Heisenberg Was on the Write Track

Pat Helland
Salesforce

THE CUSTOMER SUCCESS PLATFORM

Introduction

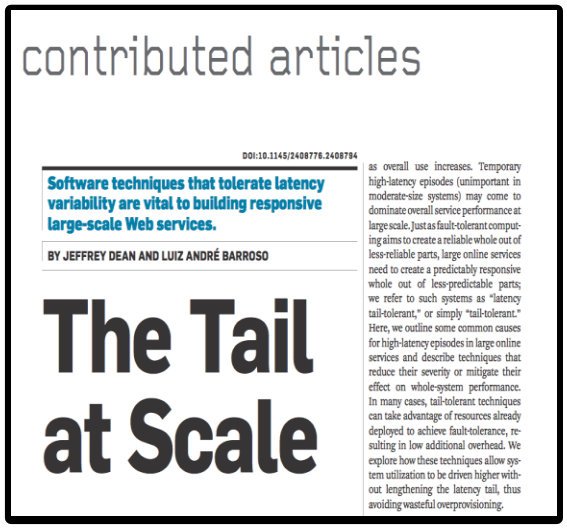
In a Distributed System,

You Can Know *Where* You Write

or You Can Know *When* You Write

but You *Can't Know Both*…

The Tail at Scale… *for Writes?*


contributed articles

DOI:10.1145/2408776.2408794

Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.

BY JEFFREY DEAN AND LUIZ ANDRÉ BARROSO

The Tail at Scale

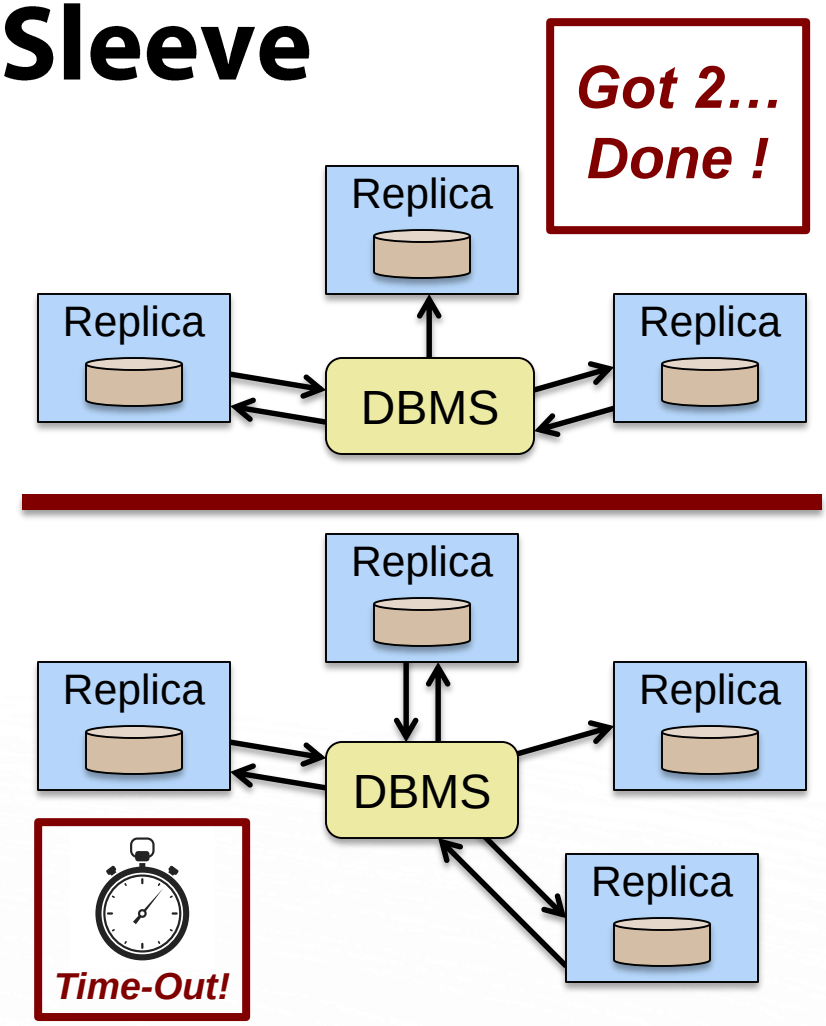
as overall use increases. Temporary high-latency episodes (unimportant in moderate-size systems) may come to dominate overall service performance at large scale. Just as fault-tolerant computing aims to create a reliable whole out of less-reliable parts, large online services need to create a predictably responsive whole out of less-predictable parts; we refer to such systems as "latency tail-tolerant," or simply "tail-tolerant." Here, we outline some common causes for high-latency episodes in large online services and describe techniques that reduce their severity or mitigate their effect on whole-system performance. In many cases, tail-tolerant techniques can take advantage of resources already deployed to achieve fault-tolerance, resulting in low additional overhead. We explore how these techniques allow system utilization to be driven higher without lengthening the latency tail, thus avoiding wasteful overprovisioning.

- **"The Tail at Scale" by Jeff Dean and Luiz Andre Barroso (Google)**
  - ✧ Managing latency at Google
- **Read-only requests (e.g. a portion of a search)**
  - ✧ *Idempotent*: No big deal if the request is issued twice
- **Natural variability in service request timing**
  - ✧ Shared resources, garbage collection, maintenance activities, queuing, etc…
- **Retry each request after 95% wait**
  - ✧ Try a different server… about 5% increase in load
  - ✧ The new server will very likely be fast!

- ***We can do this for WRITES, too!!***
  - ✧ Essential for tight SLAs (e.g. log writes for a database)
- **Writes must be idempotent and reorderable**
  - ✧ May land many times or in funky order on different replicas

Some Tricks Up Our Sleeve

- **Two Outta Three Ain't Bad**
  - ✧ Launch writes to three replicas
  - ✧ Wait for the first two durable responses
  - ✧ Have some mechanism to actively move from two to three replicas
  - ✧ Two replicas is durable enough if you keep trying to get a third



- **Love the One You're With**
  - ✧ Write to 3 replicas
  - ✧ If no response from 1 or more, find others
  - ✧ A large pool of acceptable places to write
  - ✧ Just don't wait very long to try elsewhere

Bound the SLA by Finding <u>SOMEPLACE</u> to Write 2 Going on 3 Replicas

Identity Empowers Confusion

- **Writes may arrive at the "wrong" place**
  - ✧ Durable at some replica not in the original plan
  - ✧ Must eventually shoo them home to the "right" place
  - ✧ The "right" place is a fuzzy concept
- **Writes may arrive in the wrong order**
  - ✧ Issue log writes to buffers 1, 2, 3, 4, 5, 6
  - ✧ May arrive at a replica as 4, 6, 2, 3, 5, 1
  - ✧ May arrive in different orders at different replicas
- **Intended order must be assigned by the DBMS**
  - ✧ The identity of the buffer must be intrinsic to its identity
  - ✧ Must be *reorderable* by each separate replica to intended order

Assigning the Order at the DBMS or Client Allows Durable Writes at Any Replica While Preserving Order

Tolerance of *Where* You Write Tightens the SLA for *When* You're Durable!