Slicer: Auto-Sharding for Datacenter Applications


Google
Local Memory Considered Helpful

• Server machines have a lot of memory
  • Applications should take advantage of it, e.g., caching

• Datacenter applications often don’t cache data
  • Too hard to implement

• Slicer makes it easy to build services that use local memory
Talk Outline

• Why stateful servers are difficult
• Slicer model and architecture
• Evaluation
Building a DNS Service

End-user devices

DNS Service

Virtual Machines

Cloud Platform

DNS service needs to be scalable and fast!
Full State Replicated on Every Server

- Any server can handle any request
  - Easy adaptation to failures, capacity changes, load skews
  - Hard to scale or handle mutations
Stateless: Interchangeable Servers + Database

- Any server can handle a request
- Cannot query DB for every DNS request
  - High latency
  - Network hop and marshaling costs
Stateful: Static Sharding

- Simple mapping from keys to servers via static function
- Failure adaptation: Black-hole traffic for crashed server
- Capacity adaptation: Could result in significant key churn

Frontends: Hash(key) mod 4

DNS servers: Yellow, Green, Red, Blue
Stateful: Consistent Hashing

*Implement server presence detection*

-_addresses capacity and failure adaptation, key churn
- 🙄 Stochastic load balancing is inadequate
- 🙄 Distributed decisions harm affinity
Stateful: Central Controller

Central server: presence detection, load monitoring, consistent view

❌ Fan-out assignments to large number of clients and servers

✝ Internals of a sharded distributed storage system!

Should we use stateless servers? 😞
Slicer: Refactored System for Sharded Apps

• Provides auto-sharding without tying to storage
• Separate assignment generation “control plane” from request forwarding “data plane”
  • Via a small interface
  • In a scalable, consistent, fault-tolerant manner
• Reshards for capacity and failure adaptation, load balancing
• Evaluated Slicer in production deployment
Benefits of Sharding/Affinity

• Any type of serving from memory / caching
  • E.g., Cloud DNS

• Even stateless services use stateful components
  • E.g. External caches such as Memcache

• Affinity helps aggregating writes to storage
  • E.g., Thialfi [SOSP ’11] batches notification messages to storage
Slicer Sharding Model

Hash keys into 63-bit space
Assign ranges ("slices") of space to servers
Split/Merge/Migrate slices for load balancing
“Asymmetric replication”: more copies for hot slices
Slicer Architecture: Goals

- High-quality sharding and consistency of a centralized system
- Low latency and high availability of local decisions
Slicer Overview

Frontends

Application servers

Slicelet

Distributed data plane

Centralized control plane

Hash(key)

Clerk

Slicer Service
Slicer Architecture

Frontends

Application servers

Existing Google Infrastructure

Clerk

Slicelet

Backup Distributor

Assigner

Distributor

Capacity Monitoring

Health Monitoring

Load Monitoring

Lease Manager
Tolerating Failures

Two types of failures:
• Localized failures: machine failures or datacenter offline
• Correlated failures: whole service such as Assigner or Distributor being down due to, e.g.,
  • Bad configuration push
  • Software bug
  • Bug in underlying dependencies
Tolerating Localized and Correlated Failures

Frontends
Application servers

Backup Distributor datacenters
Smaller/Simpler Components
More Complex Components
Assigner datacenters

Distributor datacenters
Slicer Features and Evaluation

• Load balancing algorithm
• Assignments with strong consistency guarantees
• Production Measurements
  • Detailed scale, load balancing, availability, assignment latencies
  • Comparison with consistent hashing
• Experiments
  • Comparing load balancing strategies
  • Load reaction time
  • Assigner recovery time
Evaluation: Slicer Usage

- Slicer load balances a few million RPS for several Google services
- 99.98% of clients requests had a valid assignment
  - < 0.01% of these requests directed to the wrong server
Evaluation: Load Balancing Effectiveness

Slicer allows tighter capacity allocation by reducing skew
Summary: Slicer makes Stateful Services Practical

- Reshards in the presence of capacity changes, failures, load skews
- Scalable and fault-tolerant architecture
  - Separates assignment generation “control plane” from request forwarding “data plane”
- Evaluated Slicer in production deployment