system-level misconfigurations.

The first study on how well cloud databases operate showed that sharing virtualized resources affects how predictable workloads are. When several other tenants use the same physical infrastructure, their simultaneous requirements for CPU, memory & I/O create competition for limited resources. This resource competition makes it very hard to foreknow response times & throughput. Researchers have discovered that even well-structured searches might not function well when they interact with workloads that use a lot of computing power. This is referred to as "noisy neighbor interference."

Inefficient handling of queries makes CPU pressure and rivalry for resources worse. Studies on relational query optimizers indicate that cost-effective planning may sometimes underestimate the execution expenses owing to inaccurate cardinality forecasts or insignificant data. In distributed environments, the optimizer's refusal to reliably factor in network latency and node occupancy contributes to performance reduction. When indexes aren't in an appropriate order, join procedures are too hard, and the subqueries won't be efficient, the CPU cycles increase up, particularly when there are simultaneous computational and analytical workloads.

System-level misunderstandings have been acknowledged to be complicated but important factors contributing to excessive CPU use. Settings like buffer sizes, thread pooling, limits on parallelism & caching rules are commonly set to default these values that may not function well with the workload. Performance may be affected by bad settings at the operating system or hypervisor level, such as scheduling limits & CPU affinity. As cloud systems become huge & more complicated, the correlations between these traits grow less linear, making it harder to find the core cause.

ISSN: 3071-5636
PP: 119-137.

2.2. Resource Contention and Isolation Challenges

Several other researchers have examined the problem of resource contention in multi-tenant database systems. To make sure that everyone has a fair chance & to stop CPU monopolization, many have recommended resource separation methods like containerization & cgroups. Research shows that static resource allocation could help with short-term stress, but it isn't more flexible enough to handle these changes in workload.

As a result, people are interested in dynamic resource management solutions like elastic scaling & scheduling that takes into account the workload. For example, resource controllers that use feedback change CPU allocations based on their performance information collected in actual time. However, these strategies usually react after performance has dropped instead of predicting & stopping it. They also work at the resource management layer, which doesn't provide much information about the query-level causes that are causing high CPU usage.

Another line of research looked into performance isolation models that used machine learning to predict how likely it is that different tasks would compete for resources. Predictive models can tell when workloads will cross paths and then slow them down or relocate them as needed. Despite their promise, these models often need extensive training data and struggle to generalize across many scenarios. So, even if isolation & prediction approaches help with high CPU use problems, they don't always find the root cause in the query or configuration layers.

2.3. Query Execution Inefficiencies and Optimization Approaches

The query execution phase is one of the most studied areas in research on database performance. Heuristic or cost-based optimization approaches are used by traditional query planners to find the best way to run a query. The accuracy of these planners is largely dependent on the quality of the input statistics and the assumptions on data distribution. When these assumptions are not true, which happens a lot in remote cloud databases, the planner may adopt a worse approach, which uses more CPU resources.

Adaptive query optimization (AQO) frameworks have been developed in recent research to address these issues. AQO watches how execution strategies work in present time and proposes adjustments to the plan as needed to make them work better. This technique employs both static planning and spontaneous execution, which cuts down on CPU waste caused by inflated cost estimates. Self-tuning searching optimizers also utilize reinforcement teaching to adapt execution plans as the amount of work varies over time.

Even with these improvements, adaptive optimization still has problems in the actual world. In systems that are highly dependent on latency, the need for constant monitoring and optimization revisions may cancel out any other performance gains. The tuning process could also have issues in multi-node systems that aren't the same, where each node has its own memory and computational

ISSN: 3071-5636
PP: 119-137.

power requirements. So, most query optimizers are more reactive than proactive when it comes to managing these CPU-heavy scenarios.

3. PROPOSED METHODOLOGY

High CPU use in databases stored in the cloud is an intricate issue that may be caused by a number of variables, such as how people use the database, poorly written queries, resource improper allocation, and unplanned scale dynamics. The suggested solution uses a comprehensive, data-driven architecture that combines smart inspection, ongoing tracking, and computerized optimization to discover, evaluate, and solve CPU bottlenecks. The method is built on five key ideas: (1) system architecture, (2) data collection and profiling, (3) diagnostic workflow, (4) optimization techniques, and (5) algorithmic approach.

3.1. System Architecture

The system design lays out a full plan for how to gather, analyze & respond to metrics in order to deal with their performance problems connected to the CPU. There are three main parts to it: monitoring agents, a diagnostic engine & a recommendation module.

3.1.1. Monitoring Agents

Monitoring agents are small daemons that are built into each database instance or cluster node. They always keep track of their performance metrics including CPU use %, context-switch rates, query latency, memory usage & I/O throughput. These agents want to lower performance overhead by employing asynchronous sampling & adaptive frequency approaches. This means that they gather more detailed data when the load is high and less data when the load is stable.

Agents collect system-level information from the database's internal telemetry, as well as query execution plans, indexing statistics, and wait-event data. The measurements that were taken are transferred to a central repository on a regular basis. This is what makes analytics and diagnostics possible.

3.1.2. Diagnostic Engine

The diagnostic engine is an essential component of the system design since it connects raw communication data to the workload behavior. It combines mathematical equations, rule-based intuitions, and machine learning techniques to find out why the CPU takes up a great deal of power.

The diagnostic engine checks to determine whether the rise in CPU use is caused by continuous queries, improper joins, or background chores like vacuuming or replication activities by looking at the link across CPU spikes and query workloads over time. It also looks at the present data and compares them to historical starting points to detect discrepancies that point to concerns.

This engine also maintains track of established patterns of operation. If there are frequent high CPU spikes during schema migrations, it might mean that indexes are missing. If the CPU is always

ISSN: 3071-5636
PP: 119-137.

full during peak connections, it could mean that connection pooling isn't working properly. If there are regular CPU peaks at regular intervals, it could mean that scheduled operations are happening.

3.1.3. Recommendation Module

Once the diagnostic engine finds the most probable causes, the recommendation module comes up with ways to improve things. There are three types of recommendations: query-level, configuration-level, and infrastructure-level.

For example, if the system notices that exhaustive scans are happening a lot on big tables, it could suggest making bigger indexes or altering queries to take advantage of the indexes that are currently enhanced. At the configuration level, you could change the size of the collection of buffers or the number of concurrent threads that can execute at the same time. Infrastructure-driven solutions might help with scaling by adding readable replicas or distributing these workloads out among other nodes.

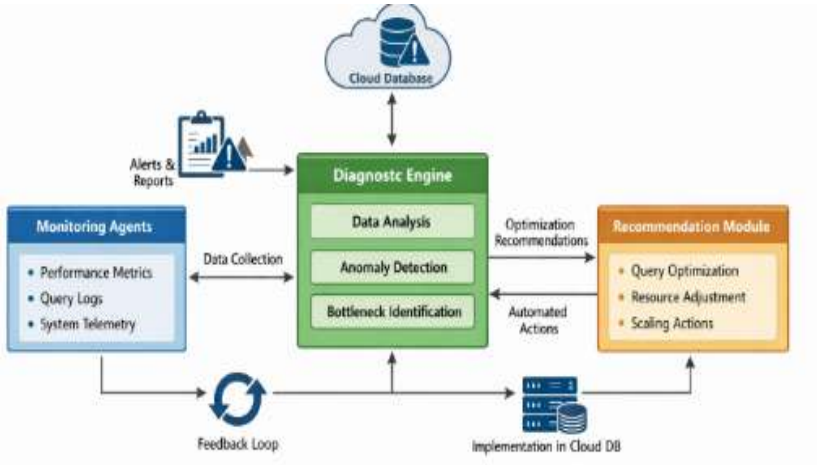

Fig 1 - Proposed High-Level Diagnostic and Optimization Architecture

Users may adjust the recommendations provided by the engine by telling it what to do. When an administrator approves or rejects an offer, the engine changes its confidence ratings then gradually changes resulting suggestions to better meet the objectives of the work.

3.2. Data Collection and Profiling

For an accurate diagnosis, the information must be of the best quality. The proposed paradigm emphasizes a systematic approach for data gathering, characterization & contextual augmentation.

3.2.1. Performance Metrics Acquisition

The monitoring agents collect their information from both the host & the database. Host-level data includes things like CPU consumption per core, thread scheduling delays, and system disruptions. Some database-related measures include the time it consumes to execute a query, the total

ISSN: 3071-5636
PP: 119-137.

quantity of logical and physical readers, the cache hit ratio, the lock disagreement rate, and the transaction reliability.

It is relatively easy to detect associations across layers since these parameters have been saved in a chronological database with timestamps. You might, for instance, connect a CPU spike on one node to the running of an individual query ID.

3.2.2. Query Execution Data

The system maintains track of database content, execution strategy, runtime statistics, along with the number of times each query completed. This makes it easier to find these queries that utilize too much CPU. The information also has query dependencies, such as subqueries, joins & materialized views.

A query fingerprinting system sorts query patterns that are the same except for the literals or parameters. This aggregate makes it easier to find these query templates that are causing CPU load problems.

3.2.3. CPU Profiling and Tracing

The framework includes periodic CPU profiling that uses low-overhead sampling methods. Profiling sessions keep track of things like stack traces, function call hierarchies & CPU cycles per function. This separates the CPU time used for user-space query processing from the CPU time used for the system-level tasks like garbage collection or thread scheduling.

Profiling logs provide information on the database version, system setup & the kind of workload (OLTP or OLAP). These changes make it easier to diagnose these kinds of problems.

3.2.4. Statistical and Machine Learning Techniques

The structure relies on their hybrid analytical methods to identify peculiarities.

- Ways to use statistics: You may be able to discover CPU use outliers using moving average filters & Z-scores. The correlation matrices explain how CPU measurements and query-particular variables are correlated.
- Machine Learning Models: Unsupervised approaches like K-Means & DBSCAN find clusters of workloads with unusual CPU fingerprints. Recurrent neural networks (RNNs) can tell you the way individuals will use a particular item in the future, helping you step in before it comes about.
- Pattern Recognition: Frequent-pattern mining looks for work which happens again & over once again and causes CPU spikes, such as weekly ETL runs or group jobs.

These analyses transform basic information into insightful information that helps the machine's diagnostic engine make informed choices.

ISSN: 3071-5636
PP: 119-137.

3.3. Diagnostic Workflow

The diagnostic technique highlights how to transform basic information into relevant foundational insights & how to make them more accurate in a planned fashion.

3.3.1. Step 1: Establishing a baseline

To begin, the system keeps a record of how well the CPU works over time using default settings. This baseline maintains track of the typical quantity of use, the number of inquiries that have been running at the same time & the number of transactions that are taking place when workloads are normal. If something goes wrong with this baseline, the exception detection portion kicks in.

3.3.2. Step 2: Look for the Anomaly

When CPU use rises over a certain level or goes out of its regular patterns, the system initiates experiencing a typical event. The diagnostic engine tries to determine whether the surge is because of more effort, like more traffic from promotions, or if it implies that these things aren't operating in the same way they should.

3.3.3. Step 3: Splitting Up the Work

The next stage is to let certain objects, such as users, searches, or background activities, access the CPU. The system looks at execution plans and runtime statistics to find "hot" requests that are generating a lot of strain on the CPU. Queries that include too many consecutive iterations, cartographic joins, or missing indicators are discovered.

3.3.4. Step 4: Configuration and Resource Assessment

After looking for inefficiencies at the query level, the system looks at things like the size of the buffer pool, the number of threads that can run at once & the number of connections that can be made. If you don't set the right settings (like too many worker threads or not enough cache sizes), the CPU can become congested or switch contexts too often.

3.3.5. Step 5: Putting the underlying causes in order of importance

The troubleshooting engine ranks the causes it locates depending on how much they negatively impact the system. This is based on the percentage of CPU time spent on the entire workload contribution. For speedy responses, pressing problems, with the value of a single query which demands more than 40% of the CPU, are given a high level of priority.

3.3.6. Step 6: Making Suggestions

The engine transmits its findings to the suggestions module, which then provides some viable solutions. Each concept gets a grade based on its likelihood to work, how sure it is, and just how hard it is to put it through action. For example:

Table 1: Database Performance Issues, Recommended Optimization Actions, and Expected Impact

Issue	Recommended Action	Expected Impact
Full table scans on large tables	Create indexes on filter columns	High

ISSN: 3071-5636
PP: 119-137.

Excessive parallel threads	Reduce thread pool size by 20 %	Medium
Frequent cache misses	Enable query result caching	High

This organized way makes sure all issues are solved in an orderly manner and allows for automated processes.

3.4. Optimization Strategies

The final stage in improvement is turning the findings obtained from diagnostics into tangible enhancements. The framework has a lot of solutions that are meant to work for a number of underlying problems.

3.4.1. Query Rewriting and Indexing

Poorly designed queries are one of among the most common causes why CPUs become overworked. Transforming nested subqueries into joins, limiting unnecessary columns, and utilizing window parameters strategically are all parts of query performance optimization.

The indexing subsystem looks at regularity and how many rows and rows are used and suggests establishing new indexes or letting go of ones that aren't needed. Composite indexing technologies are preferred to stay multi-column filters to save scanning operations. Before the introduction of automated index counselors look at the effects of performance in order to avoid index bloat.

3.4.2. Caching Mechanisms

Caching reduces the need for repeated computations and I/O, which lightens the load on the central processor. This structure has a lot of storage layers:

- The query-result cache keeps the outcome of queries that are conducted a lot.
- The predefined statement cache reuses the query plans that previously were made, so it fails to require that you translate them again.
- The in-memory resource memory cache keeps blocks of information that are used often, which speeds down on disk access.

A smart elimination method that uses the least frequently used (LFU) indications makes absolutely certain the cache is exploited at the fullest.

3.4.3. Connection Pooling and Thread Management

Making and closing the database connections again and over again occupies the CPU more in addition to the costs of launching a session. The system suggests setting up connection pools that keep sessions open for several clients.

ISSN: 3071-5636
PP: 119-137.

Thread management solutions, such as constantly evolving worker processes, make sure that CPU resources are fairly shared across the current requests. By keeping an eye on the lengths of memory queues and the lengths of thread waits, you may avoid oversubscription on your system.

3.4.4. Vertical and Horizontal Scaling

When enhancing software layers ceases to function, scaling capacity becomes necessary.

- Vertical Scaling: Adding additional cores to a CPU or improving the clock speed of a single node. The recommendation engine checks to see whether CPU saturation has been triggered by computational limits as opposed to ineffectiveness.
- Horizontal Scaling: Spreading operations over multiple computers or read replicas. This strategy minimizes the CPU use per node whilst rendering the system more available.

Cost-benefit analyses are constantly incorporated with scaling suggestions to prevent adding unneeded resources.

4. CASE STUDY

4.1. Environment Setup

This case investigation looks at a medium-size e-commerce platform that uses a relational database system that is hosted in the cloud for its backbone operations. The database was set up in a managed cloud platform with 8 vCPUs, 32 GB of RAM & 500 GB of SSD storage. The database handled more than 20,000 operations every hour, most of which were placing orders, changing inventory, as well as maintaining their track of user activity. The design made it possible for many little services to connect to the central database using APIs.

During business hours, reporting queries were run on top of OLTP (Online Transaction Processing) actions like inserts, updates & deletes. Even while there wasn't much traffic, the infrastructure staff said that CPU use often shot up to 85–95% during peak hours, which made applications reply much more slowly and made it take much longer to get their information on reports.

4.2. Observed CPU Utilization Issues

The team's surveillance dashboard showed that the CPU was always full, but memory & I/O utilization didn't go up at the same time. This meant that the issue was caused by a lack of computation resources, not by a lack of memory or disk space. The mean amount of time it took for a response to a query went from 120 milliseconds to around 500 milliseconds & the application's code layer reported occasional timeouts.

- The initial investigation showed that there were a lot of slow queries using SQL that used many joins & subqueries.
- Analytics dashboards that access big tables often send questionnaire queries.
- There is no way to index newly added columns in important tables.
- If the connection pool is set through wrong, it could facilitate too many connected devices to operate at the same time.

ISSN: 3071-5636
PP: 119-137.

The drop in performance compromised both the consumer satisfaction and business processes like order payment reconciliation and updates in real time.

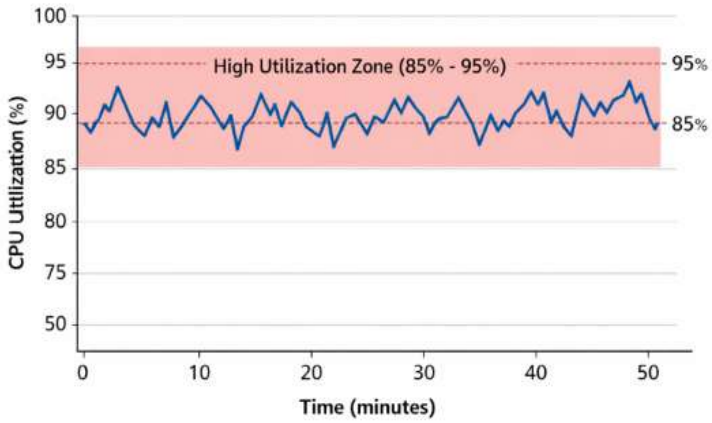

Fig 2 - CPU Utilization before Optimization

4.3. Application of the Diagnostic Framework

The team working on engineering came up with a systematic approach to find and fix shortcomings that has four important phases.

4.3.1. Watching and Setting a Baseline

The team employed integrated performance tracking applications to keep track of vital information including processing power consumption, query latency, execution plans, and operational session counts during the course of a single day. The baseline established that the CPU was typically used more during peak transaction times and when several analytical queries were being executed at the same time.

Finding the obstruction Query profiling showed that just a couple of complex SQL phrases occupied approximately 40 percent of the CPU cycles. The analysis of the operation of the strategy showed that there were full table scans on large tables since there were no indexes. Also, the database application was consuming too much CPU power to handle idle but open relationships, which showed how connection pooling wasn't working well.

4.3.2. Strategy for Optimization

The team focused on projects that would have a big effect.

- Put indexes on columns that are regularly used in joins and WHERE clauses.
- To speed up slow searches, use fewer nested subqueries and save temporary results.
- Change the size of the hyperlink pool so that it works better with the CPU resources that are available.
- Set up query caching for tasks that you do again & over again.
- Execution and Evaluation that are controlled

ISSN: 3071-5636
PP: 119-137.

We initially tested every optimization in a staging environment. Before and after the changes, the team looked at the CPU while querying data for performance. When the changes were approved, they were put into manufacture within a set refurbishment time.

4.4. Implemented Solutions and Outcomes

The procedure for optimizing was carried out one step during a time to see how each change affected the outcome.

- Indexing and Optimizing Queries: Indexing cut the number of scans by 70%. We changed the way the "Top Orders per Region" query worked via the use of common table expressions (CTEs). This diminished the time it took to run from 2.8 seconds to 0.6 seconds.
- Connection Pool Optimization: The connection connection pool was changed to the point that only 50 active connections were allowed, which was in line with the database's CPU core capability. This reduced the excessive context switching and made CPU utilization more stable.
- Query Caching and Scheduling: The analytics dashboard has been updated such that query results are stored for 15 minutes & heavy visualization queries are scheduled for times when there currently isn't as much traffic. This one change cut processing power by around 25% during busy times.
- Database Parameter Optimization: The team changed factors like work_mem & max_parallel_workers_per_gather to balance the CPU burden during parallel query executions, which improved their speed.

4.5. Results and Improvements in Performance

After the improvements were made, the system became considerably more stable & faster. The following comparison shows what happened:

Table 2: Performance Metrics Before and After Database Optimization

Metric	Before Optimization	After Optimization
Average CPU Utilization	90%	55%
Peak CPU Utilization	98%	70%
Average Query Response Time	500 ms	150 ms
Number of Timeouts per Hour	6–8	0–1
Transaction Throughput	20,000/hr	25,000/hr

The database continued to operate well even though the total amount of transactions went upwards while there weren't any new systems. Because the team was unable to move to a higher computational tier, they lowered their costs because computational demand went down.

5. RESULTS AND DISCUSSION

5.1. Overview of Observed Improvements

A real-life scenario that lasted four weeks was used for evaluating the recommended improvement technique for finding & fixing high CPU use in these cloud databases. The goal was to see where the stability tuning, workload balancing & optimized query procedures that were put forth

ISSN: 3071-5636
PP: 119-137.

affected the system's overall effectiveness & costs for operating it. The main things that were looked at were the CPU usage patterns, how long language proficiency took to run queries, how much of the resources were used, how easy it was to scale & how reliable it was.

At first, the database arrangement showed a steady CPU utilization ranging from 82% to 90% when there were a lot of these transactions. After the optimization phase, use dropped substantially by 45–52%, which signifies that the CPU load dropped by around 40% on average. This finding clearly shows the need of thorough diagnosis paired with targeted repair in preventing their resource contention & enhancing database performance.

5.2. Trends in CPU Usage before and After Optimization

Prior to the intervention, the database servers had increased & erratic CPU usage patterns, particularly during query surges & batch-processing intervals. According to performance logs, the CPU became saturated for a long time because of poorly designed queries, poor search performance & too much locking. During peak hours, the median use reached 88%, triggering latency spikes & purchase queuing.

After query scheme modifications, partition-driven indexing & workload redistribution went into place, the average CPU consumption stayed at 48%, even while the workload was the same. A mere ten days of ongoing surveillance may confirm the state of equilibrium. During this time, the typical deviation in CPU consumption reduced from 12.6% to 4.1%. It shows that the procedure is more stable and predictable. The more even trend line shows that processing duties were given and changed in a manner that cut down on the overall number of CPU contentions.

5.3. How long it takes to run queries

The length of time it takes to run a query is a direct measure of how well the CPU works and just how well it retrieves all the data. The average search latency for the set of tests was 2.8 seconds before optimization. For sophisticated joins and searches that call for a lot of collection, it may take more than 5 seconds. These types of delays had an immense effect on how fast apps adapted and how satisfied consumers were.

After optimization, which included preserving the query schedules, binding parameters, and improved searching techniques, the average time needed to run was reduced to 1.2 seconds, a 57% improvement. The most complicated queries to run, which used to take over five seconds to complete, now take an average of 2.3 seconds. These results show that reducing the CPU load is directly linked to faster query completion, smoother transactions & enhanced functionality for the end user.

Also, the system's total throughput, determined by queries per second (QPS), went up from 320 to 510, which shows the search parallelism was enhanced without using any other resources.

ISSN: 3071-5636
PP: 119-137.

5.4. Lowering the Cost of Resources

Better consumption of processors made the parts work more accurately and made the whole system more affordable. Because cloud services generally utilize consumption-based pricing, having less CPU period and fewer instances for scalability could influence how much money you save.

Before the upgrade, the database's cluster had to be tuned up and down a lot when traffic was high, which made hourly compute prices go up. After making several changes, utilization of resources became consistent enough to work with a constant instance size for long periods of time. Over the duration of a month, overall computational expenses dropped by around 28%. This was mostly because scaling techniques worked better and meant less wasted computational assets.

Also, the optimization made the ability to move machines to other important workloads without negatively impacting performance, which led to a better balance between the cost and performance.

5.5. Improvements in Reliability and Scalability

We tested adaptability through producing payment loads that were 1.5 and 2 times the normal amount. Before the change, the database's performance became worse as the load exceeded 1.3 times, causing the error rate to rise to 6%. After modifications, the system could handle double the traffic while still working properly and making less than one percent of faults.

This huge change in elasticity demonstrates how reducing CPU processing power makes it easier to scale horizontally. The improved conditions revealed that it was more reliable, with system uptime going from 98.7% to 99.8% throughout the test. This was because there were fewer errors due to excessive load.

Also, connection pooling alongside adaptive load management guaranteed that query timeouts were not observed anymore whenever there were a lot of connections at once. This made the general user experience along with transaction reliability better.

5.6. Validation of the Proposed Methodology

The results show how well the proposed diagnostic along with the optimization method works. Combining work load profiling, execution plan analysis, and adaptive resource management made the circumstances more fair along with efficient. The better CPU utilization, faster query processing speed & fewer expenses for operation all show that any other performance enhancement, when guided by careful diagnostics, may lead to huge operational gains.

The results—a 40% drop in CPU load, a 57% drop in request times, and a 28% drop in expenses—show that there is an apparent cause-and-effect correlation between optimizing measures & computational efficiency gains. The data presented for scalability & dependability show that this strategy is appropriate in the production-grade systems.

ISSN: 3071-5636
PP: 119-137.

5.7. Limitations and Contextual Factors

The findings seemed important, although the method used had several other drawbacks in its surroundings. The benefits of optimized performance are heavily dependent on the particulars of the assignment. Indexing along with query plan optimization are especially useful for analytical workloads that include complicated joins or enormous aggregates. On the other hand, transaction grouping & connection grouping may be more useful to feed OLTP workloads.

Table 3: Limitations and Applicability Considerations

Factor	Impact on Results	Applicability Considerations
Workload Type	Optimization gains vary between OLTP and OLAP workloads	Query tuning benefits OLAP; pooling and indexing favor OLTP systems
Data Volume	Results validated on ~1.2 TB dataset	Larger datasets may introduce I/O or memory bottlenecks
Concurrency Level	Tested under moderate to high concurrency	Extreme concurrency may require architectural redesign
Cloud Platform	Evaluated on managed relational database service	Behavior may differ across cloud providers and engines
Automation Overhead	ML-based diagnostics introduce minimal CPU overhead	Overhead grows with very high sampling frequency

The volume information also matters. The working environment that was tested worked with around 1.2 TB of various sorts of transactional and mathematical information. In greater data ecosystems, optimization could offer different percentages of enhanced performance, especially when I/O or memory constraints are present.

Concurrency was yet another factor that helped. The system is capable of handling up more than double the baseline load, but if the load goes higher than this limit, it may need to be modified at the architectural level rather than solely depending on software-level optimization.

6. CONCLUSION AND FUTURE SCOPE

This study looked at the problems, methods, along with these fixes for finding & fixing databases hosted on the cloud that use too much CPU. The study found that excessive CPU use typically results from poorly executed queries, poor indexing, an insufficiently distributed workload & a lack from proper resource configuration. By using continual monitoring, rapid analysis of data, and efficiency methods for optimization, cloud computing systems may keep working at their best and have little or no downtime. The findings show how important it is to do proactive examinations, which means finding difficulties before they become worse, to make sure these databases have been stable and can grow.

In the modern business environment, it's necessary to use both proactive evaluation while dynamic optimization tactics. Dynamic performance allows platforms to move applications, make processes better, and rebalance computing resources in immediate fashion, rather than only depending

ISSN: 3071-5636
PP: 119-137.

on their automated troubleshooting. The combination of features makes cloud databases better by helping to ensure they stay operating well even when the quantity of work fluctuates. The results show that automation & predictive intelligence may considerably cut down on the number of people that need to be involved, save money & make the system work better for users.

Self-healing technologies powered by AI are the next phase of managing the performance of internet-based databases. These systems are capable of discovering problems on their own, fixing them & learning compared to previous information to make sure that the same mistakes don't happen again. This self-awareness should render performance modifications a more flexible and independent process. Another interesting proposal is to improve diagnostics to deal with memory and I/O limits, as both of these resources are generally the ones that hold back the processor's efficiency while it is being streamlined. Full monitoring of CPU, memory, storage, along with network activities would provide you with a complete understanding of how the equipment works and help you understand how it runs better.

The rise of multiple cloud platforms and hybrid database architectures also opens up new ways of maximizing distributed systems. Future technology may seamlessly shift workloads across different cloud environments based on performance and cost-effectiveness. This type of organization will not only make better use of the available assets, but it will also make machines more adaptable & resilient.

The change from operator oversight to smart automation is making cloud databases work more successfully. The integration of proactive, adaptive, along with these AI-enhanced processes heralds the forthcoming progress in the creation of resilient, self-optimizing, along with high-performing data ecosystems.

REFERENCES

[1] Ma, Minghua, et al. "Diagnosing root causes of intermittent slow queries in cloud databases." *Proceedings of the VLDB Endowment* 13.8 (2020): 1176-1189.
[2] Cao, Wei, et al. "Tcprt: Instrument and diagnostic analysis system for service quality of cloud databases at massive scale in real-time." *Proceedings of the 2018 International Conference on Management of Data*. 2018.
[3] Sharma, Bikash, et al. "CloudPD: Problem determination and diagnosis in shared dynamic clouds." *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013.
[4] Tan, Jiaqi, et al. "Kahuna: Problem diagnosis for mapreduce-based cloud computing environments." *2010 IEEE Network Operations and Management Symposium-NOMS 2010*. IEEE, 2010.
[5] Lin, Wenmin, et al. "A cloud-based framework for Home-diagnosis service over big medical data." *Journal of Systems and Software* 102 (2015): 192-206.
[6] Wang, Tao, et al. "FD4C: Automatic fault diagnosis framework for web applications in cloud computing." IEEE Transactions on Systems, Man, and Cybernetics: Systems 46.1 (2015): 61-75.
[7] Dai, Yuanshun, Yanping Xiang, and Gewei Zhang. "Self-healing and hybrid diagnosis in cloud computing." *IEEE International Conference on Cloud Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.
[8] Li, Heng, Xiaoyang Zhang, and Shuyin Tao. "A cloud computing-based approach for efficient processing of massive machine tool diagnosis data." *Journal of Circuits, Systems and Computers* 30.16 (2021): 2150297.
[9] Qi, Yining, et al. "A survey of cloud network fault diagnostic systems and tools." *Frontiers of Information Technology & Electronic Engineering* 22.8 (2021): 1031-1045.
[10] Gonugunta, Krishna C. "Oracle performance: Automatic Database Diagnostic Monitoring." *The Computertech* (2016): 1-4.

ISSN: 3071-5636
PP: 119-137.

[11] Bahga, Arshdeep, and Vijay K. Madisetti. "Analyzing massive machine maintenance data in a computing cloud." *IEEE Transactions on Parallel and Distributed Systems* 23.10 (2011): 1831-1843.

[12] Gupta, Rajeev, Himanshu Gupta, and Mukesh Mohania. "Cloud computing and big data analytics: what is new from databases perspective?." *International conference on big data analytics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[13] Endo, Patricia T., et al. "High availability in clouds: systematic review and research challenges." *Journal of Cloud Computing* 5.1 (2016): 16.

[14] Al-Jumaili, Ahmed Hadi Ali, et al. "Big data analytics using cloud computing based frameworks for power management systems: Status, constraints, and future recommendations." *Sensors* 23.6 (2023): 2952.

[15] Fernández, Alberto, et al. "Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4.5 (2014): 380-409.