Mirador: An Active Control Plane for Datacenter Storage

Jake Wires and Andrew Warfield
Coho Data
### Trends

<table>
<thead>
<tr>
<th>SSD</th>
<th>Cap / 1u</th>
<th>Xput per data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 TB</td>
<td>64TB</td>
<td>312MB/s/TB</td>
</tr>
<tr>
<td>8 TB</td>
<td>256TB</td>
<td>78 MB/s/TB</td>
</tr>
<tr>
<td>32 TB</td>
<td>1PB</td>
<td>20 MB/s/TB</td>
</tr>
<tr>
<td>128 TB</td>
<td>4PB</td>
<td>5 MB/s/TB</td>
</tr>
</tbody>
</table>

NVMe device: x4 PCIe  
Broadwell CPU: 40 PCIe lanes  
TOR cross-rack links typically oversubscribed at 3 or 4:1

Placement is critical
Progress
Mirador actively migrates data and network flows to optimize for efficiency, performance, and scale.
Challenges

- Software defined networking provides a nice model, but: persistence presents additional challenges
  - More constraints to satisfy
  - More dimensions to optimize
  - More expensive to reconfigure
Placement

- Replicate files across failure domains
- and minimize cross-rack traffic
- and co-locate related files
- and stripe files across devices
- and respect device capacity limits
- and respect device performance limits
- and arrange for parallelizable device rebuilds
- and distribute load evenly across nodes
- and ensure exclusive caching
- and move cold data to cheaper media
- and support customer X’s special requirements for multimedia files
- and ...
Pipeline

Centralized three-stage pipeline continuously optimizes placement
Monitor collects resource utilization levels and longitudinal workload profiles
Pipeline

**Planning engine** optimizes configuration along multiple dimensions
Scheduler coordinates migration of data and network flows
Policy

Approach policy as a search problem:

- **Rules** (aka *objective functions*) codify intent
- **Costs** prioritize rules
- **Solvers** optimize cost

```python
@rule(model.Device)
def load_balanced(fs, device, domain):
    cost, penalty = 0, DEVICE_BALANCED_COST
    # compute load of current device
    # for the current sample interval
    load = device.load()
    # compute load of least-loaded device
    minload = fs.mindevice().load()
    if load - minload > LOAD_SPREAD:
        # if the difference is too large,
        # the current device is overloaded
        cost = penalty
    return cost
```

*Rules quantify violations*
Optimization

● Given an existing configuration and a set of policy rules:
  ○ Minimize cost of violations
  ○ Minimize churn of reconfiguration

● Pluggable solver interface
  ○ Branch and bound
  ○ Greedy

Solvers search for solutions
Our Production Policy and Constraint Solver

7 rules governing:

- Network and storage performance and capacity balancing
- Replication across tiers and failure domains
- Device parallelism for striped files

Two-pass greedy algorithm

- Addresses highest-cost violations first
- Uses hints provided by rules to prune search space

rules.py: 219 sloc       solver.py: 128 sloc       glue.py: 1330 sloc
A Monolithic Alternative

gine.py: 2,289 sloc
Assigning Costs

- Rules do not eliminate the tension between conflicting goals
- They do provide convenient knobs for tuning the overarching policy

A typical policy test case

test/*.yaml: 11,954 sloc
Assigning Costs

- Rules do **not** eliminate the tension between conflicting goals
- They **do** provide convenient knobs for tuning the overarching policy

This complexity exists independently from policy language
Finding Solutions

<table>
<thead>
<tr>
<th>Objects</th>
<th>Devices</th>
<th>Reconfigurations</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>10</td>
<td>6.40 ± 2.72</td>
<td>0.40 ± 0.06</td>
</tr>
<tr>
<td>1K</td>
<td>100</td>
<td>145.50 ± 33.23</td>
<td>0.83 ± 0.08</td>
</tr>
<tr>
<td>1K</td>
<td>1000</td>
<td>220.00 ± 12.53</td>
<td>10.11 ± 0.49</td>
</tr>
<tr>
<td>10K</td>
<td>10</td>
<td>0.00 ± 0.00</td>
<td>1.61 ± 0.01</td>
</tr>
<tr>
<td>10K</td>
<td>100</td>
<td>55.70 ± 5.46</td>
<td>5.54 ± 0.37</td>
</tr>
<tr>
<td>10K</td>
<td>1000</td>
<td>1475.00 ± 69.70</td>
<td>16.71 ± 0.88</td>
</tr>
<tr>
<td>100K</td>
<td>10</td>
<td>0.00 ± 0.00</td>
<td>17.10 ± 0.37</td>
</tr>
<tr>
<td>100K</td>
<td>100</td>
<td>9.30 ± 4.62</td>
<td>22.37 ± 5.38</td>
</tr>
<tr>
<td>100K</td>
<td>1000</td>
<td>573.80 ± 22.44</td>
<td>77.21 ± 2.87</td>
</tr>
</tbody>
</table>

$O(N \times \log N \times \log M)$ for $N$ objects and $M$ devices
Workload-Aware Placement

- Policy rules informed by detailed workload profiles present new opportunities:
  - Working set size bin-packing
  - Noisy neighbor isolation
  - Workload co-scheduling
- See paper for more details!
Conclusion

● Separate control path for optimizing placement
● Active placement of data and network flows
● High dimensionality makes placement a hard problem
● Configuration as search