Machine Learning in the Self-* Architecture

Eno Thereska
Mike Mesnier, Greg Ganger
and Self-* Team

PARALLEL DATA LABORATORY
Carnegie Mellon University

Self-*: Towards easier to manage storage

- Why: storage management is far too work-intensive
  - we’re working for the computers (1 admin / 1-2 TB)

- How: automate, automate, automate
  - people specify goals, system figures out how
  - people complain, system improves itself

- How: Integrating management functions into design
  - rather than designing tools after-the-fact

Self-*: Administration challenges/issues

- Data protection
  - mistake recovery, device failure recovery, archiving

- Capacity planning
  - initial provisioning and expanding over time

- Tuning and load balancing
  - dataset placement, device parameters, etc.

- Problem diagnosis and healing
  - identifying source and adjusting appropriately

Self-* Storage Architecture

I/O request routing

Management hierarchy

Workers

Clients

Administrator

John & Andy know all the details
What is Machine Learning?

- It’s really statistics…
- Two high-level goals:
  - Predicting the future
  - Maximizing future rewards
- …through probabilistic models

Why Machine Learning in Self-*?

- Large state space
  - workers need to explore different layouts
  - supervisors need to explore different configurations
- Design cannot handle all system scenarios
  - workloads change
  - storage bricks have different properties
- Ok, but…
  - What does “classical” systems analysis tell us?

Classical Analysis in Systems

- workload
  - name space locality
  - organize by age
  - disk shuffling
  - place track-aligned
  - new layout

Self-* Worker Desired Design

- workload
  - constraint generation
  - heuristic 1
  - ... heuristic n
  - adaptive combiner
    - performance analyzer
    - layout manager
  - learner
  - final layout

Self-* Worker Design
Classical Analysis in Systems

- Rules of thumb
  - aka ad hoc methods when combined
- Trial and error
- Systems often over-provisioned
- Most methods focus on initial configuration only

Self-* Supervisor Desired Design

- Workload
  - capture defining characteristics

= Goals?

Self-* Other Considerations

- Lots of room for other cool stuff
  - smart caching of objects
  - smart routing
  - smart scheduling
  - etc...

- Bottom line: impossible to design for all possible scenarios. Needs to adapt to new, unaccounted for cases

Outline

- Motivation and Overview
- ABLE
- Preliminary Evaluations
- Conclusions
ABLE: Introduction by Analogy

- Suppose you are the manager of a warehouse
- Must capacity plan and provision resources
- Carriers (e.g., FedEx) are contracted by customers (e.g., Sears, Intel, Starbucks)
  - Customers determine nature of packages
  - Carriers determine nature of delivery
- You quickly learn the behaviors of both
- Careful profiling of your clientele will allow you to classify and generalize their behavior
  - Intel and AMD more similar than Intel and Sears
  - Makes adding new customers easier...

ABLE: Applying the Analogy

- Customers are the applications
  - backup, cc, make, pine
- Carriers are the storage applications
  - File systems, databases
- Objects are items being stored
- We want to profile object behavior
  - Who, what, when and how accessed
- Based only on existing object attributes
  - such as nlinks, file name, mode, mtime, ctime, etc

- Customers are the applications
  - backup, cc, make, pine
- Carriers are the storage applications
  - File systems, databases
- Objects are items being stored
- We want to profile object behavior
  - Who, what, when and how accessed
- Based only on existing object attributes
  - such as nlinks, file name, mode, mtime, ctime, etc

Types of Object Behavior

- Functional lifetime (cache vs disk lifetime)
  - Difference between last written and read
  - Ex: .o files are short-lived, .c long-lived
- Access pattern
  - Sequential, random, strided, partial, full
  - Read/write ratio
- Maximum sizes
  - zero, <4K, >1MB
  - etc...

ABLE is the prototype for a powerful predictor

Simple Example

<table>
<thead>
<tr>
<th>Extension</th>
<th>Functional Lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>object 1 .c</td>
<td>long</td>
</tr>
<tr>
<td>object 2 .h</td>
<td>long</td>
</tr>
<tr>
<td>object 3 .o</td>
<td>short</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>object j .h</td>
<td>long</td>
</tr>
<tr>
<td>object j+1 .c</td>
<td>long! 😊</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>object n .o</td>
<td>short! 😊</td>
</tr>
</tbody>
</table>

Easy to learn the pattern
Apply what we've learned

Not all classifications are this simple…
ABLE in Self-*

- ABLE: predicts maximum object size, object access patterns etc
- Worker: Continuously reorganizes layout
- ABLE: gives hints on object lifetime
- Cache manager: decides on the best caching policy for the object

Anthony Brockwell’s team

Goal: Predict how well a workload will do on a certain black-box worker

Outline

- Motivation and Overview
- ABLE
- Preliminary Evaluations
- Conclusions

Learning Model Requirements

- Computational Scalability
  - training speed and recall speed
- Model Storage Requirements
  - kNN needs to store all training data points
  - DTs only store “compressed stats” in a tree
- Incremental Learning
  - after initial training system is deployed
  - deployed system must still learn
Learning Model Requirements

- Good predictive power
  - accuracies higher than 80% ?
- Low impact of misprediction
  - want to only improve system by using the model hints
  - examples:

```
+-----+-----+-----+-----+
| long| short| long| short|
+-----+-----+-----+-----+
| 500 | 500  | 500 | 0    |
| 0   | 3000 | 3500| 0    |
```

ABLE lifetime predictions confusion matrices

ABLE: Preliminary Evaluations

- Experimental setup:
  - Decision trees as base learner
    - Training speed: $O(AxN\times\log(N))$
    - Recall speed: $O(\text{height(tree)})$
  - Storage requirements: small (~1MB/tree)
  - Incremental learning: yes (ID4 & ID5)
  - Cost matrix reduces impact on misprediction
  - Harvard NFS traces (7 days)
  - P3 with 384MB of RAM

ABLE: Preliminary Evaluations

- Goal: Predict size and cache lifetime based on file attributes

![Graph](https://example.com/graph.png)
### ABLE: Preliminary Evaluations

![Graph showing cache lifetime prediction accuracies](image)

### Future Work

- Analyze various re-training methods
- Combine application needs with learning model
  - can predict really well size in 3 buckets
  - but what if application wants finer grained predictions
- Evaluate ABLE in a real storage system
  - FFS
  - Self-*

### Conclusions

- Machine learning and systems have met
  - systems not mature for classical analysis
- Self-* requires clean automation
  - Worker combines heuristics
  - Supervisor assigns workloads to devices

ABLE is the prototype for a powerful predictor