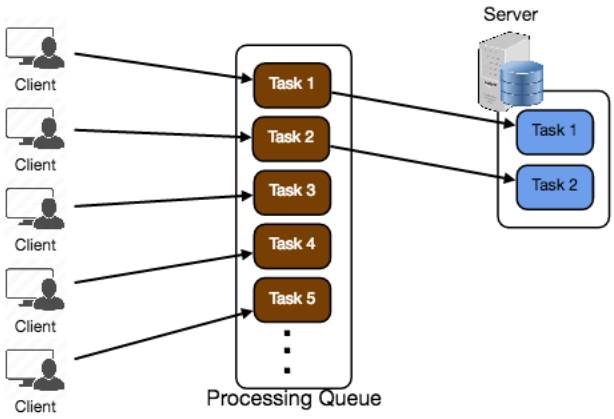
Queues

Queues are used to effectively manage requests in a large-scale distributed system. In small systems with minimal processing loads and small databases, writes can be predictably fast; however, in more complex and large systems writes can take an almost non-deterministically long time. For example, data may have to be written in different places on different servers or indices, or the system could simply be under high load. In such cases where individual writes (or tasks) may take a long time, achieving high performance and availability requires different components of the system to work in an asynchronous way; a common way to do that is with queues.

Let's assume a system where each client is requesting a task to be processed on a remote server. Each of these clients sends their requests to the server, and the server tries to finish the tasks as quickly as possible to return the results to the respective clients. In small systems where one server can handle incoming requests just as fast as they come, this kind of situation should work just fine. However, when the server gets more requests than it can handle, then each client is forced to wait for other clients' requests to finish before a response can be generated.

This kind of synchronous behavior can severely degrade client's performance; the client is forced to wait, effectively doing zero work, until its request can be responded. Adding extra servers to address high load does not solve the problem either; even with effective load balancing in place, it is very difficult to ensure the fair and balanced distribution of work required to maximize client performance. Further, if the server processing the requests is unavailable, or fails, then the clients upstream will fail too. Solving this problem effectively requires building an abstraction between the client's request and the actual work performed to service it.



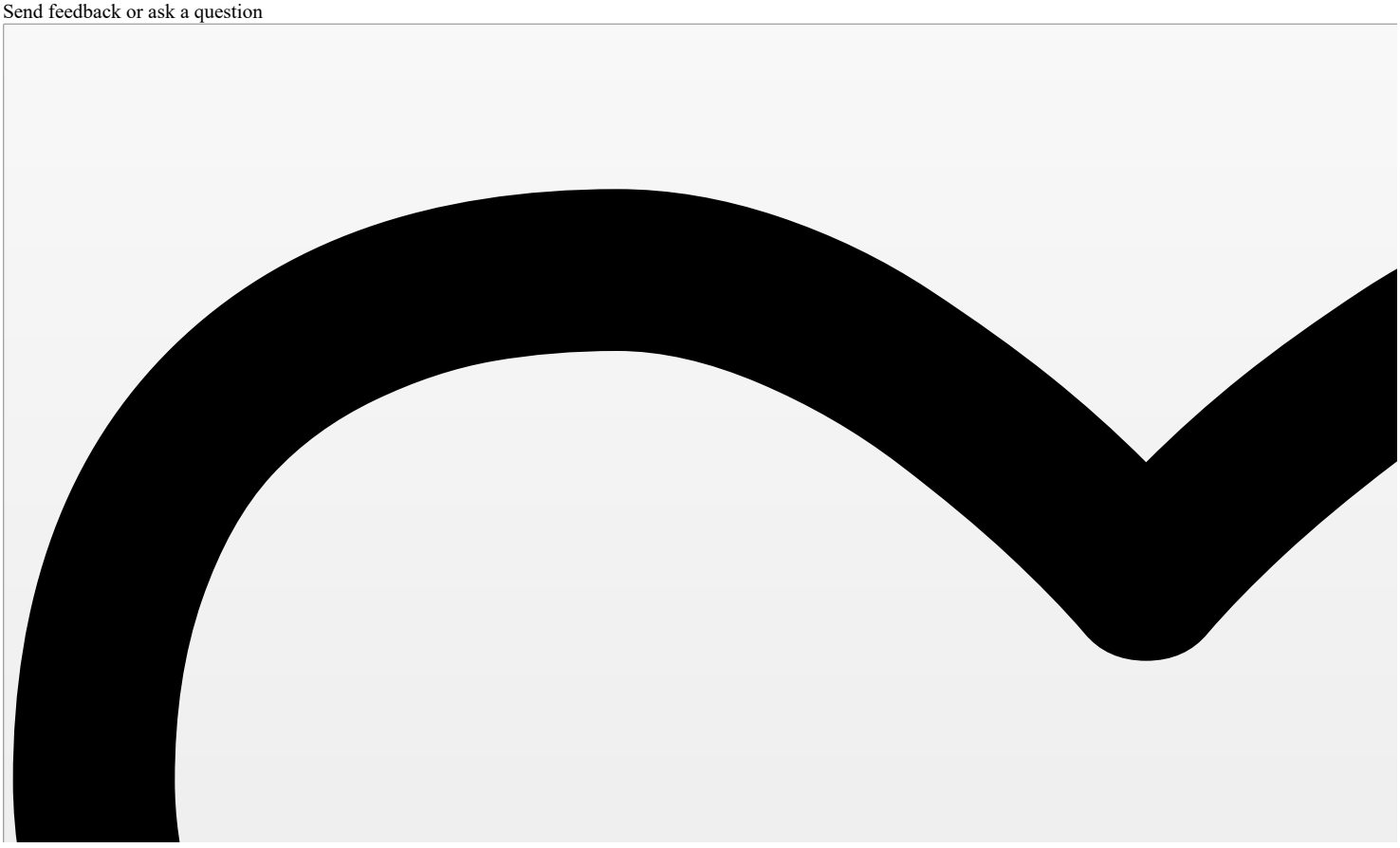
A processing queue is as simple as it sounds: all incoming tasks are added to the queue, and as soon as any worker has the capacity to process, they can pick up a task from the queue. These tasks could represent a simple write to a database, or something as complex as generating a thumbnail preview image for a document.

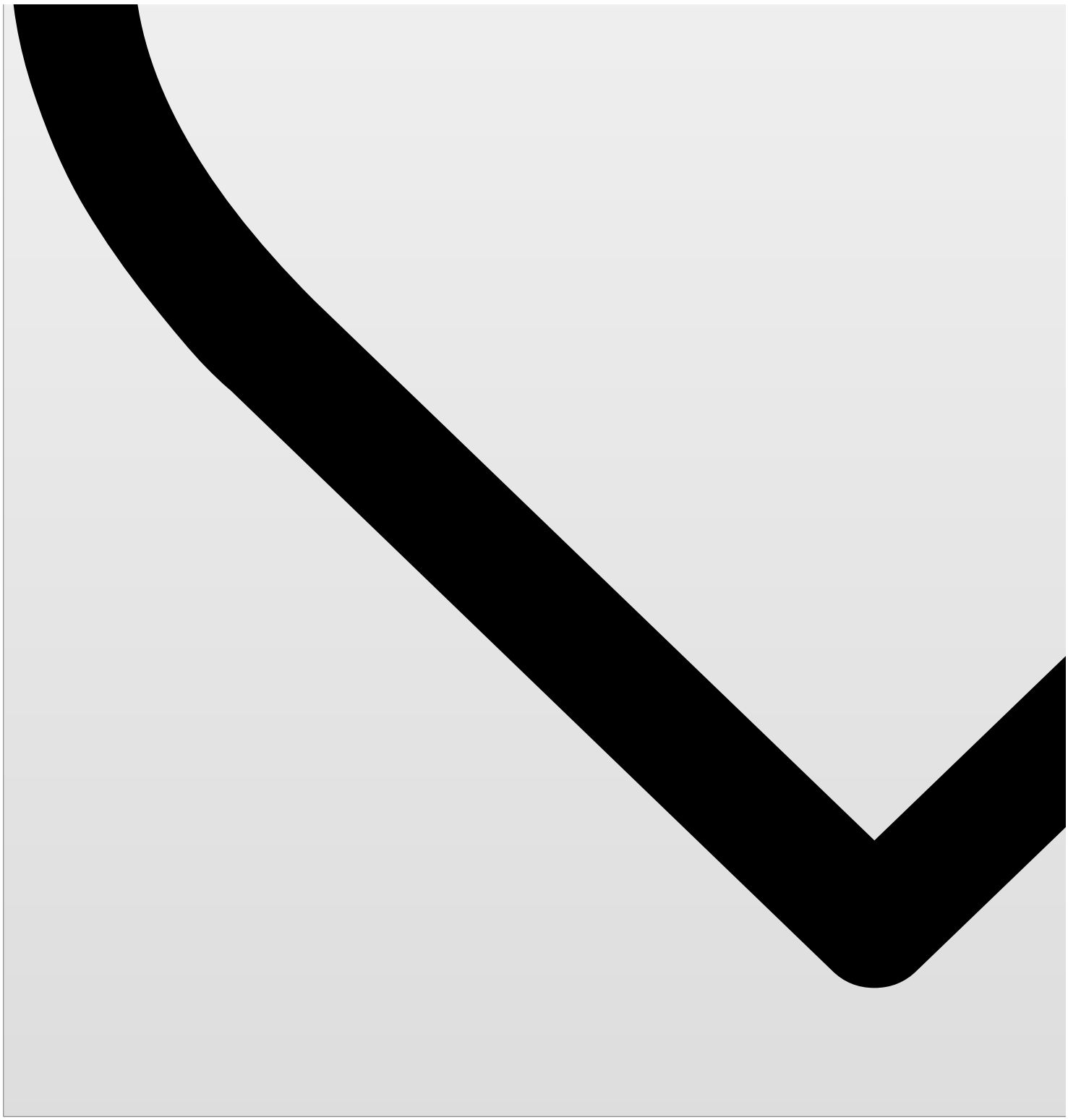
Queues are implemented on the asynchronous communication protocol, meaning when a client submits a task to a queue they are no longer required to wait for the results; instead, they need only acknowledgment that the request was properly received. This acknowledgment can later serve as a reference for the results of the work when the client requires it. Queues have implicit or explicit limits on the size of data that may be transmitted in a single request and the number of requests that may remain outstanding on the queue.

Queues are also used for fault tolerance as they can provide some protection from service outages and failures. For example, we can create a highly robust queue that can retry service requests that have failed due to transient system failures. It is preferable to use a queue to enforce quality-of-service guarantees than to expose clients directly to intermittent service outages, requiring complicated and often inconsistent client-side error handling.

Queues play a vital role in managing distributed communication between different parts of any large-scale distributed system. There are a lot of ways to implement them and quite a few open source implementations of queues available like RabbitMQ, ZeroMQ, ActiveMQ, and BeanstalkD.

Mark as completed
← *Previous* Proxies *Next* → Redundancy and Replication
Send feedback or ask a question





15 recommendations

- Home
- Featured
- Team
- Blog
- FAQ
- Terms of Service
- Contact Us