newsletter.scalablethread.com

How to Solve Producer Consumer Problem with Backpressure?

Sidharth

3–4 minutes

In distributed applications, each service can be a data *producer* or *consumer* in a given context. For a system to operate smoothly, consumers must keep up with the producer's data generation speed. Scenarios where producers continue to send data to slow consumers for extended periods (especially in asynchronous systems) without considering whether consumers can consume them result in the classic *Producer-Consumer* problem.

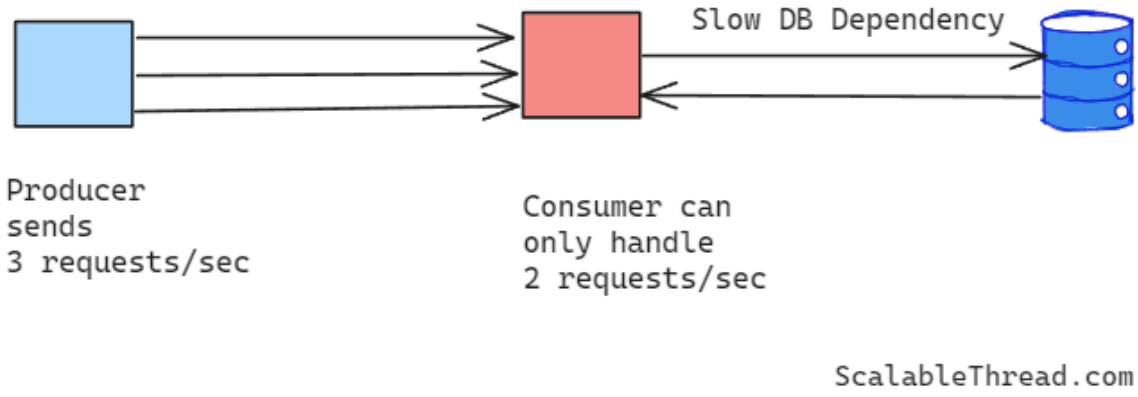
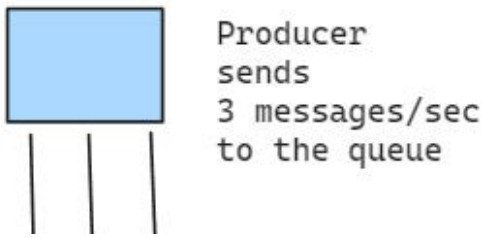


Fig. Overloaded consumer in synchronous communication



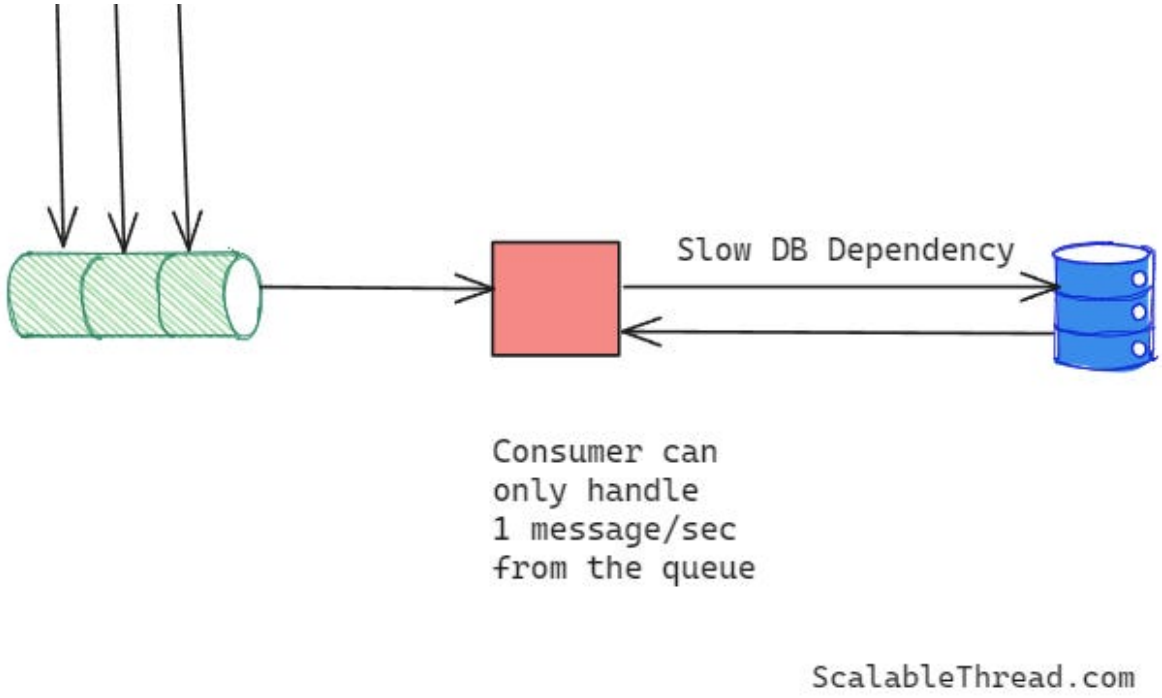


Fig. Overloaded consumer in asynchronous communication

Borrowed from flow control, *Backpressure* is a technique for controlling the data transmission rate between producer and consumer to prevent the producer from overwhelming a slow consumer. It can be applied in three ways:

Dropping (*load shedding*) or sampling the input data is generally not ideal unless the system is fine with data sampling —for example, logging systems, etc.

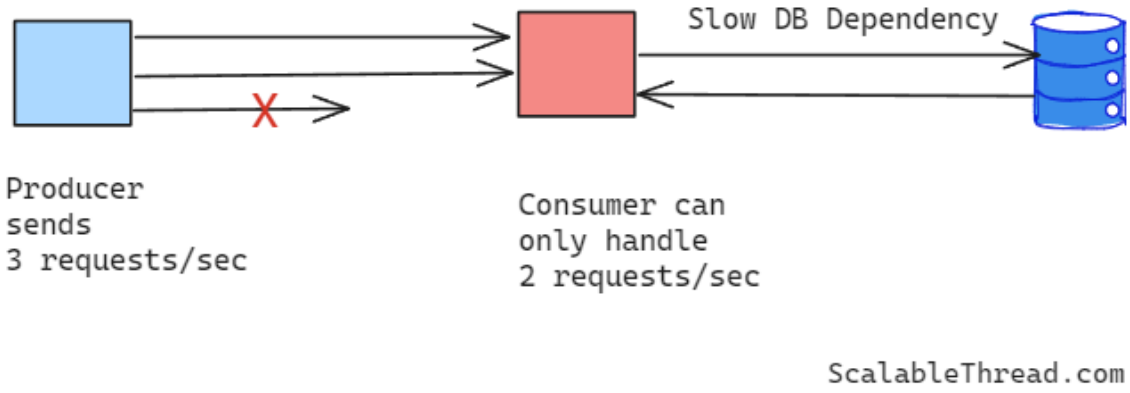


Fig. Consumer load shedding the input in synchronous communication

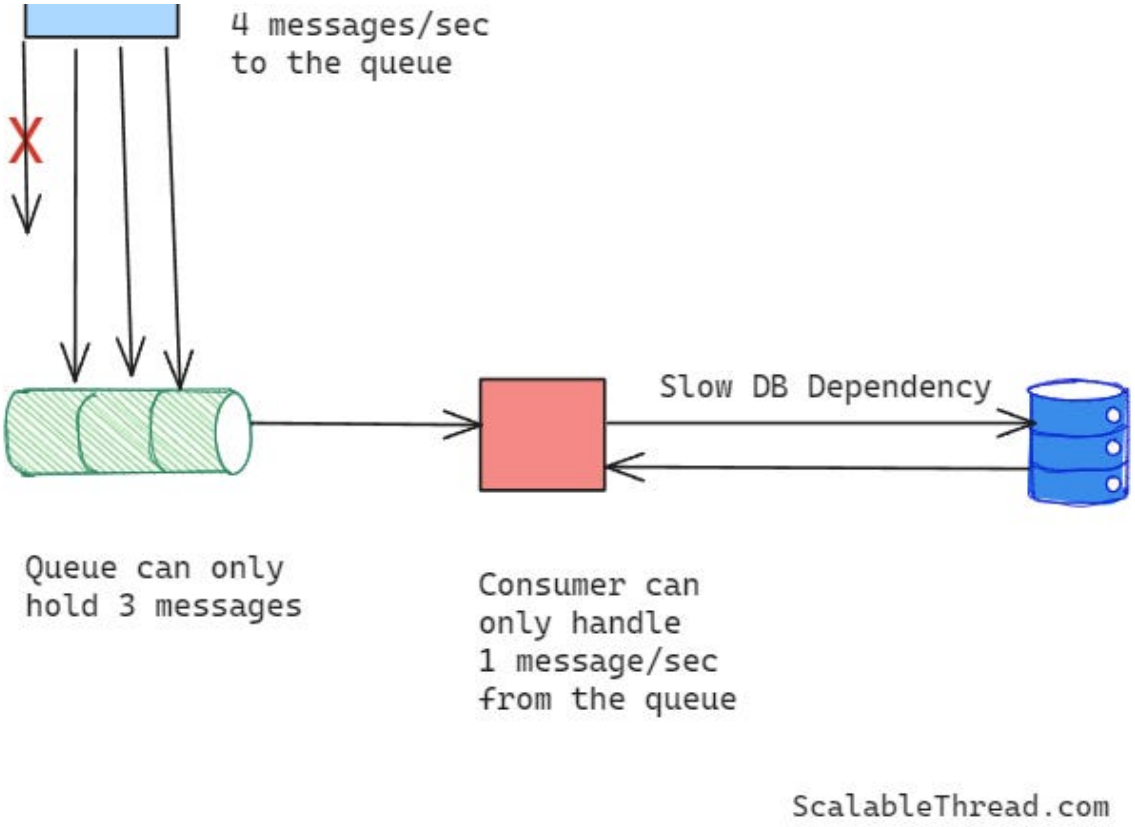
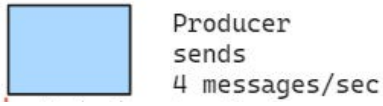


Fig. Message queue load shedding the input once it's full in asynchronous communication.

This is a standard solution. The communication between two services is made asynchronous by connecting them with a message buffer (*queue*). Although this decouples the *producer* and *consumer* and solves the thundering herd problem, it still isn't ideal in a scenario where a producer continues to be faster than the consumer for prolonged periods. When a high throughput producer is connected to a slow consumer through a message queue, it can result in the following:

- **Queue Spilling Data —** Since the queue is a bounded buffer, there can be a scenario when the queue is full, and there is no space for the producer to send more data. This can result in the new data getting dropped from the queue or the producer seeing exceptions when writing to the queue.



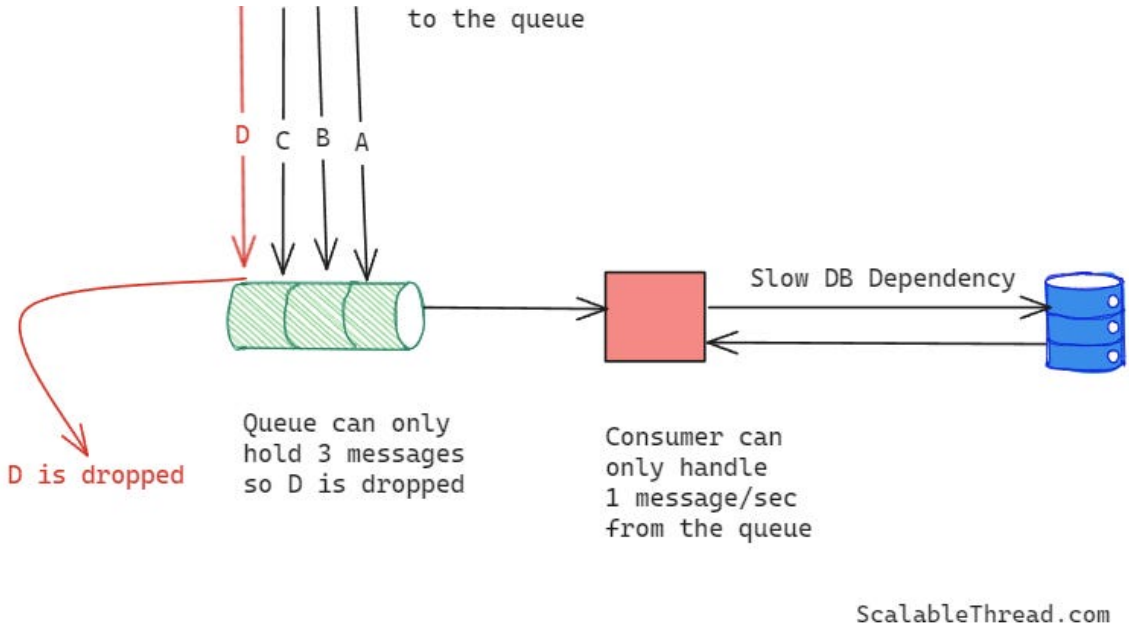
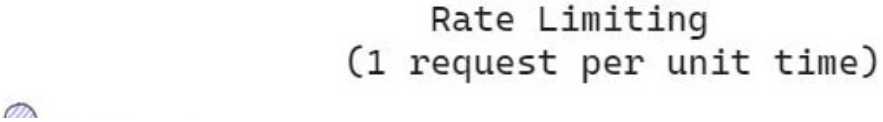


Fig. Queue Dropping Data

- **Out-of-Memory Consumer —** The consumer can see out-of-memory issues if the input batch size for data read from the queue is not fixed and is relative to the data on the queue. In this case, the consumer can try to read more data than it can keep in memory and end up in a crash loop due to out-of-memory exceptions.
- **Increased Latency—**Data sitting in the queue while waiting for the consumer to process it can increase the application's overall latency and storage costs.

Controlling the producer is the most optimal solution for applying Backpressure, as this can bring down the producer's input rate and allow the consumer to complete the processing of existing data. This can be achieved by —

- Similar to flow control, the system allows the consumer to indicate to the producer whether it's ready to receive more data. Solutions like rate-limiting can help safeguard the service's upper limits.



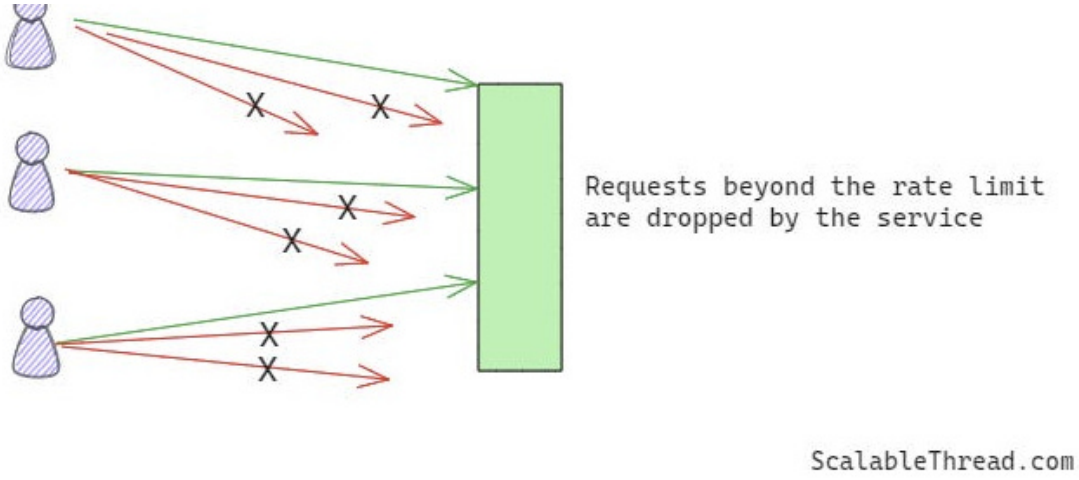
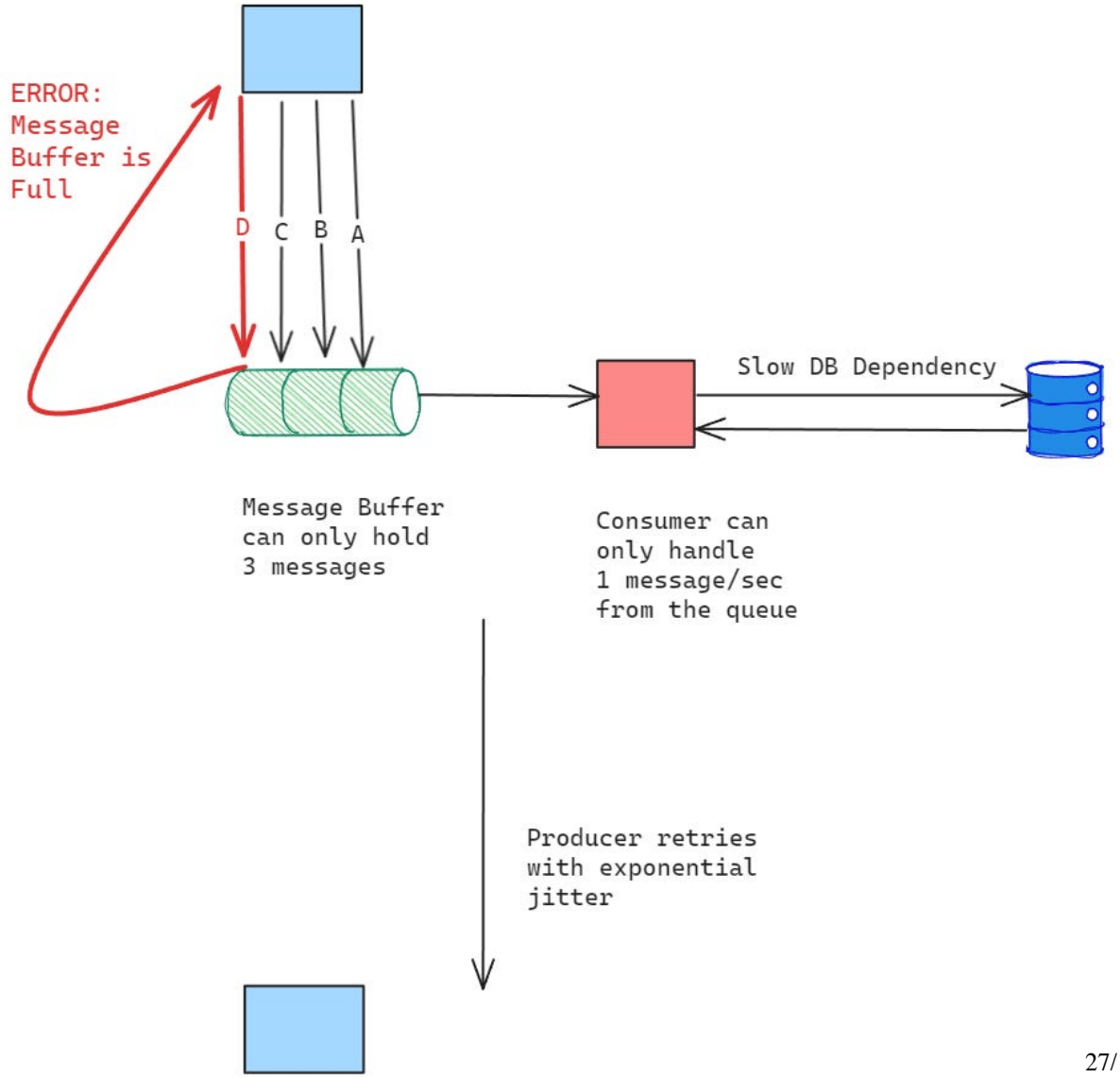


Fig. Rate limiting by the consumer in synchronous communication

- The producer controls its flow based on the error messages received while writing to the message buffer when it's full and retries again with exponential jitter.



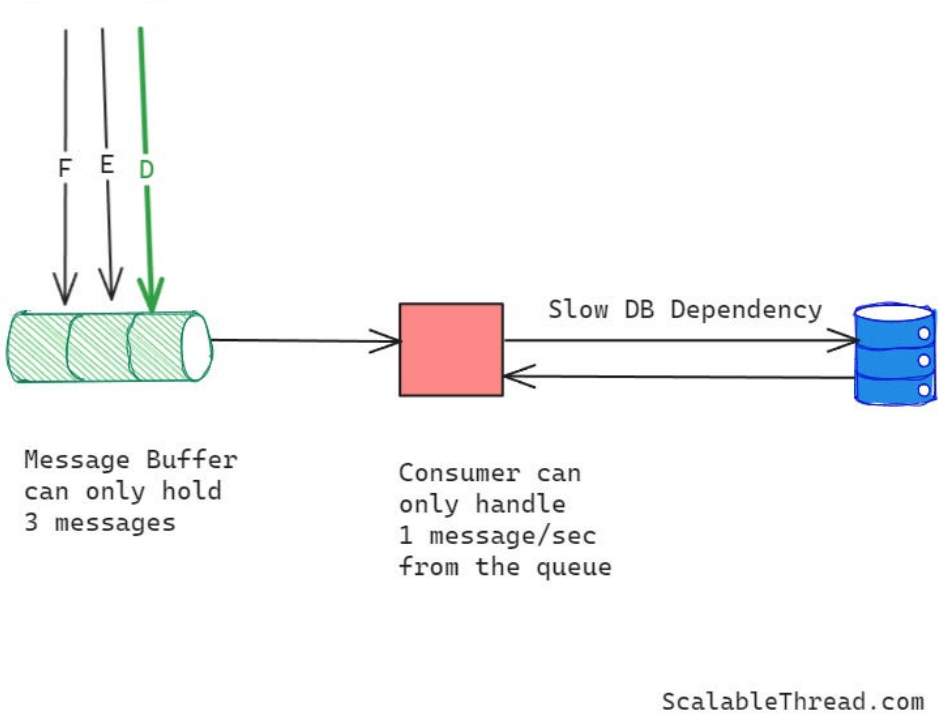


Fig. Producer controls its flow based on the “Buffer Full“ error message from the message buffer and retries with exponential jitter

Subscribe to The Scalable Thread

One well-researched system design concept simplified like you're five, in just 5 minutes, every week! Written by a FAANG software engineer