Logging Plan of the Running Query

Atsushi Torikoshi
torikoshia@oss.nttdata.com
X: @atorik_shi

Motivation

Being able to get the plan of running query helps determine whether a suboptimal execution plan is the root cause of long runtimes. It will be useful in production environments where reproducing long-running queries or attaching a debugger is impractical.

Currently, tools like EXPLAIN and auto_explain only output the plan after the query has completed. This means users often have no choice but to cancel long-running queries without knowing whether the plan caused the slowdown.

If we could obtain the execution plan mid-query, it would provide crucial insight into whether the plan itself is problematic.

What the proposed patch enables

This patch allows you to obtain the plan of a running query from client backends, using its PID:

```
 =# SELECT pg_log_query_plan(201116);
```

Then in the logs:

```
 LOG:  query plan running on backend with PID 201116 is:
   Query Text: SELECT * FROM pgbench_accounts;
   Seq Scan on public.pgbench_accounts
(cost=0.00..52787.00 rows=2000000 width=97)
```

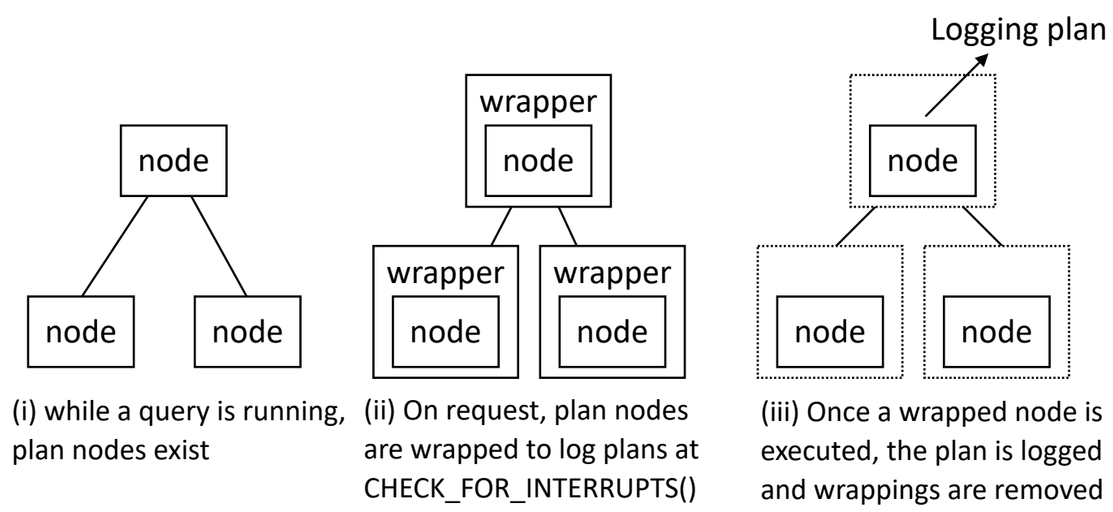
Patch design

This patch aims to:

(1) Safely get the plan of running query.
(2) Minimize overhead.
(3) Provide a minimal initial implementation as a foundation.

The patch enables to wrap each plan node with a function that call EXPLAIN logic. When a plan logging is requested, this function is injected into each plan node at CHECK_FOR_INTERRUPTS().

Once any of the wrapped plan nodes is executed, the current plan is logged, and all wrappers are removed to avoid repeated output.



(1) Safety

Our initial idea was to send a signal to the target backend process, which would invoke EXPLAIN logic at the next CHECK_FOR_INTERRUPTS() call. However, we realized during prototyping that EXPLAIN is complex and may not be safely executed at arbitrary interrupt points. Therefore, we moved to a safer design using controlled plan-node wrapping.

(2) Overhead

An alternative approach -- always generating and storing an EXPLAIN output in advance -- seems impractical due to the overhead. While introducing a GUC to control whether to capture EXPLAIN output might help, it assumes prior knowledge of which queries need to be monitored, which seems difficult in practice.

Current design avoids such overhead by injecting the plan logging only when explicitly requested.

(3) Minimal Functionality

While I understand the value of more advanced features such as getting live execution progress or viewing output via SQL views instead of logs, I believe achieving just (1) and (2) is not easy work and starting with a minimal scope is easier to review and commit.

Collaboration Wanted

- Patch review.
- Coordination with related patches, especiallycalled Progressive EXPLAIN. As described in (3), I believe it is beneficial to first complete and stabilize this minimal-scope patch before merging broader functionality, but there may be other ways.

Commitfest item:

PGConf.dev 2025