Wait! What’s going on inside my database?

PostgreSQL and Optimizing Database Performance

Jeremy Schneider

SCaLE 21x
3-14-2024
About PostgreSQL

1970: Mathematician Edgar F. Codd, working as researcher for IBM, publishes “A Relational Model of Data for Large Shared Data Banks”

1973: Michael Stonebraker and Eugene Wong at University of California Berkeley seek funding and begin development of a relational database called INGRES

1986: Michael Stonebraker and Lawrence A. Rowe at University of California Berkeley publish “The Design of POSTGRES” – a new database that is the successor to INGRES

1994: Andrew Yu and Jolly Chen at University of California Berkeley add support for the SQL language

1996: Transition to non-university core team of volunteers, official release under new name POSTGRESQL
PostgreSQL supports most of the major features of SQL:2011. Out of 179 mandatory features required for full Core conformance, PostgreSQL conforms to at least 160. In addition, there is a long list of supported optional features. It might be worth noting that at the time of writing, no current version of any database management system claims full conformance to Core SQL:2011.

In the following two sections, we provide a list of those features that PostgreSQL supports, followed by a list of the features defined in SQL:2011 which are not yet supported in PostgreSQL. Both of these lists are approximate: There might be minor details that are nonconforming for a feature that is listed as supported, and large parts of an unsupported feature might in fact be implemented. The main body of the documentation always contains
About Database Performance
Response time in man-computer conversational transactions

by ROBERT B. MILLER

International Business Machines Corporation
Poughkeepsie, New York

INTRODUCTION AND MAJOR CONCEPTS

The literature concerning man-computer transactions abounds in controversy about the limits of "system response time" to a user's command or inquiry at a terminal. Two major semantic issues prohibit resolving this controversy. One issue centers around the question of "Response time to what?" The implication is that different human purposes and actions will have different acceptable or useful response times.

This paper attempts a rather exhaustive listing and definition of different classes of human action and purpose at terminals of various kinds. It will be shown that "two-second response" is not a universal requirement.

The second semantic question is "What is a need or requirement?" In the present discussion, the reader is asked to accept the following definition: a need is some demonstrably beneficial known alter- 

Operating needs and psychological needs

An example of an operating need is that unless a given airplane's velocity exceeds its stall speed, the airplane will fall to earth. Velocity above stall speed is an undebatably operating need. In a superficially different context, it is a "fact" (let's assume we know the numbers) that when airline customers make reservations over a telephone, any delays in completing transactions above five minutes will reduce their making future reservations with this airline by 20%. A related form of need in this context is that the longer it takes to process one reservation, the larger the number of reservation clerks and reservation terminals that will be required. These are just two examples of the context of operating needs. This report will not look into the problems of operating needs except to mention when they may be more significant than a psychological need. The following topics address psychological needs.
About Database Performance

Other Methodologies

While the USE Method may find 80% of server issues, latency-based methodologies (eg, Method R) can approach finding 100% of all issues. However, these can take much more time if you are unfamiliar with software internals. They may be more suited for database administrators or application developers, who already have this familiarity. The USE Method is more
About Database Performance

- Evaluating Software and Hardware
  - Vendors
  - Customers
- Price - Performance
  - Throughput = Response Time \times Concurrency
  - Scalability? (can add HW?)
  - Efficiency? (HW cost?)

- Responding to Problems
  - User Experiences
  - Response Time
  - Coordination across departments, across vendors, across experts
1990’s Manager:

“Dear DBA: Expert consultants have taught us that if the Buffer Cache Hit Ratio (BCHR) is below 90% then the system immediately needs an expensive tuning engagement.

Please report any databases that have BCHR < 90%.”
About Database Performance

1990's Manager: "Dear DBA: Expert consultants have taught us that if the Buffer Cache Hit Ratio (BCHR) is below 90% then the system immediately needs an expensive tuning engagement. Please report any databases that have BCHR < 90%.

Choose any hit ratio

In these days where people are still mistakenly waiting to bump up your hit ratio to any value desired. This procedure can be used.

```sql
create or replace procedure choose_a_hit_ratio(p_ratio number, v_phy number, v_db number, v_con number, v_count number);
```

SQL> exec choose_a_hit_ratio(85);
Current ratio is: 82.308333
Another 29385 consistent gets needed...
Current ratio is: 86.24548
PL/SQL procedure successfully completed.

SQL> exec choose_a_hit_ratio(90,true);
Current ratio is: 86.24731
Another 79053 consistent gets needed...
PL/SQL procedure successfully completed.

SQL> exec choose_a_hit_ratio(90);
Current ratio is: 86.24731
Another 79053 consistent gets needed...
Current ratio is: 90.5702
Correct Instrumentation Is Key

In the mid 1980s, I realized that no matter how many counters and monitors you looked at, it was still pure guesswork (hence luck or lack thereof) whether a particular metric was the bottleneck of a given application or business unit. So they instrumented the whole mainframe environment, including DB2.

Another story. Around 1991 or 1992, Juan Loaiza and others from Oracle development were forced to instrument the Oracle kernel in the same way. Here’s the story, as told to me by Juan (he’s now vice president in Oracle kernel development). It is also my tribute to one of the truly great minds inside Oracle Development.

I think what you are referring to are the wait statistics that were implemented in 7.0. This stuff was developed because we were running a benchmark that we couldn’t figure out why.

Statistics and ratios and kept coming up with theories, the of them were right. So we wasted weeks tuning and fix-
R = S + W

“How long the SQL takes to run”


See also:
About Database Performance

Active Session Sampling

Images & Quotes Used With Permission
Optimizing Oracle Performance

Published 2003

Oracle Wait Interface: A Practical Guide to Performance Diagnostics & Tuning

Published 2004

For many people, Oracle performance is a very difficult problem. Since 1984, I’ve worked with thousands of professionals engaged in performance improvement projects for their Oracle systems. Oracle performance improvement projects appear to progress through standard stages over time. Although the factors of these stages are well described in many consultation-based books, I’ve observed recently, the emphasis on Oracle performance tuning is often based on the lack of appreciation for the overall Oracle database administration.

The Old Fashion of Oracle Performance Optimization

Some claim that the mindset of Oracle database administrators (DBAs) has changed over the years. Some DBAs consider themselves experts in specific areas of Oracle, such as SQL tuning and the database cache. These areas present challenges to Oracle DBAs, who can face difficult situations in their efforts to optimize performance. However, the mindset of Oracle DBAs has not changed significantly over the years.

Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBAs still focus on SQL tuning. In the past, Oracle performance tuning was limited to SQL tuning. However, the mindset of Oracle DBAs has not changed significantly over the years. Many DBAs still focus on SQL tuning, and while the database cache is important, many DBA...
What about PostgreSQL?
Wait Events

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA
**SMF (System Management Facilities)** is a feature of the IBM System/360 Operating System OS/VSE that provides the means for gathering and recording information that can be used for billing customers or evaluating system usage. Information is gathered and recorded by SMF data-collection routines and by user-written exit routines. Because the data-collection and exit routines are independent of one another, they may be used in combination or separately.

Note: SMF cannot be used for monitoring system tasks.

SMF data collection routines gather several types of information:

- Accounting information, such as CPU time and device and storage usage.
- Data-set activity information, such as EXCP count and the user of the data set.
- Volume information, such as the space available on direct access volumes and error statistics for tape volumes.
- System use information, such as system wait time and I/O configuration.

The type of data to be collected can be modified by the operator at each initial program loading (IPL).

Through user-written analysis routines and report routines, this information can be used in a variety of ways. For example, this information can be used to prepare customer's bills. The information might also be used to measure system usage against departmental standards of efficiency and performance.
Wait Events

R = S + W

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose


“How long the SQL takes to run”

See also:
Wait Events

Active Session Sampling

(DB TIME = area under the curve)

Height = # of Sessions
Width = seconds
Area under curve = DB Time

\[
DB Time = \sum_{i}^{n} \text{active sessions}(ii) \times \Delta t
\]

DB Time = sum of active time in database

Date: Tue, 5 Mar 2019 13:12:51 -0800
From: John Beresnewicz
To: Jeremy Schneider

since you asked...

The story of how the group that designed the Performance Page came to be is somewhat interesting as it involves myself, Kyle Hailey, Gaja Vaidyanatha and James Morie being hired as a kind of “design Tiger Team” by a VP at Oracle within the EM organization who needed to expose DB Manageability (marketed as Diagnostic and Tuning Packs) through Enterprise Manager... a team of experienced outsiders with successful product experience (at Savant and Quest) was recruited to be the designers and usability experts. We were there because we had been DBAs, knew their mindset, had created successful products in the space, and had good design sensibilities. I joined the group last (or maybe James came after) in October 2002, having learned of the opportunity by serendipitously running into Gaja in a hotel gym in San Juan, Puerto Rico where he was on the last day of his stay and my wife and I had just arrived and were touring the facility.

Images & Quotes Used With Permission

(JB’s notebook, 2004)
Wait Events

• 1990s: Database kernel instrumentation:
  • Counters and tools to snapshot/compare them
  • Events (log a message under certain circumstances)

• 1992: Unable to solve a performance problem, as a last resort, engineers added event code in version 7.0.12 capable of emitting trace messages when the database waited for something

• First exposed in V$SESSION_WAIT and later in V$SESSION (equivalent of pg_stat_activity)

• PostgreSQL built on concepts that had become standard across the industry
Wait Events

“But why are these events called wait events?"

…”

In short, when a session is not using the CPU, it may be waiting for a resource, an action to complete, or simply more work. Hence, events that associated with all such waits are known as wait events.”

Wait Events

High-Level Idea:

The database is WAITING any time when it’s not running on the CPU

Caveats:

• OS scheduling/runqueue
• Measurement overhead
• Non-database CPU time
Wait Events
Re: Waits useless on MySQL?

From: "Jonah H. Harris" <jonah.harris@xxxxxxxxx>
To: gogala.mladin@xxxxxxxxx
Date: Mon, 20 Feb 2023 15:51:00 -0500

In 2007, I was working on trying to get EnterpriseDB/Postgres to the point where we could run an audited TPC-C. While there was no way in hell that was going to actually happen, I got tired of dealing with the lack of instrumentation and trying to track down where the slowdowns were without using profiling/debugging-compiled builds that didn't reflect what we were actually trying to run. Accordingly, I wanted to add Oracle-style wait instrumentation to it, which ended-up being a multi-hour-long argument with our sponsored Postgres community members, who felt it wasn’t needed and didn’t see the point. “Who needs that when you have sar, top, vmstat, etc.,” they said. (Anyway, with the support of Korry Douglas (who now leads the Babelfish architecture at AWS), I finally won the argument and decided to code it that night out of sheer rage. As I generally code better a little buzzed, I grabbed a nearby bottle of tequila and margarita mix and got to work. The next morning, all the major components were instrumented. I named the instrumentation system MARGARITA (Managed Array-based Reporting, Grading, and Aggregating Runtime Instrumentation and Tracing Architecture.) Management ended-up renaming it DRITA, as they felt my original name wasn't fit for publication. A few months later Peter Steinhoeuser wrote a simple AWR clone on top of it. I don't know if they still have it, but it was better than what exists in community Postgres today.

--

Jonah H. Harris

Jeremy Schneider
@jer_s

An epic slice of EDB and Postgres history around wait events:

"I finally won the argument and decided to code it that night out of sheer rage ... management ended-up renaming it as they felt my original name wasn't fit for publication"

freelists.org/post/oracle-1/...

9:43 PM · Aug 2, 2023 · 1,268 Views
<table>
<thead>
<tr>
<th>state_change</th>
<th>timestamp with time zone</th>
<th>Time when the state was last changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>waiting</td>
<td>boolean</td>
<td>True if this backend is currently waiting on a lock</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>state</th>
<th>text</th>
<th>Current overall state of this backend. Possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• active: The backend is executing a query.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• idle: The backend is waiting for a new client command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• idle in transaction: The backend is in a transaction, but</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• idle in transaction (aborted): This state is similar to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fastpath function call: The backend is executing a fast</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• disabled: This state is reported if track_activities is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• disabled:</td>
</tr>
</tbody>
</table>

[https://www.postgresql.org/docs/9.5/monitoring-stats.html](https://www.postgresql.org/docs/9.5/monitoring-stats.html)
Wait Events

Significant Commits: Version 9.6
- Aa65de0 – 11 Sep 2015 – Autogenerate lwlocknames.[c|h]
- 53be0b1 – 10 Mar 2016 – Heavy/Lightweight Locks, Buffer Pins

Version 10
- 6f3bd98 – 4 Oct 2016 – Latches & Sockets, Clients, Main Loops
- 249cf07 – 18 Mar 2017 – I/O
- Fc70a4b – 26 Mar 2017 – Background and Auxiliary Processes

Version 11
- 1804284 – 20 Dec 2017 – Parallel-Aware Hash Joins
Wait Events

Version 12
- Add a wait event for fsync of WAL segments (Konstantin Knizhnik)
- Ensure that TimelineHistoryRead and TimelineHistoryWrite wait states are reported in all code paths that read or write timeline history files (Masahiro Ikeda)

Version 13
- Rename various wait events to improve consistency (Fuji Masao, Tom Lane)
- Report a wait event while creating a DSM segment with posix_fallocate() (Thomas Munro)
- Add wait event VacuumDelay to report on cost-based vacuum delay (Justin Pryzby)
- Add wait events for WAL archive and recovery pause (Fujii Masao)
- The new events are BackupWaitWalArchive and RecoveryPause.
- Add wait events RecoveryConflictSnapshot and RecoveryConflictTablespace to monitor recovery conflicts (Masahiko Sawada)
- Improve performance of wait events on BSD-based systems (Thomas Munro)

Version 14
- Add wait event WalReceiverExit to report WAL receiver exit wait time (Fujii Masao)
- Wake up for latch events when the checkpoint is waiting between writes. This improves responsiveness to backends sending sync requests. The change also creates a proper wait event class for these waits. (Thomas Munro)

Version 15
- Add wait events for local shell commands. The new wait events are used when calling archive_command, archive_cleanup_command, restore_command and recovery_end_command. (Fujii Masao)
- Correct the name of the wait event for SLRU buffer I/O for commit timestamps. This wait event is named CommitTsBuffer according to the documentation, but the code had it as CommitTSBuffer. Change the code to match the documentation, as that way is more consistent with the naming of related wait events. (Alexander Lakhin)
- Re-activate reporting of wait event SLRUFlushSync. Reporting of this type of wait was accidentally removed in code refactoring. (Thomas Munro)

Version 16
- Add wait event SpinDelay to report spinlock sleep delays (Andres Freund)
- Create new wait event DSMAllocate to indicate waiting for dynamic shared memory allocation. Previously this type of wait was reported as DSMFillZeroWrite, which was also used by mmap() allocations. (Thomas Munro)
- Allow parallel application of logical replication. Wait events LogicalParallelApplyMain, LogicalParallelApplyStateChange, and LogicalApplySendData were also added. Column leader_pid was added to system view pg_stat_subscription to track parallel activity. (Hou Zhijie, Wang Wei, Amit Kapila)
- Have wal_retrieve_retry_interval operate on a per-subscription basis. Previously the retry time was applied globally. This also adds wait events >LogicalRepLauncherDSA and LogicalRepLauncherHash. (Nathan Bossart)

Version 17
- Support custom wait events for wait event type “Extension” (Masahiro Ikeda)
Wait Events

Gaps after migrating to Open Source/Community PostgreSQL

1. SQL/Session/Wait Tracing
2. Wait Event Counters and Cumulative Times (and LWLock counters), both instance and session level
3. Wait Event Arguments (object, block, etc)
4. Comprehensive tracking of CPU time (POSIX rusage; avail session level)
5. Ability to find previous SQL for COMMIT/ROLLBACK
   • Needed to identify which transaction is committing
6. On-CPU State
   • SQL Execution Stage (parse/plan/execute/fetch)
   • SQL Execution Plan Identifier in pg_stat_statements
   • Current plan node
7. Progress on long operations (e.g. large seqscan)
8. Better runtime visibility into PLs
I can haz Wait Events?

Solving Problems with Wait Events in PostgreSQL
Solving Problems With Wait Events

Repository of Historical Perf Data (from pg_stat_activity)

Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate WAIT EVENT & STEP Taking The Most TIME
Solving Problems With Wait Events

Repository of Historical Perf Data (from pg_stat_activity)

Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate **WAIT EVENT** & **STEP** Taking The Most **TIME**
Solving Problems With Wait Events

28.2.3. pg_stat_activity

The pg_stat_activity view will have columns for monitoring activities of the backend process.

Table 28.3. pg_stat_activity View

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datid</td>
<td>oid</td>
<td>OID of the database this backend is connected to.</td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td>Name of the database this backend is connected to.</td>
</tr>
<tr>
<td>pid</td>
<td>integer</td>
<td>Process ID of this backend</td>
</tr>
<tr>
<td>leader_pid</td>
<td>integer</td>
<td>Process ID of the parallel group leader if this process is a parallel group worker, or does not participate in parallel work.</td>
</tr>
<tr>
<td>wait_event_type</td>
<td>text</td>
<td>The type of event for which the backend is waiting, if any; otherwise NULL. See Table 28.4.</td>
</tr>
<tr>
<td>wait_event</td>
<td>text</td>
<td>Wait event name if backend is currently waiting, otherwise NULL. See Table 28.5 through Table 28.13.</td>
</tr>
<tr>
<td>state</td>
<td>text</td>
<td>Current overall state of this backend. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• active: The backend is executing a query.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• idle: The backend is waiting for a new client command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• idle in transaction: The backend is in a transaction, but is not currently executing a query.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• idle in transaction (aborted): This state is similar to idle in transaction, except one of the statements in the transaction caused an error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fastpath function call: The backend is executing a fast-path function.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• disabled: This state is reported if track_activities is disabled in this backend.</td>
</tr>
<tr>
<td>backend_xid</td>
<td>xid</td>
<td>Top-level transaction identifier of this backend, if any; see Section 74.1.</td>
</tr>
<tr>
<td>backend_xmin</td>
<td>xid</td>
<td>The current backend’s xmin horizon.</td>
</tr>
<tr>
<td>query_id</td>
<td>bigint</td>
<td>Identifier of this backend’s most recent query. If state is active this field shows the identifier of the currently executing query. In all other states, it shows the identifier of last query that was executed. Query identifiers are not computed by default so this field will be null unless compute_query_id parameter is enabled or a third-party module that computes query identifiers is configured.</td>
</tr>
<tr>
<td>query_text</td>
<td>text</td>
<td>Text of this backend’s most recent query. If state is active this field shows the currently executing query. In all other states, it shows the last query that was executed. By default the query text is truncated at 1024 bytes; this value can be changed via the parameter track_activity_query_size.</td>
</tr>
</tbody>
</table>
Solving Problems With Wait Events

```bash
while true; do
    psql --csv -Xtc "
    SELECT extract(epoch from now()), query, wait_event_type, wait_event
    FROM pg_stat_activity
    WHERE application_name='pgbench' and state='active';
    "
    sleep 15
done >wait_events.csv
```
Solving Problems With Wait Events

Repositories of Historical Performance Data
(Active Session Sampling of Wait Events)

• https://wiki.postgresql.org/wiki/Monitoring

• Amazon RDS Performance Insights
  • RDS for PostgreSQL 10+
  • Aurora PostgreSQL-Compatible Edition 9.6+
    (v10 Wait Events were backported)
  • Rolling 7 days of history is free. Up to 2 years on paid tier.
Solving Problems With Wait Events

Repository of Historical Perf Data (from pg_stat_activity)

Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate WAIT EVENT & STEP Taking The Most TIME
Solving Problems With Wait Events


Figure 3-7. Collecting data that are scoped improperly on the time dimension also conceals the nature of Wallace’s performance problem, even though the data were collected for the correct action scope.

<table>
<thead>
<tr>
<th>count</th>
<th>state</th>
<th>wait_event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>idle</td>
<td>ClientRead</td>
</tr>
<tr>
<td>2</td>
<td>idle in transaction</td>
<td>ClientRead</td>
</tr>
<tr>
<td>3</td>
<td>active</td>
<td>[null]</td>
</tr>
<tr>
<td>4</td>
<td>[null]</td>
<td>AutoVacuumMain</td>
</tr>
<tr>
<td>5</td>
<td>active</td>
<td>WalSenderMain</td>
</tr>
<tr>
<td>6</td>
<td>[null]</td>
<td>[null]</td>
</tr>
</tbody>
</table>

Can you do event+count(*), where active, group by event?

Also a count of idle in transaction
Solving Problems With Wait Events

Repository of Historical Perf Data (from pg_stat_activity)

Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate **WAIT EVENT** & **STEP** Taking The Most **TIME**
14.1. Using EXPLAIN

14.1.1. EXPLAIN Basics
14.1.2. EXPLAIN ANALYZE
14.1.3. Caveats

PostgreSQL devises a query plan for each query it receives. Choosing the right plan to match the query structure and the properties of the data is absolutely critical for good performance, so the system includes a complex planner that tries to choose good plans. You can use the EXPLAIN command to see what query plan the planner creates for any query. Plan-reading is an art that requires some experience to master, but this section attempts to cover the basics.

Examples in this section are drawn from the regression test database after doing a VACUUM ANALYZE, using 9.3 development sources. You should be able to get similar results if you try the examples yourself, but your estimated costs and row counts might vary slightly because ANALYZE's statistics are random samples rather than exact, and because costs are inherently somewhat platform-dependent.

The examples use EXPLAIN's default "text" output format, which is compact and convenient for humans to read. If you want to feed EXPLAIN's output to a program for further analysis, you should use one of its machine-readable output formats (XML, JSON, or YAML) instead.

14.1.1. EXPLAIN Basics

The structure of a query plan is a tree of plan nodes. Nodes at the bottom level of the tree are scan nodes: they return raw rows from a table. There are different types of scan nodes for different table access methods: sequential scans, index scans, and bitmap index scans. There are also non-table row sources, such as VALUES clauses and set-returning functions in FROM,
Solving Problems With Wait Events

(one of many options)
Solving Problems With Wait Events
Solving Problems With Wait Events

DiffStats and ExplainFull (DSEF)

Detailed SQL reports for 3rd party help & support

DiffStats and ExplainFull can generate detailed reports which are useful for performance of a SQL statement, and especially for working with 3rd party tools in the process. It reduces the amount of back-and-forth requests for information in a great deal of commonly useful data about the performance of a SQL statement.

The extension consists of a number of functions which are installed into the database. These functions fall into two broad categories:

1. A function that is a wrapper around "EXPLAIN ANALYZE" - besides ensuring that all planner diagnostics options are used, it also dumps additional information like server version, full planner statistics for all functions and tables referenced by the SQL.

2. A set of functions to capture and report all statistics tracked by the database during a test SQL statement execution

Installation

pairs well with: github.com/awslabs/pg-collector
Solving Problems With Wait Events

Repository of Historical Perf Data (from pg_stat_activity)

Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate WAIT EVENT & STEP Taking The Most TIME
<table>
<thead>
<tr>
<th>Row</th>
<th>Duration</th>
<th>Name</th>
<th>Query</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.130</td>
<td>3,458.595</td>
<td>1.0</td>
<td><code>\text{List}\ (cost=10,044.33, \text{3.846,124.58 rows=1.000 width=8} \ (\text{actual time=7.650, 3.458.595 rows=1.000 loops=1})</code></td>
</tr>
<tr>
<td>2</td>
<td>0.484</td>
<td>3,458.465</td>
<td>1.0</td>
<td><code>\text{Nested Loop}\ (cost=10,044.33, \text{3.803,130.15 rows=1.329 width=8} \ (\text{actual time=7.350, 3.458.465 rows=1.000 loops=1})\Join Filter: (gsnk_klikhm_konunu_id = bsnk_klikhm_konunu_id)</code></td>
</tr>
<tr>
<td>3</td>
<td>3.780</td>
<td>3,457.881</td>
<td>1.0</td>
<td><code>\text{Gather}\ (cost=10,044.33, \text{3.803,240.00 rows=1.329 width=12} \ (\text{actual time=7.884, 3.457.881 rows=1.000 loops=1})\Workers Launched: 7 \Workers Completed: 7</code></td>
</tr>
<tr>
<td>4</td>
<td>0.194</td>
<td>2,727.130</td>
<td>2.1</td>
<td><code>\text{Nested Loop}\ (cost=4.33, \text{3.782,107.10 rows=190 width=12} \ (\text{actual time=11.230, 2.727.130 rows=395 loops=6})</code></td>
</tr>
<tr>
<td>5</td>
<td>0.208</td>
<td>2,711.151</td>
<td>2.1</td>
<td><code>\text{Nested Loop}\Join\ (cost=3.89, \text{3.792,011.51 rows=188 width=16} \ (\text{actual time=10.980, 2.711.151 rows=395 loops=6})</code></td>
</tr>
<tr>
<td>6</td>
<td>1.200</td>
<td>2,699.439</td>
<td>1.8</td>
<td><code>\text{Hash Join}\ (cost=0.31, \text{3.779,129.86 rows=17.69 width=24} \ (\text{actual time=6.50, 2.699.439 rows=875 loops=8})\Hash Cond: (gsnk_klikhm_konunu_id = bsnk_klikhm_konunu_id)\Join Filter: (gsnk_servisi_dentene_sayis = pmsn_servisi_kanunen_sayis AND (gsnk_gonderim_zaman = pusch_geceari_i_kuruur) \text{ Durum =} new) AND (COALESCE(pmsn_servisi_kanunen_sayis,zzz) \text{ zaman} = timezone)\ (interv)\ \text{ps_servisi_kanunen_sayis} + pmsn_servisi_kanunen_sayis + pmsn_servisi_kanunen_sayis \text{ Durum = new})</code></td>
</tr>
<tr>
<td>7</td>
<td>0.522</td>
<td>2,699.102</td>
<td>1.5</td>
<td><code>\text{Hash Join}\ (cost=2.01, \text{3.772,255.75 rows=636.866 width=44} \ (\text{actual time=0.021, 2.699.102 rows=975 loops=8})\Hash Cond: (psnk_klikhm_konunu_id = psnk_klikhm_konunu_id)</code></td>
</tr>
<tr>
<td>8</td>
<td>18.408</td>
<td>2,688.523</td>
<td>1.5</td>
<td><code>\text{Nested Loop}\ (cost=0.57, \text{3.772,820.81 rows=636.866 width=44} \ (\text{actual time=5.937, 2.688.523 rows=975 loops=6})</code></td>
</tr>
<tr>
<td>9</td>
<td>3,867.668</td>
<td>3,867.668</td>
<td>1.0</td>
<td><code>\text{Parallel Seq Scan on push_control ps (cost=0.00, 2.569,405.84 rows=666,071 width=16} \ (\text{actual time=0.180, 3.867.668 rows=689,808 loops=5})\Filter: (iklikhm_durum = 6)\Rows Removed By Index: 11,760,432</code></td>
</tr>
<tr>
<td>10</td>
<td>202.267</td>
<td>202.267</td>
<td>0.0</td>
<td><code>\text{Index Scan}\ using push_gonderim_pkey on push_gonderim_pkey (cost=0.57, 1.70 rows=1 width=32} \ (\text{actual time=0.007, 2.569,405.84 rows=666,071 width=16} (actual time=0.180, 3.867.668 rows=689,808 loops=5)\Filter: (gsnk_klikhm_konunu_id = pusch_geceari_i_kuruur)\Index Cond: (gsnk_klikhm_konunu_id = pusch_geceari_i_kuruur)</code></td>
</tr>
<tr>
<td>11</td>
<td>0.008</td>
<td>0.008</td>
<td>1.0</td>
<td><code>\text{Hash}\ (cost=1.21, 2.11 rows=6 width=8} \ (\text{actual time=0.036, 0.025 rows=7 loops=3})\Buckets: 1,024 Batches: 0.0 Memory Usages: 0.0 KB</code></td>
</tr>
<tr>
<td>12</td>
<td>0.049</td>
<td>0.049</td>
<td>1.0</td>
<td><code>\text{Seq Scan on push_abon_pbkey ps (cost=0.00, 1.21 rows=7 width=8} \ (\text{actual time=0.048, 0.049 rows=7 loops=4})\Buckets: 1,024 Batches: 0.0 Memory Usages: 0.0 KB</code></td>
</tr>
<tr>
<td>13</td>
<td>0.025</td>
<td>0.025</td>
<td>1.3</td>
<td><code>\text{Hash}\ (cost=1.21, 1.99 rows=3 width=8} \ (\text{actual time=0.133, 0.137 rows=4 loops=4})\Buckets: 1,024 Batches: 0.0 Memory Usages: 0.0 KB</code></td>
</tr>
<tr>
<td>14</td>
<td>0.122</td>
<td>0.122</td>
<td>1.3</td>
<td><code>\text{Seq Scan on push_abon_pbkey ps (cost=0.00, 1.21 rows=3 width=4} \ (\text{actual time=0.105, 0.112 rows=4 loops=4})\Buckets: 0.0 Batches: 0.0 Memory Usages: 0.0 KB</code></td>
</tr>
<tr>
<td>15</td>
<td>20.504</td>
<td>20.504</td>
<td>1.0</td>
<td><code>\text{Index Scan}\ using range_index pkey on range_index (cost=0.57, 0.70 rows=1 width=8} \ (\text{actual time=0.021, 0.021 rows=1 loops=7})\Index Cond: (psnk_klikhm_konunu_id = pusch_geceari_i_kuruur)\Filter: (psnk_klikhm_konunu_id = pusch_geceari_i_kuruur)</code></td>
</tr>
<tr>
<td>16</td>
<td>15.785</td>
<td>15.785</td>
<td>1.0</td>
<td><code>\text{Index Only Scan}\ using mobih_chkr f_klikhm_konunu on mobih_chkr fc (cost=0.44, 0.48 rows=1 width=4} \ (\text{actual time=0.040, 0.040 rows=1 loops=4})\Index Cond: (f_klikhm_konunu_id = pusch_geceari_i_kuruur)\Heap Fetches: 78</code></td>
</tr>
<tr>
<td>17</td>
<td>0.000</td>
<td>0.000</td>
<td>1.0</td>
<td><code>\text{Materialize}\ (cost=0.00, 1.70 rows=20 width=6} \ (\text{actual time=0.000, 0.000 rows=0 loops=1})</code></td>
</tr>
<tr>
<td>18</td>
<td>0.000</td>
<td>0.000</td>
<td>1.0</td>
<td><code>\text{Seq Scan}\ on push_klikhm_konunu ps (cost=0.00, 1.60 rows=20 width=6} \ (\text{actual time=0.000, 0.000 rows=0 loops=1})</code></td>
</tr>
</tbody>
</table>
Solving Problems With Wait Events

How to run short
ALTER TABLE without long locking
concurrent queries
110 views | 0 comments

How much RAM is PostgreSQL using?
97 views | 0 comments

Now you can \d table not only in psql :)
92 views | 0 comments

Next type of node is "Index Scan".

This type of scan seems to be very straightforward, and most people understand when it is used at least in one case:

```
explain analyze select * from pg_class where oid = 1247;
```

```
<table>
<thead>
<tr>
<th>QUERY PLAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Scan using pg_class_oid_index on pg_class (cost=0.15..8.17 rows=1 width=202) (actual time=0.007..0.007 rows=1 loops=1)</td>
</tr>
<tr>
<td>Index Cond: (oid = 1247::oid)</td>
</tr>
<tr>
<td>Total runtime: 0.077 ms</td>
</tr>
<tr>
<td>(3 rows)</td>
</tr>
</tbody>
</table>
```

That is – we have index that matches the condition, so PostgreSQL does:

- opens the index
- in the index if finds where (in table data) there might be rows that match given condition
- opens table
- fetches row(s) pointed to by index
- if the rows can be returned – i.e. they are visible to current session – they are returned
RDS for PostgreSQL wait events

The following table lists the wait events for RDS for PostgreSQL that most commonly indicate performance problems, and summarizes the most common causes and corrective actions.

<table>
<thead>
<tr>
<th>Wait event</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock:Relation</td>
<td>This event occurs when a query is waiting to acquire a lock on a table or view that's currently locked by another transaction.</td>
</tr>
<tr>
<td>Lock:transactionid</td>
<td>This event occurs when a transaction is waiting for a row-level lock.</td>
</tr>
<tr>
<td>Lock:tuple</td>
<td>This event occurs when a backend process is waiting to acquire a lock on a tuple.</td>
</tr>
<tr>
<td>LWLock:BufferMapping</td>
<td>This event occurs when a session is waiting to associate a data block with a buffer in the shared buffer pool.</td>
</tr>
<tr>
<td>LWLock:buffer_content</td>
<td>This event occurs when RDS for PostgreSQL is waiting for other processes to finish their input/output (I/O) operations when concurrently trying to access a page.</td>
</tr>
<tr>
<td>LWLock:LockManager</td>
<td>This event occurs when a session is waiting to read or write a data page in memory while another session has that page locked for writing.</td>
</tr>
<tr>
<td>LWLock:lock_mananger</td>
<td>This event occurs when the RDS for PostgreSQL engine maintains the shared lock's memory area to allocate, check, and deallocate a lock when a fast path lock isn't possible.</td>
</tr>
<tr>
<td>Timeout:PgSleep</td>
<td>This event occurs when a server process has called the \texttt{pg_sleep} function and is waiting for the sleep timeout to expire.</td>
</tr>
<tr>
<td>Timeout:VacuumDelay</td>
<td>This event indicates that the vacuum process is sleeping because the estimated cost limit has been reached.</td>
</tr>
</tbody>
</table>
Solving Problems With Wait Events

LWLock_managers (LWLock-manager)

This event occurs when the RDS for PostgreSQL engine maintains the shared lock's memory area to allow

Topics
Supported engine versions
Context
Likely causes of increased waits

Fast path locking

To reduce the overhead of locks that are taken and released frequently, but that rarely conflict, backends following criteria:

- They use the DEFAULT lock method.
- They represent a lock on a database relation rather than a shared relation.
- They are weak locks that are unlikely to conflict.
- The engine quickly verifies that no conflicting locks can possibly exist.

The engine can't use fast path locking when either of the following conditions is true:

- The lock doesn't meet the preceding criteria.
- No more slots are available for the backend process.

To tune your queries for fast path locking, you can use the following query.

```
SELECT count(*), pid, mode, fastpath FROM pg_locks
WHERE fastpath IS NOT NULL
GROUP BY 4, 5, 2
WHERE pid_m <> mode;
```
Scenario:

Small Bank, Lots of Business
Small Bank, Lots of Business

1. BEGIN;
2. UPDATE accounts
   SET abalance = abalance + :delta
   WHERE account = :account;
3. SELECT balance FROM accounts
   WHERE account = :account;
4. UPDATE tellers
   SET balance = balance + :delta
   WHERE teller = :teller;
5. UPDATE branches
   SET balance = balance + :delta
   WHERE branch = :branch;
6. INSERT INTO history
   VALUES (:teller, :branch, :account,
           :delta, CURRENT_TIMESTAMP);
7. END; (COMMIT TRANSACTION)

Our bank is small because we only have 10 branches

Bank branches (can scale this)

10 tellers
work at each branch

100,000 accounts
opened at each branch

Very important for regulators!

Audit history
A Measure of Transaction Processing Power
Anon Et Al
February 1985

ABSTRACT

Three benchmarks are defined: Sort, Scan and DebitCredit. The first two benchmarks measure a system's input/output performance. DebitCredit is a simple transaction processing application used to define a throughput measure: Transactions Per Second (TPS). These benchmarks measure the performance of diverse transaction processing systems. A standard system cost measure is stated and used to define price/performance metrics.

TABLE OF CONTENTS

Who Needs Performance Metrics? ........................................... 2
Our Performance and Price Metrics ...................................... 4
The Sort Benchmark .......................................................... 6
The Scan Benchmark .......................................................... 7
The DebitCredit Benchmark .................................................. 8
Observations on the DebitCredit Benchmark ......................... 11
Criticism ........................................................................... 11
Summary ........................................................................... 13
References .......................................................................... 14

1 A condensed version of this paper appears in Datamation, April 1, 1985. This paper was scanned from the Tamdem Technical Report TR 85.2 in 2001 and reformatted by Jim Gray.

DebitCredit Benchmark

The Sort and Scan benchmarks have the virtue of simplicity. They can be ported to a system in a few hours if it has a reasonable software base - a sort utility, a Cobol compiler, and a transactional file system. Without this base, there is not much sense considering the system for transaction processing.

The DebitCredit transaction is a more difficult benchmark to describe or port - it can take a day or several months to install depending on the available tools. On the other hand, it is the simplest application we can imagine.

A little history explains how DebitCredit became a de facto standard. In 1973 a large retail bank wanted to put its 1,000 branches, 10,000 tellers and 10,000,000 accounts online. They wanted to run a peak load of 100 transactions per second against the system. They also wanted high availability (central system availability of 99.5%) with two data centers.

The bank got two bids, one for $5M from a minicomputer vendor and another for $25M from a major-computer vendor. The mini solution was picked and built [$Good]. It had a $50K/TPS cost whereas the other system had a $250K/TPS cost. This event crystallized the concept of cost/TPS. A generalization (and elaboration) of the bread-and-butter transaction to support those 10,000 tellers has come to be variously known as the TPI, ETI, or DebitCredit transaction [$Gray].

Details, namely the communication protocol (x.25) and presentation services.

The DebitCredit application has a database consisting of four record types. History records are 50 bytes, others are 100 bytes.

- 1,000 branches .1 MB random access
- 10,000 tellers 1 MB random access
- 10,000,000 accounts 1 GB random access
- a 90 day history 10 GB sequential access

Has the flow:

TRANSACTION
READ MESSAGE FROM TERMINAL (100 bytes)
Small Bank, Lots of Business

```
pgbench --initialize --scale=10

for CLIENT in 1 2 5 10 25 50
    100 200 400 800; do
gbench --client=$CLIENT
    --progress=2 --time=30
done 2>&1 | tee benchmark.log
grep '(clients:|progress: 30)' benchmark.log | paste -
```

Xeon Ice Lake, 1 socket (processor), 32 cores, 64 threads
512 GiB memory
The Old Method: Counter and Ratio Metrics

pgbench --client=100 --progress=5 --time=9999

```
while true; do
  psql csv -Xtc "
  SELECT extract(epoch from now()),
  sum(blks_read) as heap_read,
  sum(blks_hit) as heap_hit,
  sum(blks_hit) / (sum(blks_hit) + sum(blks_read)) as ratio
  FROM pg_stat_database;
  " -c "
  SELECT pg_stat_reset();
  "
  sleep 5;
done;
```
The Right Method: Wait Events

Database load Sliced by Waits

Mar 10 23:58 UTC
- Other: 0, 0%
- LWLock:ProcArray: 0, 0%
- LWLock:BufferContent: 0, 0%
- LWLock:LockManager: 2, 2%
- Timeout:VacuumTruncate: 1, 1%
- IO:WALSync: 1, 1%
- Client:ClientRead: 3, 3%
- LWLock:WALWrite: 6, 6%
- Lock:tuple: 27, 28%
- Lock:transactionid: 55, 57%
- CPU: 2, 2%
- Total DB load: 97
- Max vCPUs: 64

Load by waits (AAS)

SQL statements
- UPDATE pgbench_tellers SET tbalance = tbalance + ? WHERE tid = ?
- UPDATE pgbench_branches SET bbalance = bbalance + ? WHERE bid = ?
The Right Method: Wait Events

1. BEGIN;
2. UPDATE accounts
   SET abalance = abalance + :delta
   WHERE account = :account;
3. SELECT balance FROM accounts
   WHERE account = :account;
4. UPDATE tellers
   SET balance = balance + :delta
   WHERE teller = :teller;
5. UPDATE branches
   SET balance = balance + :delta
   WHERE branch = :branch;
6. INSERT INTO history
   VALUES (:teller, :branch, :account, :delta, CURRENT_TIMESTAMP);
7. END; (COMMIT TRANSACTION)

Top SQL: UPDATE tellers & UPDATE branches

Top Waits Events: Lock:transactionid & Lock:tuple
   = you have hot records in your table

-- Scheduled job, run every few seconds

UPDATE tellers
   SET balance = (SELECT SUM(delta) 
      FROM history 
      WHERE h.teller = t.teller 
      GROUP BY teller)

UPDATE branches
   SET balance = (SELECT SUM(balance) 
      FROM accounts 
      WHERE a.branch = b.branch 
      GROUP BY branch)

Lock:transactionid

The Lock:transactionid event occurs when a transaction is waiting for a row-level lock.

Topics
- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

The event Lock:transactionid occurs when a transaction is trying to acquire a row-level lock that has already been granted to a transaction that is running at the same time. The session that shows the Lock:transactionid wait event is blocking because of this lock. After the blocking transaction ends in either a COMMIT or ROLLBACK statement, the blocked transaction can proceed.

The multiversion concurrency control semantics of RDS for PostgreSQL guarantee...
The Right Method: Wait Events

```
for CLIENT in 1 2 5 10 25 50 100 200 400 800; do
  pgbench --client=$CLIENT --progress=2 --time=30
  --builtin=simple-update
  done 2>&1 | tee benchmark-2.log
egrep '(clients:|progress: 30)' benchmark-2.log | paste -
```

---

**pgbench-step4-step5-job.sql:**

```
UPDATE pgbench_tellers t
  SET tbalance = (SELECT sum(h.delta)
                   FROM pgbench_history h
                   WHERE h.tid=t.tid
                   GROUP BY h.tid)
;
UPDATE pgbench_branches b
  SET bbalance = (SELECT sum(a.abalance)
                  FROM pgbench_accounts a
                  WHERE a.bid=b.bid
                  GROUP BY a.bid)
;
pgbench --initialize --scale=10
pgbench --no-vacuum --client=1 --rate=1
         --progress=5 --time=9999
         --file=pgbench-step4-step5-job.sql
```

---

Optimized application: 5x speedup
The Right Method: Wait Events

Midjourney: May 2023 Incident

8 TB non-partitioned table
8,000-10,000 QPS
Outbound Logical Replication

Four weeks after minimal downtime migration to a partitioned table

Two incidents (4 days apart) of critically elevated application error rates, caused by severe & sudden database performance degradation

AAS & Wait Events History used to quickly identify contention point, greatly accelerating mitigation.

source: www.kylehailey.com
More Scenarios

CPU
• Overall rate of work: Review SQL execution plans, check for plan flips and optimize total blocks accessed.

DataFileRead, BufferIO
• I/O Read Path: Review SQL execution plans, check for plan flips and optimize total blocks accessed.

WALWrite
• I/O Write Path: Check commit rate, volume of change.

For more info, RDS docs on wait events

transactionid, relation, etc.
• Hot records: check top SQL or pg_locks during contention, review application flow of updating records.

BufferContent
• Hot Block in Memory: check foreign keys, optimize contention (e.g. schema redesign, fillfactor, etc).

LockManager
• Lock System High Pressure: Check total number of indexes and partitions involved in tables used by the query, reduce query execution rate, use replicas
Solving Problems With Wait Events

Repository of Historical Perf Data (pg_stat_activity)

Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate **WAIT EVENT** & **STEP** Taking The Most **TIME**
PostgreSQL Happiness Hints

Checksums and Huge Pages Enabled

Connection Pooling
- Centralized (e.g. pgbouncer) and decentralized (e.g. JDBC) architectures
- Recycle server connections (e.g. server_lifetime)
- Limit or avoid dynamic growth when practical – queue at a tier above the DB

Default Limits: Temp Usage, Statement & Idle Transaction Timeout
- Timeouts 5-15 minutes or lower, increase at session level if needed

Scaling
- Measure conn count in hundreds (not thousands), table count in thousands (not hundreds of thousands), relation size in GB (not TB), indexes per table in single digits (not double digits)
- Higher ranges work, but often require budget for experienced & expensive PostgreSQL staff
- To scale workloads, shard across instances or carefully partition tables

Updates and Upgrades
- PostgreSQL quarterly stable "minors" = security and critical fixes only
  - On Aurora: minors can have new development work
- Before major version upgrade, compare plans and latencies of top SQL on upgraded test copy
- Remember to upgrade extensions; it’s not automatic
- Stats/analyze after major version upgrade

Logging
- Minimum 1 month retention (on AWS: use max retention and publish to Cloudwatch)
- Log autovacuum minimum duration = 10 seconds or lower
- Log lock waits
- Log temp usage when close to the default limit
- On AWS: autovacuum force logging level = WARNING

Multiple Physical Data Centers (= Multi-AZ on AWS)

Physical Backups
- Minimum 1 month retention
- Regular restore testing

Logical Backups (at least one)
- Scheduled exports/dumps and redrive/replay
- Logical replication

Active Session Monitoring (= Performance Insights on AWS)
- Save snapshots of pg_stat_activity making sure to include wait events
- Keep historical data, minimum 1 month retention (hopefully much more)

SQL and Catalog and Other Database Statistics Monitoring
- Preload pg_stat_statements
- Save snapshots of pg_stat_statements and key statistics
  - Exec plans (e.g. auto_explain or others), relation sizes (bytes & rows incl catalogs), unused indexes
  - Rates: tuple fetch & return, WAL record & fpi & byte, DDL, XID, subtransaction, multitct, conn
- Keep historical data, minimum 1 month retention (hopefully much more)

OS Monitoring (= Enhanced Monitoring on AWS)
- Granularity of 10 seconds or lower (1 second if possible)
- Keep historical data, minimum 1 month retention (hopefully much more)

Alarms
- Average active sessions (= dbload cloudwatch metric on AWS)
- Memory / swap
- Disk space: %space and %inodes (and free local storage on Aurora)
- Hot standby & logical replication lag / WAL size (disk space) on primary
- Unexpected errors in the logs, both database and application tier
- Maximum used transaction IDs (aka time to wraparound)
- Checkpoint: time since latest & warnings in log (doesn’t apply to Aurora)
Thank you!
aws.amazon.com/rds/postgresql