OIL: the OceanStore Introspection Layer

autonomous adaptation for internet systems

Dennis Geels (geels@cs.berkeley.edu) **John Kubiatowicz (kubi@cs.berkeley.edu)**

MOTIVATION

Internet systems must be adaptive

Operating environment is extremely dynamic
- Nodes fail, user load shifts targets and varies intensity, partitions form in the network, new resources come online, etc.

Availability and performance require continuous tasks
- Data replica management
- Node failure recovery
- Overlay network optimization
- Peer reputation management

Manual optimizations are expensive and not scalable

Each server requires tuning of dozens of parameters per server
− consider millions of machines,interacting across administrative boundaries
System administrators hard to train, expensive to keep, prone to human error

Solution: Introspection

Add monitors and controls to every system component
Allow machines to automatically adjust their own behavior

Our contribution: OIL

OIL is a framework for building and managing introspective capabilities for large, distributed systems
The OIL toolkit contains components for java−based servers

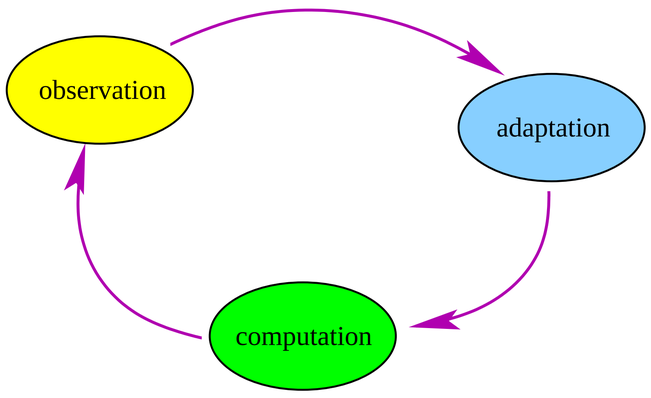
DESIGN GOALS

- General, extensible framework
- Usable interface for human administrators
- Efficient communication
- Deployable and installable into running servers, remotely
- Detection/avoidance of instability in the control feedback cycle
- Minimal memory, CPU resources consumption

INTROSPECTION

Introspection is an architectural paradigm modeled after adaptation in biological systems. It uses some of the growing surplus of storage and processing resources to improve overall system stability and performance.

As shown below, introspection augments normal system computation with observation and adaptation.



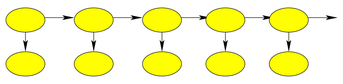
The observation component constructs a model of the environment or of the current system behavior. The adaptation component uses this model to adjust the computation for better performance, availability, efficiency, etc.

MODELS

Introspective components store knowledge and communicate via models. Models represent the state of the system at a level of abstraction higher than the events from which they are built.

Simple models:
- Single−variable average
- Value/range detection
- LRU cache

Complex models:
- Kalman Filter
- Hidden Markov Model
- Probability Density Estimator



One of OILs key strengths is the system−wide use of model deltas, segments, and summaries for efficient storage and communication. Introspective components are thus able to interact at a high level of abstraction and communicate detailed knowledge efficiently.

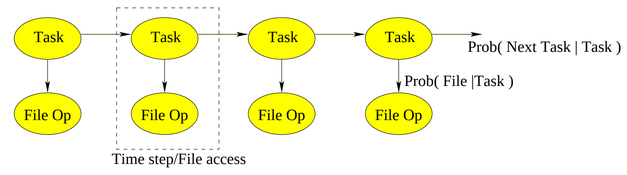
EXAMPLE: REPLICA MANAGEMENT

Problem

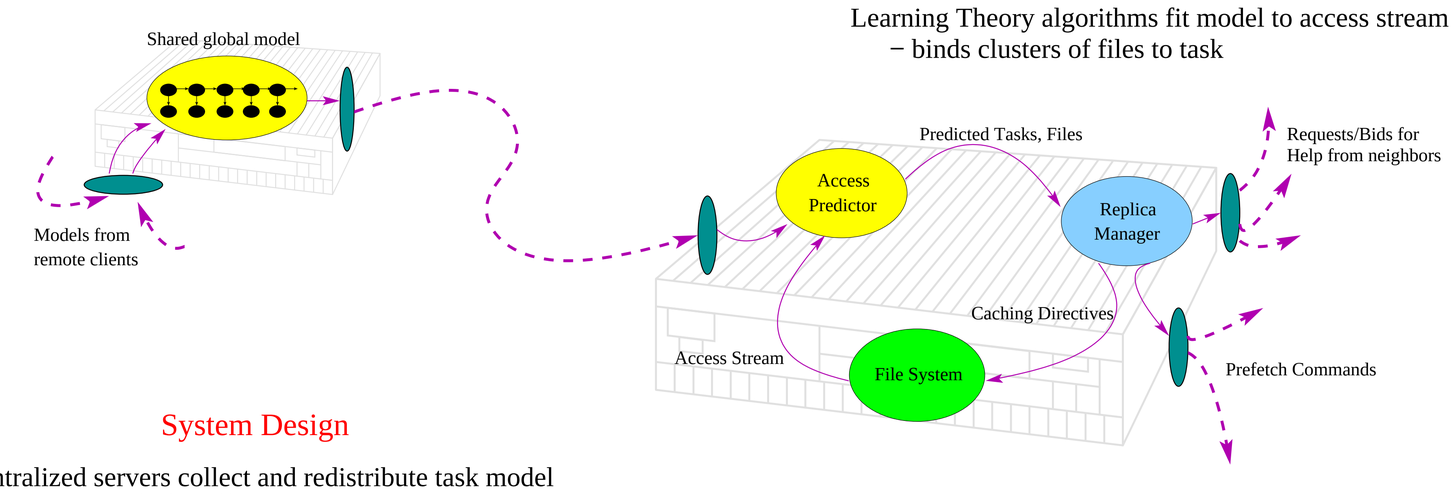
We want millions of users to access trillions of files on billions of servers,
− with the availability and performance of a local storage system.
We need three forms of replica management:
- Hoarding − client machines cache aggressively
- Prefetching − hide latency by fetching files proactively
- Replica coordination − servers cooperate to provide good coverage

Client Activity Model

We model client activity with a Hidden Markov Chain
Hidden states correspond to user tasks



Learning Theory algorithms fit model to access stream
− binds clusters of files to task



System Design

Centralized servers collect and redistribute task model
- aggregate info for efficiency ,anonymity

Each machine contains a Replica Manager which optimizes the local replica set
- hoard entire tasks to minimize network interaction
- prefetch predicted files to hide latency

Replica Managers coordinate shared replica placement
- help neighbors in exchange for help/money
- minimize duplicate work
- maximize coverage for less popular files

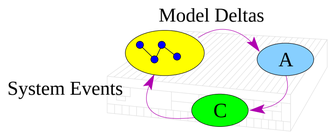
ARCHITECTURE

The OIL framework contains composable tools for introspection
Here we present the initial architecture, grouped roughly by component type

Observation Components

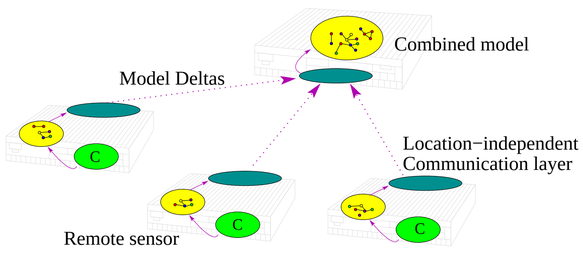
Model Builder

Models are built to provide a higher−level view of the system to the adaptation stage



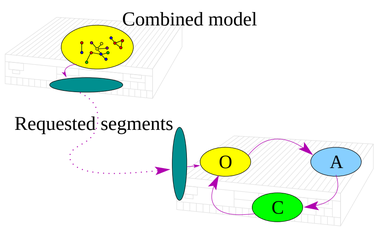
Combining distributed models

Centralized nodes may contain components which incorporate information from many clients into a single model



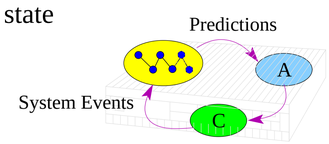
Using external models

Builders, Predictors can swap in segments of these external models



Support for proactive adaptation

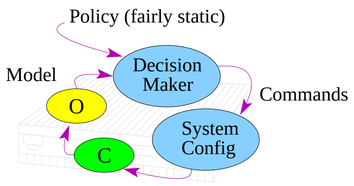
Components may extend the model and forward predictions rather than the current state



Adaptation Components

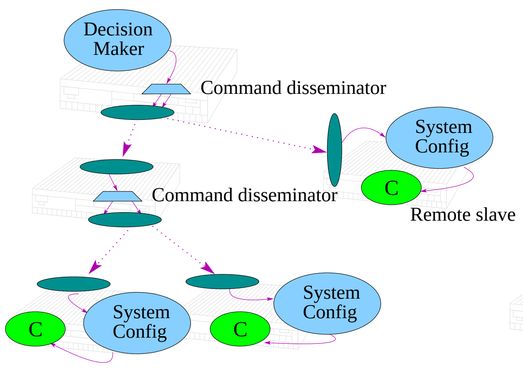
Rule−based Controller

System Adaptation can be organized as a controller, which exports tunable "knobs" and a policy−based rule checker which tunes those knobs based on the current state



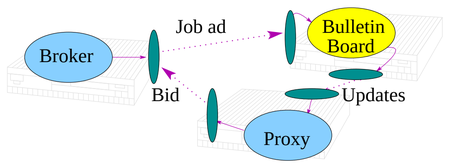
Multicast commands

Decision making components use Multicast components to send commands to multiple remote servers

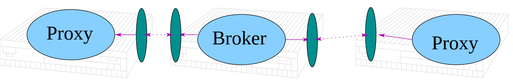


Service discovery and Proxies

Clients use Broker components to find proxy servers
Brokers may advertise work at external "bulletin−board" model



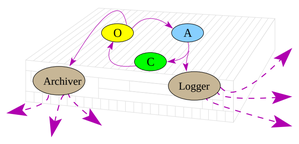
Alternately, Broker directly solicits small set of known proxies



Components for Meta−Introspection

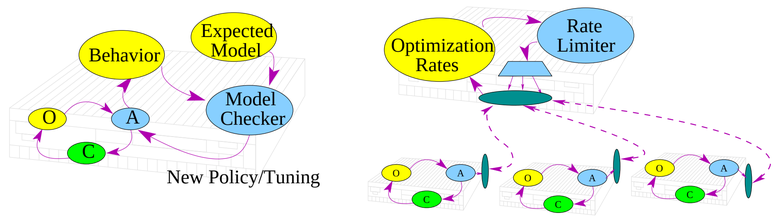
Archival interface

Archival components store models for later use, or log decisions for later audits and diagnostics
- uses OceanStore archive system



Hierarchical Introspection

Introspective cycles for introspective components
- component checking
- stability analysis



Human input and control

Any introspective component can be replaced or supplemented with a human−controlled component