Service Design Best Practices

James Hamilton
2009/2/26 Principals of Amazon
Amazon Web Services
e: James@amazon.com
w: mvdirona.com/jrh/work
b: perspectives.mvdirona.com
Agenda

• Overview
• Recovery-Oriented Computing
• Overall Application Design
• Operational Issues
• Summary

Contributors: Search, Mail, Exchange Hosted Services, Live Collaboration Server, Contacts & Storage, Spaces, Xbox Live, Rackable Systems, Messenger, WinLive Operations, & MS.com Ops
Motivation

• System-to-admin ratio indicator of admin costs
  – Inefficient properties: <10:1
  – Enterprise: 150:1
  – Best services: over 2,000:1

• 80% of ops issues from design and development
  – Poorly written applications are difficult to automate

• Focus on reducing ops costs during design & development
What does operations do?

- 51% is deployment & incident management (known resolution)
- **Teams**: Messenger, Contacts and Storage, OSSG & business unit IT services

Source: Deepak Patil, Global Foundation Services (8/14/2006)
ROC design pattern

- Recover-oriented computing (ROC)
  - Assume software & hardware will fail frequently & unpredictably
- Heavily instrument applications to detect failures

Bohr bug: Repeatable functional software issue (functional bugs); should be rare in production

Heisenbug: Software issue that only occurs in unusual cross-request timing issues or the pattern of long sequences of independent operations; some found only in production

- Machine out of rotation and power down
- Set LCD/LED to "needs service"
Overall application design

- Single-box deployment
- Keep testing after production deployed
- Zero trust of underlying components
- Pod or cluster independence
- Implement & test ops tools and utilities
- Partition & version everything
Design for auto-mgmt & provisioning

- Support for geo-distribution
- Auto-provisioning & auto-installation mandatory
- Manage "service role" rather than servers
- Multi-system failures are common
  - Limit automation range of action
- Never rely on local, non-replicated persistent state
- Don't worry about clean shutdown
  - Often won't get it & need this path tested
- Explicitly install everything and then verify
- Force fail all services and components regularly
MTTF/MTDL

• Mean time to failure/Mean time to data loss
  – Precise models to many decimal places
  – Models typically ignore S/W failure & human error
  – Assume failure independence

• Unknown unknowns make MTTF/MTDL optimistic

• Threat model approach to data protection
  – List all failures or sequence that could lead to data loss
  – Document and implement mitigation for each
  – Or document & implement that risk was accepted & why
Release cycle & testing

• Ship frequently:
  – Small releases ship more smoothly
  – Long stabilization periods not required if shipping often

• Use production data to find problems (traffic capture)
  – Release criteria includes quality and throughput data

• Track all recovered errors to protect against automation-supported service entropy

• Test all error paths in integration & in production

• Test in production via incremental deployment
  – Never deploy without tested roll-back
  – Continue testing after release
Design for incremental release

• Incrementally release with schema changes?
  – Old code must run against new schema, or
  – Two-phase process (avoid if possible)

• Incrementally release with user experience (UX) changes?
  – Separate UX from infrastructure
  – Ensure old UX works with new infrastructure
  – Deploy infrastructure incrementally
  – On success, bring a small beta population onto new UX
  – On continued success, announce and set roll-out date

• Client-side code?
  – Ensure old & new clients both run with new infrastructure
Canary in the data center

• All systems produce non-linear latencies and/or failures beyond a certain load level
  – The load limit
• The load limit is release dependent
  – It changes as the application changes
• Canary in the data center
  – Route increased load to one server in the fleet
  – When it starts showing non-linear delay or failure, immediately reduce load on it or take out of LB rotation
  – Result: limit is know before full fleet finds it (avoid or fix)
Graceful degradation & admission control

• No amount of capacity head room is sufficient
• Graceful degradation prior to admission control
  – First shed non-critical workload
  – Then degraded operations mode
  – Finally admission control
• Related concept: Metered rate-of-service admission
  – Allow a single or small number of users in when restarting a service after failure
Auditing, monitoring, & alerting

• All config changes need to be tracked via audit log
• Alerting goals:
  – No customer events without an alert (detect problems)
  – Alert to event ratio nearing 1 (don’t false alarm)
• Alerting is an art ... need to tune alerting frequently
  – Can’t embed in code (too hard to change)
  – Code produces events, events tracked centrally, alerts produced via queries over event DB
• Fine-grained monitoring of all inter-service requests
• Testing in production requires very reliable monitoring
  – Combination of detection & capability to roll-back allows nimbleness
Dependency management

- Expect latency & failures in dependent services
  - Run on cached data or offer degraded services
  - Test failure & latency frequently in production
- Don’t depend upon features not yet shipped
  - It takes time to work out reliability & scaling issues
- Select dependent components & services thoughtfully
  - On-server components need consistent quality goals
  - Dependent services should be large (“worth” sharing)
- Isolate services & decouple components
  - Contain faults within services
  - Assume different upgrade rates
Customer & press communications plan

- Systems fail & you will experience latency
- Communicate through multiple channels
  - Opt-in RSS, web, IM, email, etc.
  - If app has client, report details at client
- Set ETA expectations & inform
- Some events will bring press attention
- There is a natural tendency to hide systems issues
- Prepare for serious scenarios in advance
  - Data loss, data corruption, security breach, privacy violation
- Prepare communications skeleton plan in advance
  - Who gets called, communicates with the press, & how data is gathered
  - Silence interpreted as hiding something or lack of control
Take Aways

• Threat model approach rather than MTTF/MTTDL
  – Unknown unknowns & lack of failure independence

• Reduce application & administrative errors:
  – Easy 1-box testing of entire service
  – Automate (and test) operational actions & recoveries

• Expect application errors remain:
  – Incremental deployment with rollback
  – Deep monitoring, rapid fault detection, & enforced fault containment boundaries
  – Constant functional tests running in production
  – Canary in DC to find load limits
More Information

• Designing & Deploying Internet-Scale Services paper:

• Autopilot: Automatic Data Center Operation

• Recovery-Oriented Computing
  – http://roc.cs.berkeley.edu/
  – http://www.cs.berkeley.edu/~pattrsn/talks/HPCAkeynote.ppt
  – http://www.sciam.com/article.cfm?articleID=000DAA41-3B4E-1EB7-BDC0809EC588EEDF

• These slides:

• Email:
  – James@amazon.com

• External Blog:
  – http://perspectives.mvdirona.com