Cluster management at Google

john wilkes

Oct/Nov 2011
johnwilkes@google.com

Cluster management: what is it?

• A fleet of machines live in datacenters placed in different regions & countries
Cluster management: what is it?

• A datacenter contains 1 or more clusters

machine
rack: 40-80 machines
+ Ethernet switch

Cluster management: what is it?

• Clusters are managed as 1 or more cells
Cluster management: jobs

• Users submit *jobs* to a cell
• A job has up to a few thousand *tasks*

• Jobs & tasks have *requirements* (sizes, constraints)
Cluster management: jobs

Job types:
- *services*; e.g., user-facing (latency-sensitive)
- *batch*; e.g., MapReduce (throughput sensitive)
  - important or not
  - one-off or periodic
  - standalone or coprocessor (e.g., BigTable)
  - inter-job dependencies

Cluster management: other stuff

- Machine lifecycles
  - provisioning; testing; repairs; upgrades
- Software lifecycle
  - e.g., OS install + upgrades + downgrades
- Cluster maintenance
  - Planned Change Requests (PCRs)
  - scheduling; draining; restoring
- Monitoring (stats, events, usage, ...)

Google
Cluster management: storage

- **Persistent storage** is distributed (GFS, Colossus/CFS, BigTable)
  - spread across machines in cluster

- **Local storage** is volatile
  - cache for data being served
  - logging: (1) write locally; (2) async copy to persistent store

Cluster management: scale

- Big: the storage system **holds a Petabyte**

- Google: the storage system pages you when there’s **only a few Petabytes of space left**
  -- Luiz Barroso
Cluster management: faults 😞

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate of uncorrectable DRAM errors/machine/year</td>
<td>&gt; 1%</td>
</tr>
<tr>
<td>Annual failure rate of disk drives</td>
<td>2-10%</td>
</tr>
<tr>
<td>Crashes/machine/year</td>
<td>~2</td>
</tr>
<tr>
<td>OS upgrades/machine/year</td>
<td>2-6</td>
</tr>
<tr>
<td>Power utility events per year</td>
<td>&gt; 1</td>
</tr>
</tbody>
</table>

-- Luiz Barroso, Google

Cluster management: faults 😞

A 2000-machine service sees
>10 machine crashes per day

Main causes of service outages:
networking, power, “oops”

Rarer events:
wild dogs, sharks, dead horses,
copper thieves, drunken hunters, ...

-- Luiz Barroso/Jeff Dean, Google
Cluster management: goals

1. Run everything :-)

2. High utilization

3. Predictable, understandable behavior

Cluster management: goals

4. Fine control for the big guys (resource efficiency)

5. Ease of use for others (innovation efficiency)

6. Keep going (failure tolerance)

7. At large scale

8. With few people
Cluster management: goals

• Q: why not energy/power?
• A: we do care about energy/power proportionality

But ...
– best way to save energy is to write good software
– Google PUE was 1.13 in Q1’2011 (3-month weighted average)
– don’t buy idle machines!
Cluster management: pre-Omega

- Current system was built 2003-4
- Works pretty well 😊

- Beginning to run out of steam ...
  - scale (largest clusters)
  - inflexibility (ease of adding new features)
  - internal complexity (ease of adding new people)

Omega

- The “second system” ...
- Main user goals: predictability & ease of use
- Main team goal: flexibility

- Caveat: Omega is currently being prototyped
  - not in production!
  - many things will change!
  - it may never be deployed!
Omega the general approach

- Dedicated "verticals" for different needs
  - services, batch, machine management
- Central shared **state**
  - calendar of allocation decisions
  - minimal necessary data
  - no policies; just enforces invariants
- **Failures** are a first-class property
  - the resource model
Omega issues: *intentions*

Avoid detailed specifications of *how*
  – **not**: “place 40 tasks on that rack, 20 on this one”
  – **not**: “I need 4.6 CPUs of processor type \( p1 \)"

Use *goals* (SLOs/SLAs)
  – “\( \leq 2 \) tasks down at a time (99.9\%ile)”
  – “\(<25\text{ms} \) request latency (95\%ile)”

Omega issues: *failure tolerance*

- Goal: limit the number of concurrent outages
  - topology-aware scheduling
  - surety: likelihood of resources being available

- **Fault detection**
  - real fault, or just lost touch?
  - time to detect vs. false positives
  - correlated failures
Omega issues: *master scalability*

- Calendaring
  - super-efficient “does it fit?” checks
  - scheduling horizon? edge effects?

### Multiple scheduler verticals

- livelock / mutual interference
- optimistic concurrency?
- what needs to be communicated?
Omega issues: *predictable behavior*

In the machine
- normalized performance (CPU, memory/NUMA)
- **performance isolation**
  - low latency and high throughput
  - storage?
  - caches?
- security isolation (PII, SOX)

---

Omega issues: *predictable behavior*

Under load
Omega issues: *predictable behavior*

Across the cell

– “why was my job not scheduled?”
– “where should I provision a new service?”

**Approaches**

– admission control
– explanations that make sense
– avoid the problem

---

Omega issues: *objectives*

SLOs and SLAs

– what can/should be offered?
– how can they be controlled for at runtime?
– handling evolution

“Fair” objective functions

– is fairness useful/important?
  (reality is more complicated)
Omega issues: *cell management*

It’s 3am and your pager goes off

- are we in trouble?
- are we about to get into trouble?

- what should we do about it?

Omega issues: *ease of use*

How much can we simplify things?

- “here’s a binary ... run it”
- predictions based on prior history may help
- parameters?
Issues: configuration

- Make an app work right for one instance: *simple*
  - Google Docs uses ~50 systems and services
- Make an app work right in production: *priceless*
  - run it in half a dozen cells
  - release a new version
  - fix it on the fly in an emergency
  - move one copy to another cell

Summary

- Large-scale systems have some fun problems
  - driven by scale + failures
  - enormously fruitful ground for CS research
- Configuration may be *the* next big challenge