Service Oriented Architecture

Tom Anderson

Outline

Last time:

Applied Paxos: Raft and MetaSync

Today:

SOA: Architecture of complex web services

Yegge

Thesis: Google (and others) should use SOA as an engineering discipline.

Amazon SOA

- All teams must expose their data and functionality through service interfaces
- Teams must communicate with each other through these interfaces
- No other form of IPC allowed: only via service interface calls over the network
- All service interfaces must be externalizable

  Deisgn to be exposed to developers in the outside world

Terminology

- Service Level Agreement (SLA)
  - Guarantee provided to clients wrt service response time/availability
  - Also: a guarantee by clients as to how they will use the service
- Availability guarantee: use Paxos!
- How do you provide a performance guarantee?
  - Isn't it affected by workload/# of users/types of RPCs?

How Did Amazon Apply SOA?

- Decompose website into thousands of primitive services
- Every team runs their service as a standalone product -- including operations
- Every service provides its clients an SLA guarantee

What Did Google Do?

- Architect website into hundreds of primitive services
- Central planning for capacity utilization
- Central operations
- Expect service clients to be well-behaved

Why SOA?

- Separate interfaces for external developers => feeble API's
- Need first class API's to attract developers
- Need developers to add value: no company is perfect at building apps
- Better for system robustness

Why Platforms?

- Network Effect
  - Value of system grows faster than linearly with number of users
  - Eg: Facebook, Amazon, Google, iPhone,
- Zipf Distribution
  - Popularity as a function of popularity rank
  - Most of the volume in the tail

Lessons

What lessons do you think Amazon learned from adopting SOA?

Meta Lessons

Platform => accessibility

Design for SOA from scratch

- Not as an add on after product release

SOA Lessons

1. Pager escalation
2. Need an automated service registry
3. Every user of an interface is a potential source of DoS/SLA violation
4. Only way to tell if a service is up/functional is to use it
   - testing and fault monitoring are identical
5. Cross service debugging

SOA and SLA's

SLA's must be negotiated.

- If a service has a resource limit, then customers need to know it and work around it.
- If all interfaces are public, need an API for negotiating SLA's

Caches and SLAs

With a cache, system can run 10-100x faster than without.

If cache fails, what happens to the SLA?

Primary/backup for caches?

More Lessons

Users care about getting an answer quickly.

- Make sure the answer delivered quickly was the right answer!

Service evolution is a necessity

- How does a service registry work for evolution?
- How does RPC work with evolution?

Implications

What implications does the Yegge argument have for university CS research?

Memcache

- Application front end
  - Stateless, rapidly changing program logic
- Memcache
  - Middle layer to cache results from storage servers
- Storage backend
  - Stateful
  - Careful engineering to provide safety and performance
  - Easily overwhelmed by flash crowds

Shards

- How do we consistently assign work to a set of nodes, some of which may fail?
  - Consistently => assignment stays (mostly) the same after a failure
- Hashing not enough
  - change # of hash buckets => change every assignment

Consistent Hashing