

Request-Aware Compute Placement for Serial Database Requests

nullreference

Abstract

A web request can be close to its user but far from its database. Serial database calls make that distance part of the request’s critical path. In a test with two application compute nodes and a separate PostgreSQL node, the faster compute node changed between one and three serial calls. A routing proxy reduced the 12-call request’s p50 by 1.858 ms against user-first routing in a repeated run. A network-only placement rule matched a profiled rule within measurement noise. The result supports request-aware placement for this test service. It does not establish a general scheduler benefit or test replica placement.

1 Introduction

Conventional routing can choose the compute node with the shortest path from a client. A database-backed request also waits for the network path between its compute node and the database. When database calls depend on earlier results, their waits add to the critical path.

Prior work has used measured network latency and application performance to inform cluster placement [1]. Other work has targeted end-user latency when it places services at the edge [2]. This report measures a narrower case: whether database call count changes the better compute node for one Rails service.

The main result is workload-dependent. Compute A was faster for a one-call request. Compute B was faster for requests with three and twelve serial calls. In routed tests, database-aware policies improved the 12-call request against user-first routing. A profiled model did not improve on the simpler network model.

2 Method

2.1 Testbed and routes

The same Rails service ran on two application compute nodes. PostgreSQL ran on a separate database node. The client paths to the compute nodes differed. The test kept the database primary fixed and sent all policies to the same strong-read database.

The application had two representative routes. The `/feed` route made three sequential SQL calls. The `/feed-naive` route made twelve: one account lookup, one article query, and ten feedback queries. A separate route issued a selected number of serial `SELECT 1` calls to measure the compute-node crossover directly.

2.2 Placement rules

The user-first rule kept every request on the compute node with the shortest client path. The network-only rule selected the node with the lower estimated sum of client-to-compute latency and serial database waits:

$$C_{\text{network}}(h, n) = R_{\text{client},h} + nR_{\text{db},h}. \quad (1)$$

Here, h is a compute node, n is the number of serial database calls, and R denotes the measured round-trip time for the named path. The profiled rule also added a compute-node-specific route residual measured from a zero-query route. This residual captures unmodeled route work. It does not isolate CPU time.

The calibration used 500 requests for each route and compute node after warmup. The held-out test used 500 new requests for each route and compute node. The routed test interleaved and randomized policies. Each policy and route received 500 requests. All 3,000 requests in the initial routed test returned HTTP 200.

For the direct crossover test, one client sent 500 requests for each call count and compute node. The test used $n \in \{1, 3, 12\}$ in the repeated run. It calibrated the rules again before the repeat. The reported confidence intervals are 95% bootstrap intervals for differences in median latency.

3 Results

3.1 The faster compute node changed with call count

In the repeated direct test, Compute A was faster by 0.135 ms at one serial call. Compute B was faster by 0.243 ms at three calls and 2.271 ms at twelve calls. The 95% intervals for the Compute-A-minus-Compute-B median difference were $[-0.190, -0.095]$ ms, $[0.178, 0.314]$ ms, and $[2.086, 2.518]$ ms, respectively. The observed crossover fell between one and three calls in this test.

Table 1: Direct route p50 by compute node in the repeated crossover test.

Serial calls	Compute A p50 (ms)	Compute B p50 (ms)	Faster node
1	1.624	1.759	Compute A
3	2.586	2.343	Compute B
12	6.648	4.376	Compute B

3.2 Database-aware routing helped the longer request

The repeated routed test showed little difference for one call. At three calls, the database-aware rules selected Compute B and reduced p50 by about 0.08 ms against user-first routing. At twelve calls, they reduced p50 by about 1.86 ms.

Table 2: Routed request p50 in the repeated test.

Calls	User-first	Network-only p50 (ms)	Profiled	Selected node
1	1.978	1.973	1.966	Compute A
3	2.949	2.887	2.865	Compute B
12	6.776	4.977	4.918	Compute B

The profiled rule reduced the three-call p50 by 0.083 ms against user-first routing. Its 95% interval was $[0.010, 0.187]$ ms. For twelve calls, the reduction was 1.858 ms, with an interval of $[1.616, 2.050]$ ms. For one call, the 0.012 ms difference was not clear, with an interval of $[-0.052, 0.080]$ ms.

The profiled rule did not improve on the network-only rule. Their p50 differences were 0.007, 0.022, and 0.059 ms at one, three, and twelve calls. Each 95% interval included zero. Both rules selected the same compute node for each call count in the repeated run.

An earlier routed run showed the same pattern. The profiled rule changed the 12-call p50 from 6.356 ms under user-first routing to 5.690 ms. Its estimated reduction was 0.665 ms, with a 95% interval of [0.522, 0.780] ms. The repeat produced a larger change. These values come from separate runs and should not be treated as a pooled estimate.

4 Limits

The test used one Rails service, two application compute nodes, one separate database primary, and one client path. It did not model load variation, failures, replica placement, or production traffic. The measured crossover can change with network paths, database state, and compute-node load.

The result supports a narrow conclusion: serial database work can change which compute node gives a lower request p50. The network-only rule captured the measured placement change. The test did not show that route profiling improves placement beyond that simpler rule.

5 Reproducibility

The published artifact includes the planner, proxy, Rails route, and measured data. The [planner](#), [proxy](#), and [Rails route](#) are available with the [direct measurements](#), [direct summary](#), [routed measurements](#), and [routed summary](#). The [data guide](#) defines the published role labels and files.

References

- [1] D. A. Popescu et al., “Network Latency and Application Performance Aware Cluster Scheduling in Data Centers,” *IEEE Network*, 2022. <https://www.cl.cam.ac.uk/~dap53/pubs/popescu-ieee-network-2022.pdf>
- [2] C. Centofanti et al., “Taming Latency at the Edge: A User-Aware Service Placement Approach,” *Computer Networks*, vol. 247, article 110444, 2024. <https://doi.org/10.1016/j.comnet.2024.110444>