AutoPilot

Themes we will see

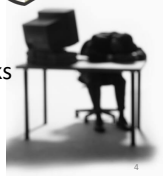
- Single master if possible
- Strong consistency when important
- Legacy code less important
- Simplicity if possible
  - Balance generality with simplicity

AutoPilot

- Goal: deploy, provision, repair data center applications
  - automate as much sysadmin tasks as possible
- Design principles
  - fault tolerant like everything else, but not byzantine failures
  - simple and good enough when possible
    - e.g. repairs only in a few categories
    - text file configuration + auditing of changes
    - **QUESTION: EmuLab seems to offer more. Why?**
  - Correct at all times, or understandably and documented incorrect
    - state exceptions, handle pathological cases
  - Crash-only
    - no explicit shutdown/cleanup
  - Replication for availability only
    - Workload is fairly small

Autopilot goal

- Handle automatically routine tasks
- Without operator intervention
- No support for legacy apps

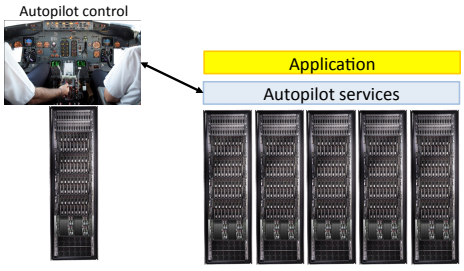


4

AutoPilot services

- Device manager:
  - stores goal state about state system should be in
  - strongly consistent using Paxos
  - doesn't do anything but store state
    - Goal state, ground truth
- Satellite services
  - based on state in DM, take actions to bring system to correct state
  - Poll for changes, but can be told to poll
    - avoids lost pushes of data
    - Told to pull for low latency

Autopiloted System



6

Recovery-Oriented Computing

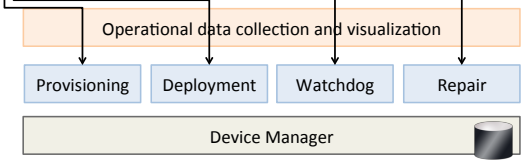
- Everything will eventually fail
- Design for failure
- Crash-only software design
- http://roc.cs.berkeley.edu

Brown, A. and D. A. Patterson. Embracing Failure: A Case for Recovery-Oriented Computing (ROC). *High Performance Transaction Processing Symposium*, October 2001.

7

Autopilot Architecture

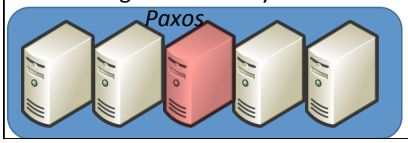
- Discover new machines; netboot; self-test
- Install application binaries and configurations
- Monitor application health
- Fix broken machines



8

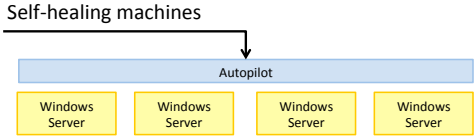
Centralized Replicated Control

- Keep essential control state centralized
- Replicate the state for reliability
- Use the Paxos consensus protocol
- Device manager uses it for ground truth – goal state of system



9

Autopilot abstraction



10

Consistency Model

Strong consistency

- Expensive to provide
- Hard to build right
- Easy to understand
- Easy to program against
- Simple application design

Weak consistency

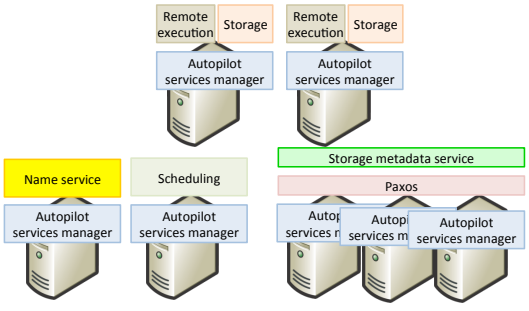
- Increases availability
- Many different models
- Easy to misuse
- Very hard to understand
- Conflict management in application

11

Device Manager

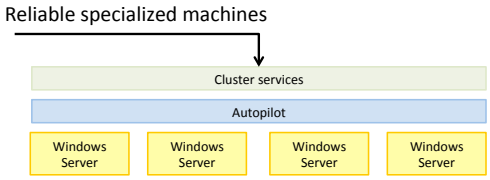
- Strongly consistent (paxos), replicated for reliability
- Stores goal state for system
  - Set of physical machines
  - Configuration each should have
- Other services pull information from it
- QUESTION: LiveSearch decided this wasn't reliable enough. Why?
  - Answer: didn't trust, didn't want to lose control to another group, or was it really unreliable?
  - Shows that apps can build their own better services on top of AutoPilot…

Cluster Machines



13

Cluster services abstraction



14

Cluster Services

- Name service: discover cluster machines
- Scheduling: allocate cluster machines
- Storage metadata: distributed file location
- Storage: distributed file contents
- Remote execution: spawn new computations

15

Low-level node services

- filesync: copy files to make sure correct files prsent
- application manager: makes sure correct applications running

High-level clusterservices

- Provisioning service:
  - Configure new machines according to policy
  - Determines what OS image,
    - install, boot, test
  - New machine asks DM what applications to run

High-level services

- Application Deployment
  - apps specify a set of *machine types* – different configurations of nodes in service
    - front-end web server
    - crawler
  - apps specify *manifest* – lists config files + app binaries needed for machine type
- Deployment service contains config files + binaries (populated by build process)
  - machines ask DM for their configuration, contact deployment service for needed binaries

Application upgrade

- Rolling upgrade built into autopilot
  - added to manifest for machine type
  - each machine in type will download new code on next poll
- DM instructs groups of machine to upgrade (to avoid whole-service downtime)
  - e.g. 1/10th of each type at a time
- Put machine on probably during upgrade in case it fails, can roll back upgrade

Failure detection

- Watchdog service:
  - Checks an attribute on a node – reports "ok", "warning", or "error"
  - Watchdog service calls node correct if all attributes OK or Warning (warning is unexpected but not fatal) to generate extra logging but not alert operators
  - Watchdog sensors can be standard (e.g. bios for memory/ disk corruption, OS version check) or app-specific
- **QUESTION: Why have apps generate their own watchdogs?**
- Note: check lots of signals, as just one could still be working while things fail

Failure Detection Latency

- AutoPilot doesn't detect things immediately
  - Down/sluggish machine can affect application latency
- SO: what to do?
  - Apps can have very short timeouts and retry quickly
  - Apps can report such failures to AutoPilot
- AutoPilot is generic, its recovery techniques are not suitable for stuttering latency problems
  - E.g. would be overkill, could hurt overall reliability

Failure/recovery

- Recovery: do simple things an admin would do automatically before taking offline
  - E.g. restart service, reboot, reinstall, replace
- Failed nodes given a repair treatment based on symptoms:
  - DoNothing, Reboot, ReImage, Replace
    - techs replace computers periodically (days/weeks)
  - Based on history (previously healthy computers go to DoNothing)
- Repaired nodes marked as on "probation"
  - expected to have a few early failures (ignored) but then become healthy
  - if correct for a while, moved to Healthy
- All machine affected by new code marked as "probation" or for expected failures during startup

Other recovery options

- Hot standby to replace a failed machine
  - But the machine is idle most of the time and not contributing
  - But apps can handle failures anyway, so why not make app deal with it until full recovery?

Monitoring service

- Collects logs/performance counters from applications in common format
  - provide central view of app
  - Real-time data but in SQL DB
- *Cockpit* visualization tool reads data, shows current status
- *Alert service* sends alert emails based on triggers/queries in cockpit DB

What AutoPilot doesn't do

- Load balancing/migration
  - This is up to the app
  - If the app needs more resources – what then?
    - Tell AutoPilot to provision more machines
- Address all issues in the data center
  - Network configuration
  - Power management/consolidation

AutoPilot Summary

- Provides common tools to:
  - install a new machine
  - provision apps onto it
  - detect failures
  - repair failures
  - record logs
  - monitor app behavior
- BUT:
  - not for legacy code (must be packaged for AutoPilot)