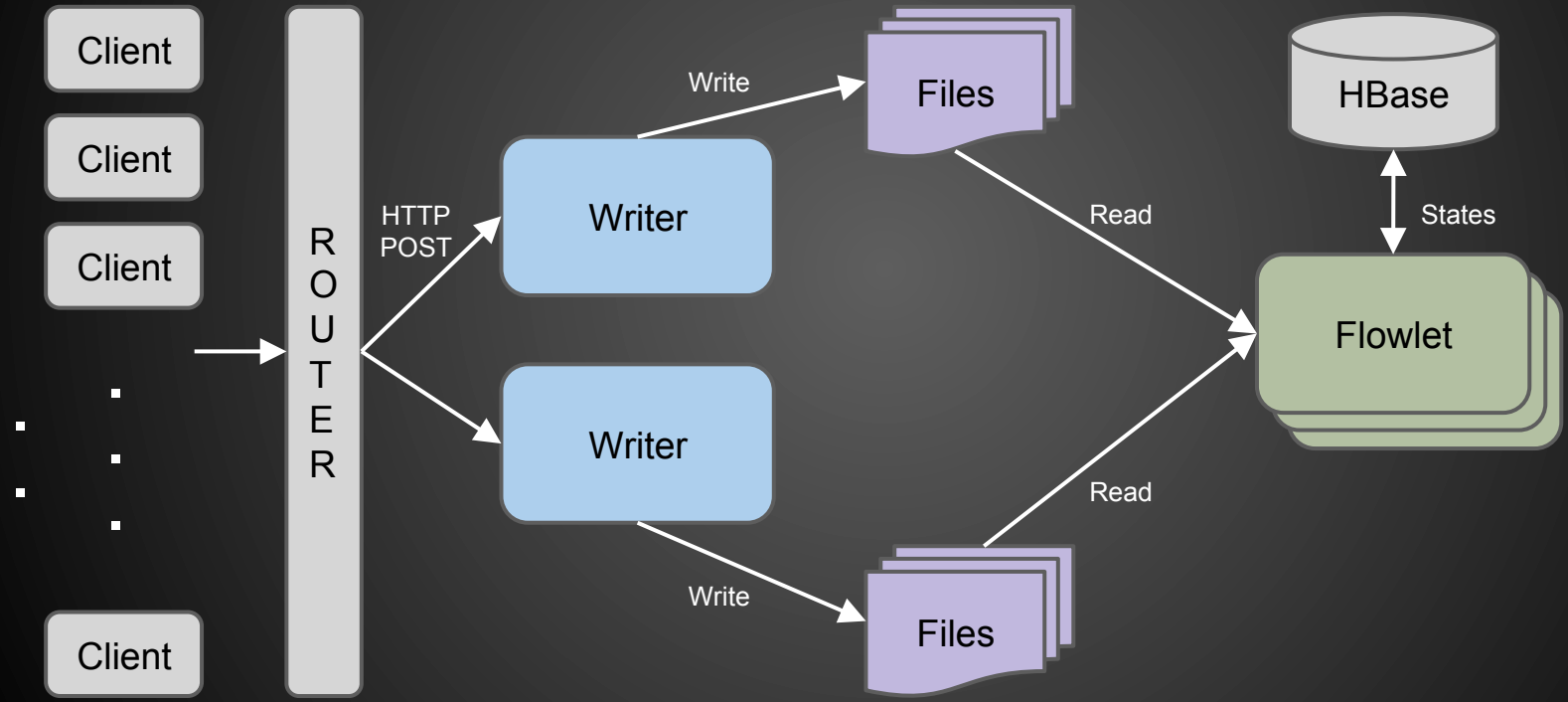
What is Stream?

- Primary means for data collection in Reactor
  - REST API to send individual event
- Consumable by Reactor Programs
  - Flow
  - MapReduce

Why on File?

- Data eventually persisted to file
  - LevelDB -> local file
  - HBase -> HDFS
- Fewer intermediate layer == Performance ++

10K Architecture



Directory Structure

/[stream_name]
    /[generation]
        /[partition_start_ts].[partition_duration]
            /[name_prefix].[sequence].("dat"|"idx")

Directory Structure

/who

Stream name = who

/who/00001

Generation = 1

/who/00001/1401408000.86400

Partition start time = 2014-05-30 GMT

Partition duration = 1 day

File name

- Only one writer per file
  - One file prefix per writer instance
- Don’t use HDFS append
  - Monotonic increase sequence number
  - Open file => find the highest sequence number + 1

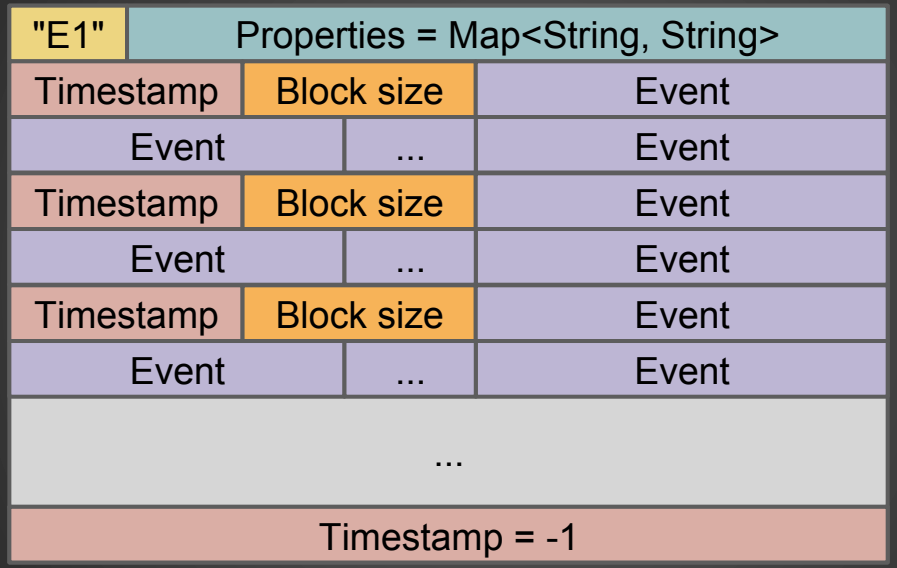
/who/00001/1401408000.86400/file.0.000000.dat

File prefix = “file.0”. Written by writer instance “0”

Sequence = 0. First file created by the writer

Suffix = “dat”, an event file

Event File Format



- Avro binary serialize “Properties” and “Event”
- Event schema stored in Properties

Writer Latency

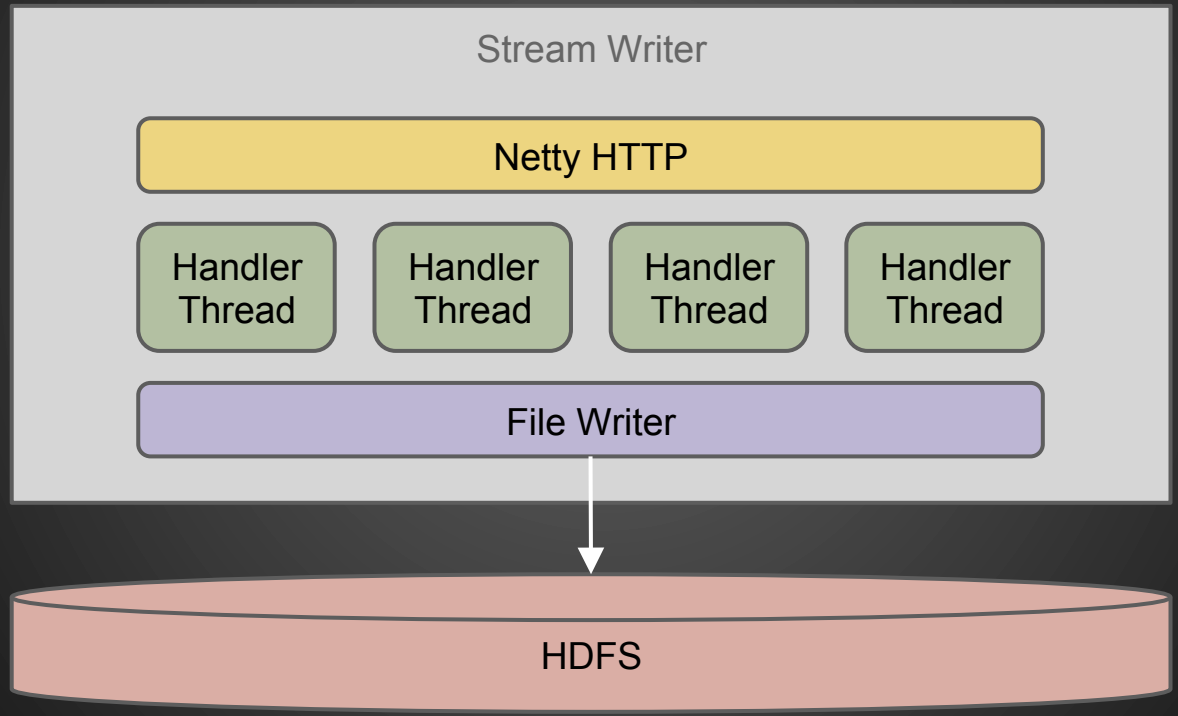
- Latency
  - Speed perceived by a client
  - Lower the better

- Guarantee no data loss
  - Minimum latency == File sync time

Writer Throughput

- Throughput
  - Flow rate
  - Higher the better
- Buffer events gives better throughput
  - Higher latency?
- Many concurrent clients
  - More events buffered write

Inside Writer



How to synchronize access to File Writer?

Concurrent Stream Writer

1. Create an event and enqueue it to a **Concurrent Queue**
2. Use **CAS** to try setting an atomic boolean flag to true
3. If successfully (**winner**), proceed to run step 4-7, **loser** go to step 8
4. Dequeue events and **write to file** until the queue is empty
5. Perform **a file sync** to persist all data being written
6. Set the state of each events that are written to **COMPLETED**
7. Set the atomic boolean back to false
    - Other threads should see states written in step 6 (**happened-before**)
8. If the event owned by this thread is **NOT COMPLETED**, go back to step 2.
    - Call **Thread.yield()** before go to step 2

Correctness

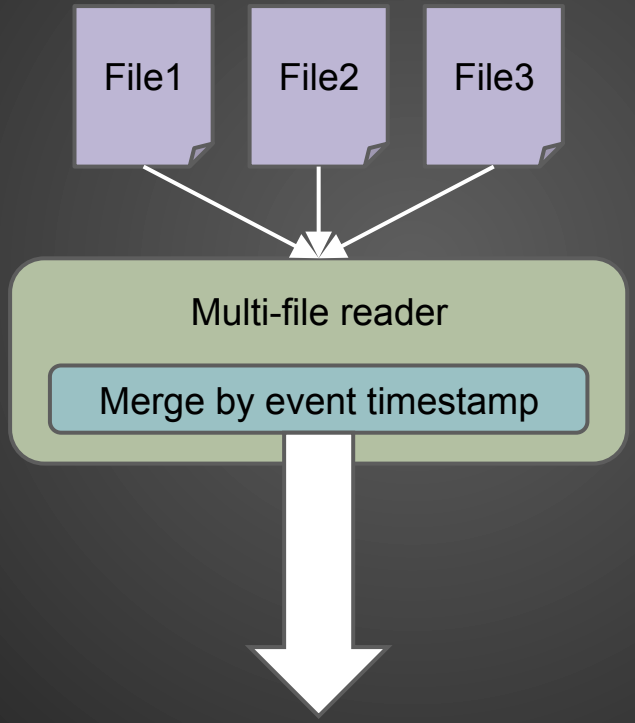
- Guarantee no losing events
  - Winner, always drain queue
    - Own event should be in the queue
  - Losers, either
    - Current winner starts drains after enqueue
    - Loop and retry, either
      - Become winner
      - Other winner start drains

Scalability

- One file per writer process
  - No communication between writers
- Linearly scalable writes
  - Simply add more writer processes

How to tail stream?

Merge on Consume



Tailing HDFS file

- HDFS doesn’t support tail
  - EOFException when no more data
    - Writer not yet closed
  - Re-open DFSInputStream on EOFException
  - Read until seeing timestamp = -1

Writer Crashes

- File writer might crash before closing
  - No tail “-1” timestamp written
- Writer restart creates new file
  - New sequence or new partition
- Reader regularly looks for new file
  - No event read
    - Look for file with next sequence
    - Look for new partition based on current time

Filtering

- ReadFilter
  - By event timestamp
    - Skip one data block
    - TTL
  - By file offset
    - Skip one event
    - RoundRobin consumer

Consumer states

- Exactly once processing guarantee
  - Resilience to consumer crashes
- States persisted to HBase/LevelDB
  - Transactional
  - Key
    - {generation, file_name, offset}
  - Value
    - {write_pointer, instance_id, state}

Consumer IO

- Each dequeue from stream, batch size = N
  - RoundRobin, FIFO (size = 1)
    - ~ (N * size) reads/skips from file readers
    - Batch write of N rows to HBase on commit
  - FIFO (size >= 2)
    - ~ (N * size) reads from file readers
    - O(N * size) checkAndPut to HBase
    - Batch write of N rows to HBase on commit

Consumer State Store

- Per consumer instance
  - List of file offsets
    - [{file1, offset1}, {file2, offset2}]
  - Events before the offset are processed
    - Perceived by this instance
  - Resume from last good offset
  - Persisted periodically in post commit hook
    - Also on close

Consumer Reconfiguration

- Change flowlet instances
  - Reset consumers' states
    - Smallest offset for each file
  - Make sure no events left unprocessed

Truncation

- Atomic increment generation
  - Uses ZooKeeper in distributed mode
    - PropertyStore
      - Supports read-compare-and-set
  - Notify all writers and flowlets
    - Writer close current file writer
      - Reopen with new generation on next write
    - Flowlet suspend and resume
      - Close and reopen stream consumer with new generation

Futures

- Dynamic scaling of writer instances
  - Through ResourceCoordinator
- TTL
  - Through PropertyStore

Thank You