(3) Simplicity and statelessness[19]

(4) Selectable write progress feedback[20]

Based on these criteria, multiple low-level networking libraries were evaluated. Most systems either supported less than five programming languages (violating (1)) or were built with a strong focus and reliability and therefore traded performance for reliability.

One framework – a very lightweight transparent communication layer called ZeroMQ (ØMQ) (see [45] & [12]) stood out:

(1) Supports 46 different languages[21]

(2) Not only built for latency and throughput comparable to raw socket communication, but also featuring zero-copy-capable lock-free queues that use a transparent API for inter-thread & inter-process-communication and communication over the network. Additionally, the model features fully asynchronous network writes, facilitating the programming of highly efficient yet simple to use software.

(3) A fully stateless yet fully routable framing model that only requires copying the content into binary ØMQ frames of arbitrary size

(4) Provides not only a request/reply pattern but also push/pull and publish/subscribe topology (see [45], also discussed in section 2.3.3)

As shown in figure 2.4, YakDB also uses ØMQ for internal communication. By using the inter-thread *"inproc"* mechanism, YakDB can use an actor-based threading model, facilitating an almost lock-free database design (see [45]). However, a detailed discussion of these design decisions would exceed the scope of this thesis.

Asynchronicity support

As shown in figure 2.5, YakDB allows both synchronous and asynchronous writes[22]. Synchronous writes reduce the likelihood of errors for example due to a file system having reached 100% disk utilization as the client will wait for an error response from the server before sending more datasets. However, asynchronous writes as shown in figure 2.5 provide a clear advantage in performance as the client will continue to compute the next dataset while the database is writing the last value to the disk.

[19] i. e. no explicit state being tied to a connection to increase the robustness of the system.

[20] i. e. support for modes where the server reports that a specific write has been performed successfully, but also for modes where for performance reasons it is only ensured that the network did not fail to deliver the request to the server.

[21] as of 2015-05-07, see [13]

[22] Although technically asynchronous reads are possible, their usage is more complex as the further execution of the program commonly relies on the result of the read request. Therefore, in this thesis only asynchronous write requests are discussed.