January 2023

Improving Speed and Reliability of Database Queries

Ed Gogel

Improving Speed and Reliability of Database Queries

ABSTRACT

Low latency databases with high query volumes and large numbers of connections often have multiple redundant communication paths from clients to database servers. When queries fail, clients that submit queries have difficulty identifying the source of query failures and are unable to take actions to ensure high performance and service availability. This disclosure describes database techniques to tune performance and identify reasons for failed queries. A metadata layer within the database is used by the database server to provide anticipated time-to-serve based on the current server load and the complexity of the query. Clients can use the time-to-serve data to plan queries, set dynamic timeouts, detect network black holes, attempt alternate communication pathways, etc. Further, clients can share amongst each other the current time-to-serve and server health, enabling multiple clients to plan respective queries based on the response by the server to a single client. Similar techniques apply to identify communication failures generally in data transfer situations.

KEYWORDS

- Database overload
- Remote database
- Query failure
- Network failure
- Network black hole
- Dynamic timeout
- Smart backoff
- Denial of service

BACKGROUND

Databases generally have a centralized data store, referred to as a server. Multiple applications connect to the server using client(s). Clients maintain connections to the server and submit requests to read or write data using a standard query format. Such connections leverage redundant communication paths to multiple servers that are often geographically separated.

When submitting queries to database servers through a client, a timeout is usually specified. The timeout is usually a number of seconds or milliseconds during which the client expects a response from the server handling the request. If no response is received from the server within the specified time, the client resends the request, once again waiting for a response.

Many strategies exist for handling timeouts, the most common being exponential backoff. Under exponential backoff, the client sets sequentially higher timeout values. Increasing the timeout values in subsequent requests prevents the server from becoming overwhelmed and overloaded by retry requests, and provides it time to catch up with service requests. Exponential backoff does not work well in real-time systems with short timeout values, e.g., those with timeouts of less than 1-2 seconds since clients in such systems require faster response times.

Multiple failure modes exist. For optimal performance, the failure mode is detected and handled correctly. Some example failure modes include:

- The server is busy and the query is taking longer than usual.
- The network is unreliable in one or both directions, and either the query or the response was lost.
- Too many clients are connecting to the server, causing a temporary overload condition.
- The server is unavailable due to a hardware or software failure.

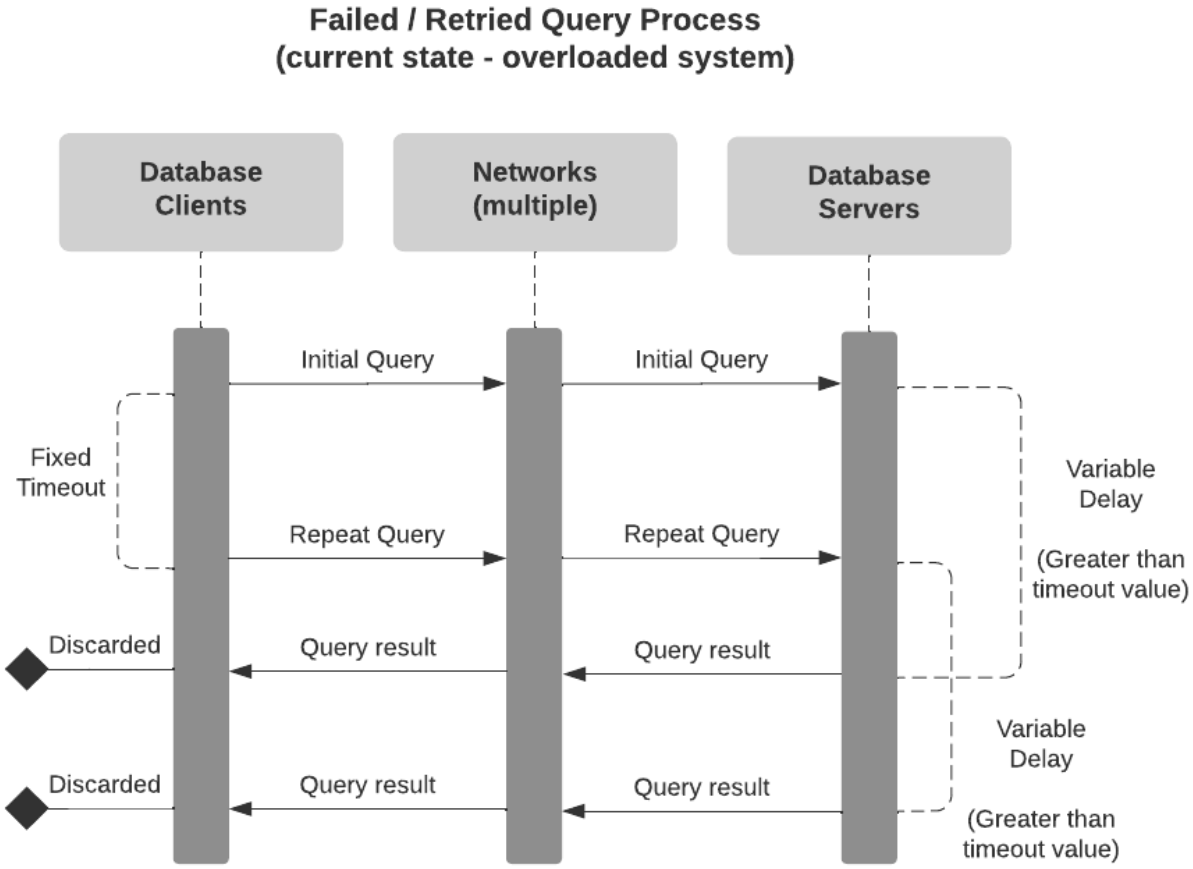

Fig. 1: An example failure mode: A delayed response from the database server that exceeds a fixed timeout results in a repeat query and discarded results. The client does not know the precise reason for the apparent lack of response from the server, and its repeated queries only serve to further overload an already overloaded server and needlessly increase network traffic.

The client does not know the reason for a lack of response from a server: The lack of response can be due to a communication problem, due to an overloaded server responding slowly (as illustrated in Fig. 1), etc. Shortening timeout is not recommended, as it can cause the server to become overloaded during communication failures, with multiple clients reconnecting and resubmitting queries en masse, in turn leading to a denial of service for all clients and an extended, systemic failure. Neither is lengthening timeouts viable, since in cases of communication failure, service availability can be maintained only by quickly resending the

request (via a different communications path as necessary). Often, network failures are not easily detected and can become black holes for network traffic, with one or both parties not detecting the other party is not receiving their messages. Routing protocols often act within seconds to minutes rather than in the milliseconds required by real-time systems.

A naive approach to solving this problem is to measure the average response time of the server from the client and use that as a baseline for timeout values. While this can work in limited cases, some queries take longer to execute because of the size and/or complexity of the request and response. Additionally, in distributed systems, clients might not send queries on a predictable enough cadence to establish reliable average timeout values.

Generally, having static timeouts also does not work well when systems are anticipated to have degraded performance, such as during maintenance windows, during times when data is being duplicated or backed up, when cluster sizes are being adjusted, when indexes are being split and merged, etc. Database clients today have no insight into anticipated wait times and cannot plan accordingly.

DESCRIPTION

This disclosure describes a server metadata layer within databases to proactively communicate with clients either via a subchannel of data or within the response to queries. The metadata layer can be used by the server to provide the time anticipated to service subsequent requests from a client. A real-world analogy is a queue in a grocery store, which enables customers to visually anticipate wait times, choose different queues, or choose a different store entirely to make a purchase. The described server metadata layer provides database clients with similar visibility into server loads and response times at the present moment, enabling clients to plan their queries and communication pathways with the database.

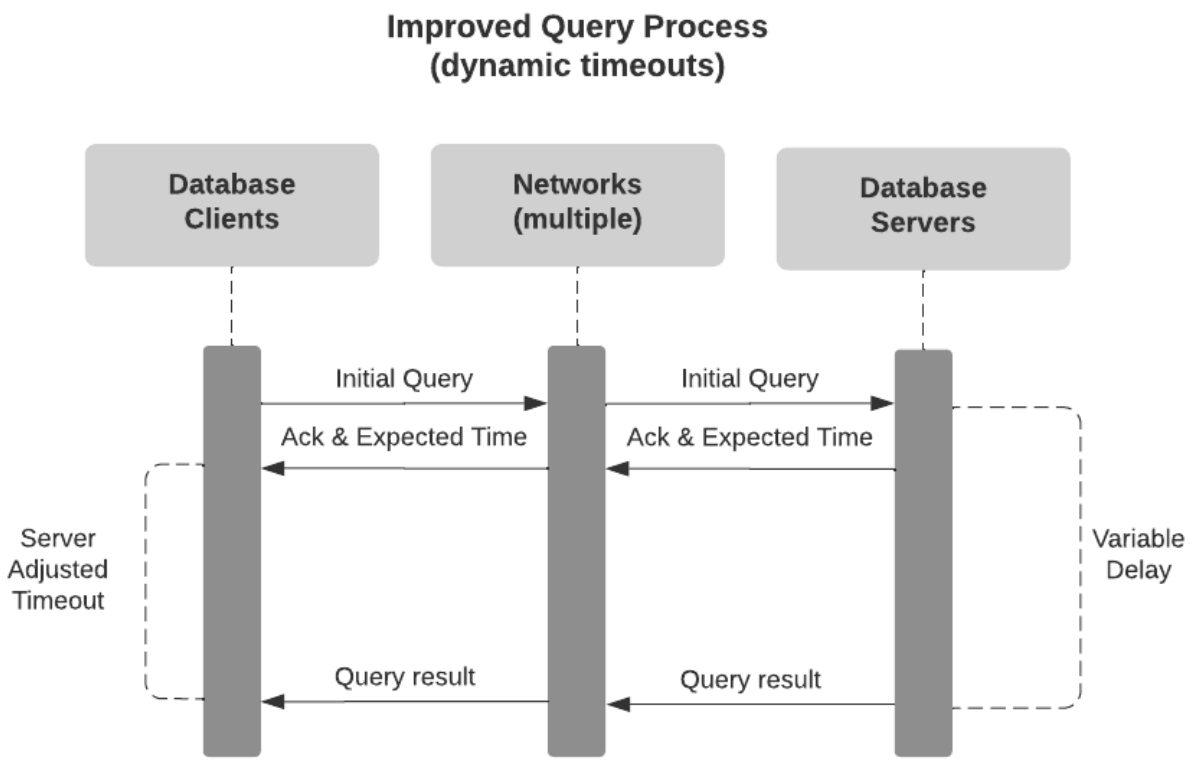

Fig. 2: Dynamic timeouts

As illustrated in Fig. 2, a client submits a request, the server receives the request, and, after accounting for its current load, returns an immediate response that specifies to the client how long it will likely need to wait for a response. In contrast to conventional static timeouts, the client has better knowledge of the expected wait time, enabling the client to plan its queries. For example, if the server is temporarily overloaded and reports a greater-than-normal wait time, the client can determine to not unnecessarily resend queries, which can cause a further overload and potentially lead to a systemic denial of service.

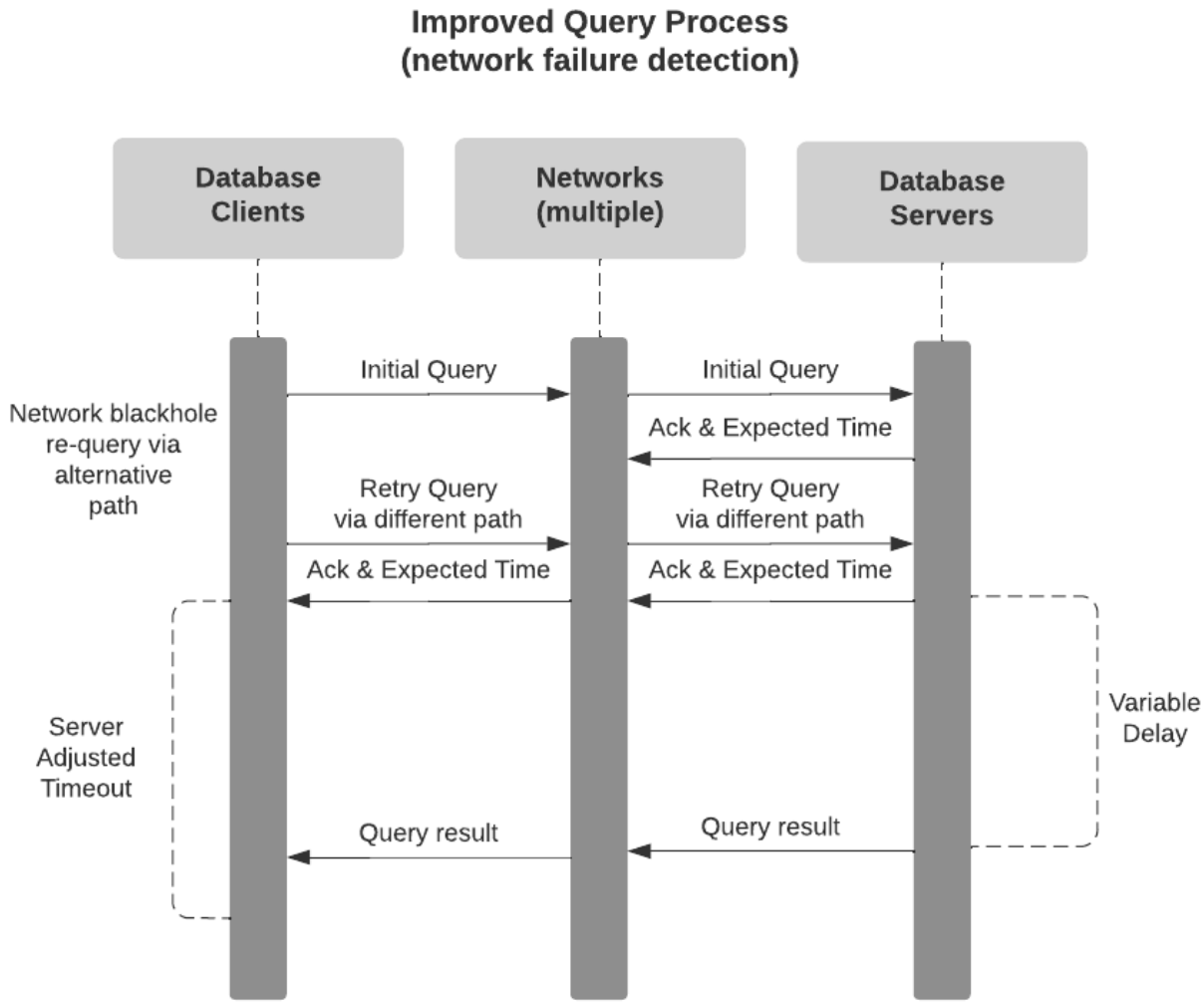

Fig. 3: Network failure detection

As illustrated in Fig. 3, the server metadata layer can also be used to detect and inform the client of the cause of a failed query. Upon receipt of a query, the server creates an execution plan for the query. The execution plan, which can typically be built in a fraction of the time needed to actually execute it and build the full query response, provides an accurate estimate of the time needed to process the query. The time estimate is returned to the client, providing the client insight into timeouts while notifying the client that the query has been received.

In the example of Fig. 3, the server responds immediately to an initial query with an acknowledgment and an expected service time, but the acknowledgment is not received by the

client. The lack of immediate receipt of acknowledgment indicates to the client a network black hole. The client then correctly retries the query via a different path. This time, the acknowledgment from the server is received. In this example, the expected service time reported by the server is larger than normal. The larger-than-normal service time indicates to the client that the server is overloaded. The client sets a correspondingly large timeout such that unnecessary resending of queries that further flood the server is avoided. In this example, the facility of the server sending back an immediate response with expected service time enables the client to detect network black holes and server overload conditions.

Clients can exchange with each other information about time-to-serve estimates and server health using client-to-client communication, e.g., in a mesh, or via a shared database or data store. In this manner, multiple clients can obtain information relating to server health from the response by the server to the query of a single client, an especially useful feature in scenarios where some clients are idle, or after a system restart.

The server can also provide bounding times, such as anticipated response times and worst-case response times. This additional data can help clients to establish the correct timeout values, enabling them to detect network failures such as black holes and providing clients the opportunity to dynamically react and reroute traffic away from failures.

Some advantages of the described techniques include:

- Clients receive immediate responses to their requests notifying them that the query was received and of the anticipated time to process the query.
- Denial-of-service incidents are reduced or eliminated when servers become busy since timeouts are dynamically adjusted by clients.

- Reliability is significantly improved because clients can resubmit queries via other paths when queries are not received by the server or when responses are not received within the timeout provided in the initial response from the server.
- Network black holes or failures can be identified and worked around quickly to maintain service.
- Dynamic timeouts and additional insight into query service times enable clients to behave more intelligently. For example, applications that depend upon the backends using the database clients can notify end-users when responses are expected to be delayed and by how much, improving the overall user experience.

CONCLUSION

Low latency databases with high query volumes and large numbers of connections often have multiple redundant communication paths from clients to database servers. When queries fail, clients that submit queries have difficulty identifying the source of query failures and are unable to take actions to ensure high performance and service availability. This disclosure describes database techniques to tune performance and identify reasons for failed queries. A metadata layer within the database is used by the database server to provide anticipated time-to-serve based on the current server load and the complexity of the query. Clients can use the time-to-serve data to plan queries, set dynamic timeouts, detect network black holes, attempt alternate communication pathways, etc. Further, clients can share amongst each other the current time-to-serve and server health, enabling multiple clients to plan respective queries based on the response by the server to a single client. Similar techniques apply to identify communication failures generally in data transfer situations.