

Thread Architecture

immediate







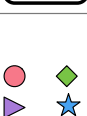
	Stage with either fixed number of threads, or has a limit on the maxim number of threads running.
	Stage without limits on number of threads. A new thread is spawned whenever needed.
	Scheduling point, where requests are queued.
	Asynchronous relationship between stages: upstream stage returns as soon as it hands off request to the downstream stage.
	Synchronous relationship between stages: upstream stage blocks on downstream stage.
	Machine boundary. Communication between different machines consumes network resources.
	Different types of resources. Legend shown in a stage box means that stage consumes the corresponding resource intensively. Multiple stages in one machine contain the same legend or one stage contain multiple legends could both indicate problems

Table 1: **Legend Summary.** *Legends used in thread architecture graphs.*

1 Thread Architecture of Popular Systems

In this section we show the thread architecture of some popular systems, and point out the scheduling problem derived from their architecture.

Table 1 summarizes the legends used in the architecture graphs, and how to read these graphs to identify potential scheduling problems. We omit the stages that are not resource intensive, or not active in normal execution path, due to limited space. We also omit background activities, e.g., compaction, which could be resource-intensive, but can be managed by different mechanisms, such as throttling or utilizing system idle time.

1.1 HBase/HDFS Storage Stack

The thread architecture of the HBase/HDFS storage stack is shown in Figure 1. When HBase clients send queries to the Region Server, the following sequence of actions occurs:

1. The RPC Read threads read data from client connections, parse the data into RPC calls, and queue the RPC calls. The RPC Handle threads take RPC calls from the queue and process them (step 1).
2. If the processing of RPC requires reading data from HDFS, the RPC Handle thread sends a read request

to the HDFS Datanode and blocks until it completes. The Datanode spawns a new thread for each block being read (step $r_1 - r_2$ in the graph). However, RPC Handle thread first checks if the data is stored locally. If it is, HBase will invoke the HDFS short-circuit read mechanism and read data directly within the RPC Handle thread, bypassing the Datanode.¹

3. Similarly, if HDFS metadata operation is needed during the processing of RPC, RPC Handle thread blocks until the operation is finished by the HDFS Namenode (step $m_1 - m_4$).
4. For puts or similar operations involving data modification, RPC Handle thread append WAL entries to the WAL entry queue (step a_1). It then blocks until then WAL entry is persisted.
5. The LOG Append thread fetches WAL entries from the queue and issues writes to HDFS. It does so by putting data to be written in the data packet queue of the corresponding Data Stream thread (step w_1).
6. Data Stream thread sends data to an HDFS Datanode, which spawns a Data Xceiver thread for each block being written (step w_2).
7. Depending on how many copies of data need to be written, Data Xceiver thread may further pass the data to another downstream datanode, which spawns another Data Xceiver thread to write one more copy of the data (step w_3).
8. Data Xceiver thread writes data to disk. For each packet being written to disk, it generates an ack and passes the ack to an Packet Ack thread (step w_4).
9. One Packet Ack thread is generated for each Data Xceiver thread. It collects acks from the downstream datanode, and send acks to either its upstream datanode (step w_5) or to the client issuing writes (step w_6).
10. Within RegionServer (which is the HDFS write client), each Data Stream thread will also spawn a corresponding Ack Process thread, which is responsible for receiving and processing the write acks

¹HBase still contacts the Datanode to get some necessary information, such as the file descriptor. The actually reading, however, is performed locally.

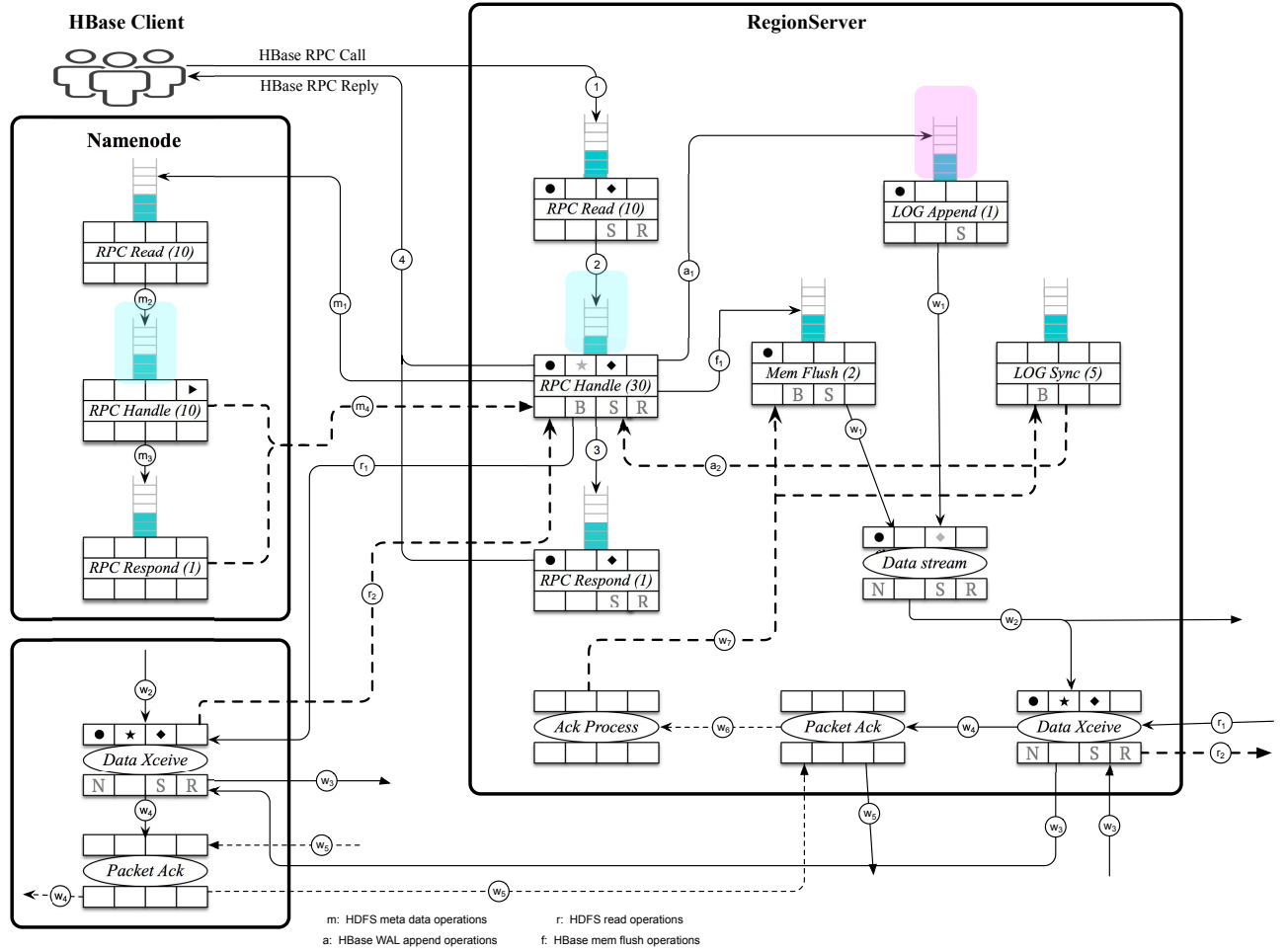


Figure 1: HBase/HDFS Thread Architecture.

- from the Datanode. Once the Ack Process thread receive acks from all relevant data packets, it notifies the LOG sync thread that write is persisted (step w_7).
11. Once LOG Sync thread confirm the WAL entries are persisted, it notifies the RPC Handle thread, who can then proceed (step a_2).
 12. The RPC Handle thread also write changes to the MemStore cache (step f_1).
 13. Once an RPC Handle thread finishes processing one RPC, it passes the result to the RPC Respond thread and continues to process another RPC (step 3). However, if the connection happens to be idle, it will bypass the responder and send the response directly through the connection (step 4).
 14. The RPC Respond thread sends the response back to the client (step 4).
 15. When the MemStore cache is full, the Mem Flush threads write the MemStore content to HDFS. The write process follows the same steps as the writes issued by the LOG Append thread (step $w_1 - w_7$).

The HDFS namenode, which also has the RPC Read, RPC Handle, and RPC Respond stages, works similar as the corresponding stages in RegionServer. However, the namenode RPC Handle threads do not need to invoke operations in other stages; they simply grab the namespace lock and perform HDFS namespace operations.

References