Adaptive Distributed Systems

Challenges and Solutions

Rick Schlichting
Executive Director, Software Systems Research
AT&T Labs-Research

In collaboration with:

- Matti Hiltunen, Kaustubh Joshi (AT&T)
- Gueyoung Jung, Calton Pu (Georgia Tech)

Adaptive Systems

Dynamically changing system behavior.

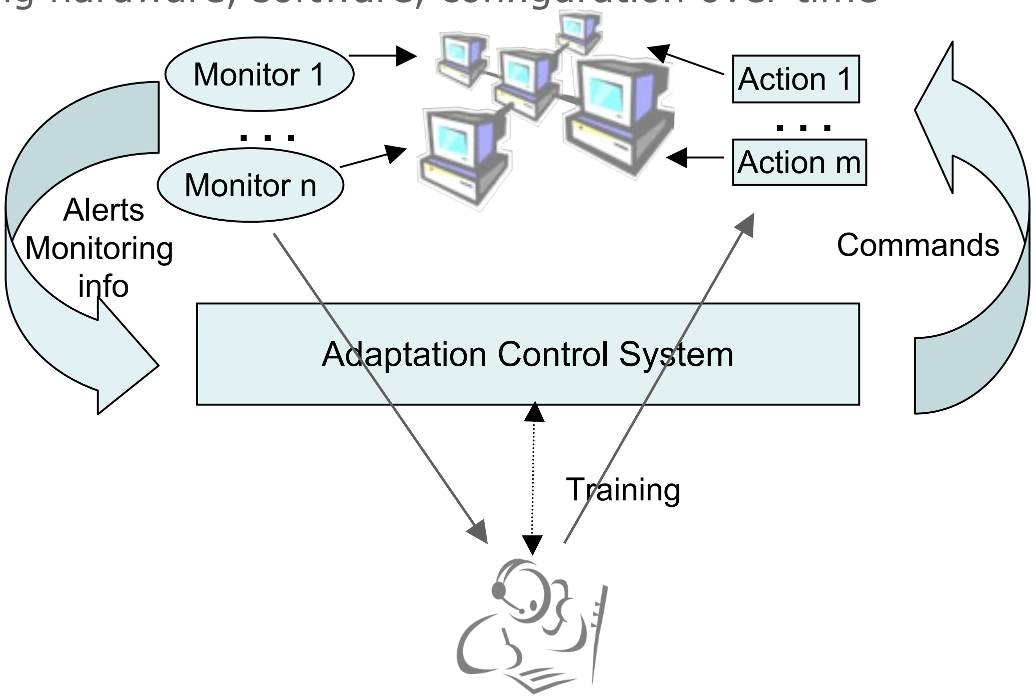
Motivation:

Short term ⇒ react to changes in the environment: May be caused by failures, intrusions, spam/virus/worm attacks, mobility, changes in hardware resources, changes in user requirements, etc.

Long term ⇒ system evolution: updating hardware, software, configuration over time

Examples:

- Networking: Changing video frame rate in response to congestion.
- Mobile systems: Implementing location-specific services.
- Fault tolerance: Reconfiguring software to deal with a host failure.
- Survivability: To impose addition barriers to counteract an intruder.



Challenges

Fundamental issue ⇒ each phase of the feedback control loop is complex in large networked systems.

Monitoring

- Collecting and correlating data across multiple hosts.
- Knowing *what* to monitor and *when*.
- Minimizing intrusiveness of monitoring mechanisms.

Analyze and decide ⇒ policies

- Determining actual system state from monitoring results.
- Developing policies ⇒ *automatic generation*
  - Predicting impact of changes ⇒ *system tomography*
- Expressing policies.
- Implementing policies efficiently.
- Making decisions in a distributed system.
- Avoiding oscillations.

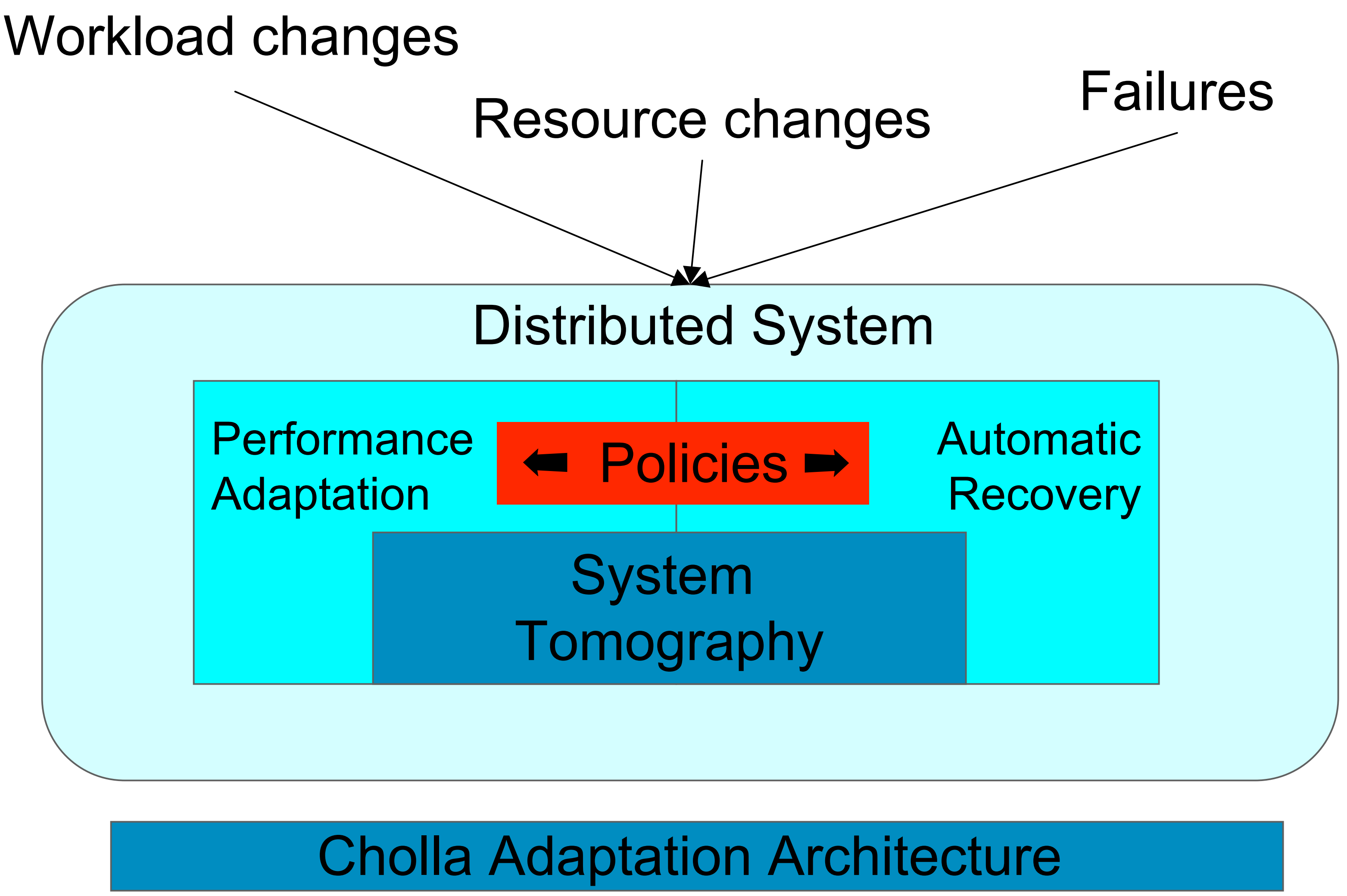
Adapt ⇒ mechanisms

- Changing values, software modules, resource allocations, etc.
- Decoupling control from regular functionality.
- Actually effecting change, especially in software where no source code is available or that cannot be changed directly.
- Maintaining correctness across and during adaptations.
  - Inter-component coordination on a single host
  - Inter-host coordination for distributed services

⇒ *Cholla adaptation architecture*

All must be done in a running system and an environment that continues to change.

at&t


Workload changes
Resource changes
Failures
Distributed System
Performance Adaptation
Policies
Automatic Recovery
System Tomography
Cholla Adaptation Architecture

Generating Adaptation Policies
(G. Jung, C. Pu)

Problem: Deciding how to continuously configure systems to adapt to changing conditions.

Technologies:

- Stochastic models, reinforcement learning, control theory

Typical approach:

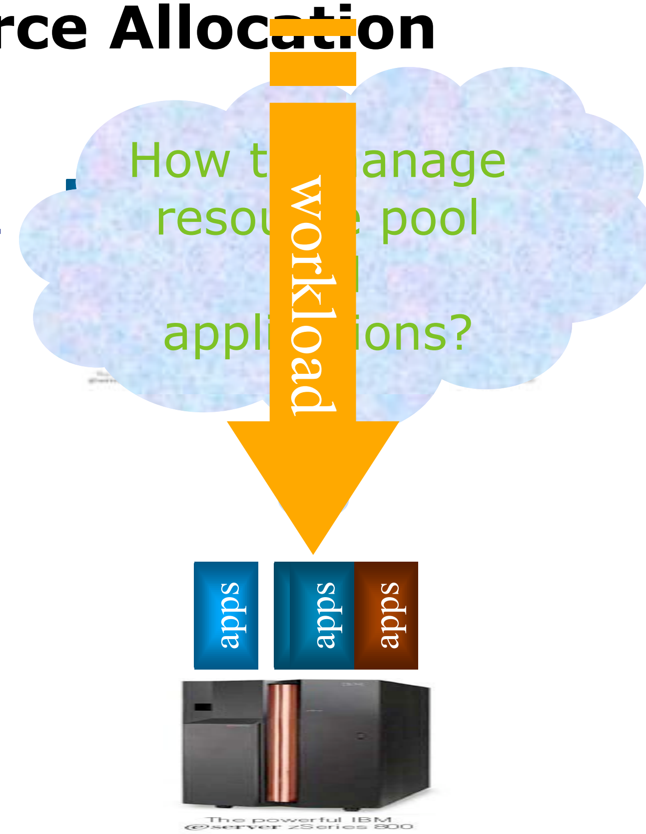
- Construct a parametric model of the target system
- Fix some parameters through experiments or learning
- Devise strategy for optimizing rest of parameters using runtime state as input
- Implement strategy as an *online controller*
- Use output of controller to configure system

Disadvantages: lack transparency and predictability, performance can be an issue, etc.

➡ Have developed a *hybrid* approach:

- *Offline* optimization and model solution to generate optimal configurations.
- Use generated rule sets at runtime (*policies).*

Application Context: Dynamic Resource Allocation



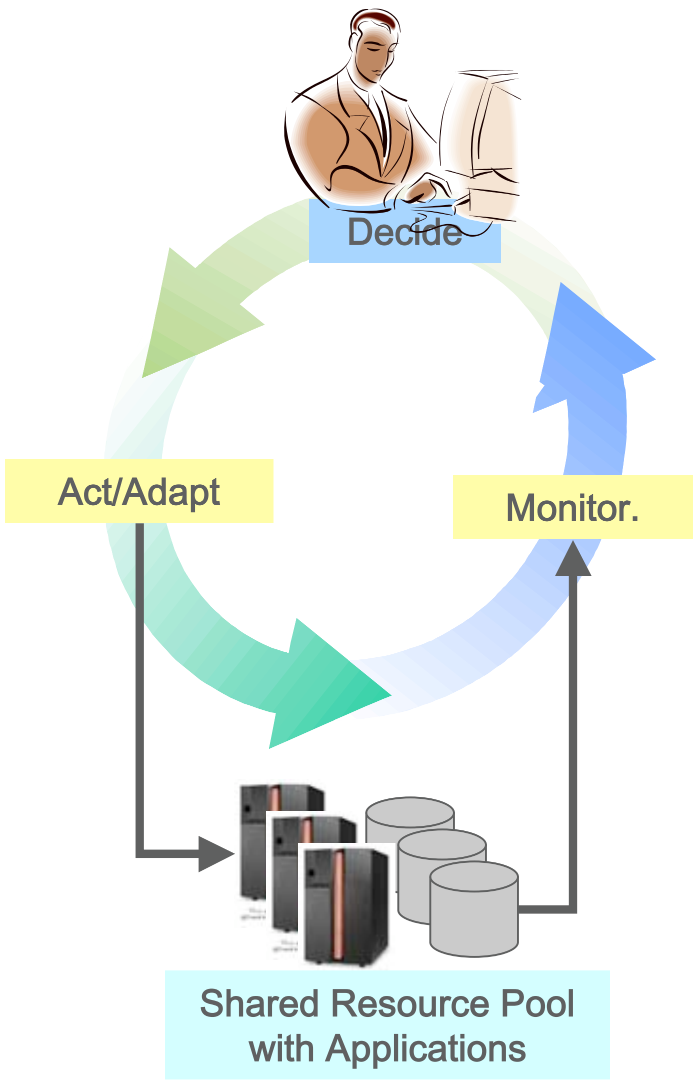
Costs of power, air conditioning, data center space, operations, low utilization, multi processor and multi core processors + virtualization technology:
⇒ **server consolidation**.

Promise: Cost reduction, handling of flash crowds, failures.
Challenge: unpredictable workload.

⇒More sophisticated **management system** and complex **adaptation policies** required.

⇒Focus on *multi-tier enterprise applications*: web server + application server + backend database

Runtime Resource Management



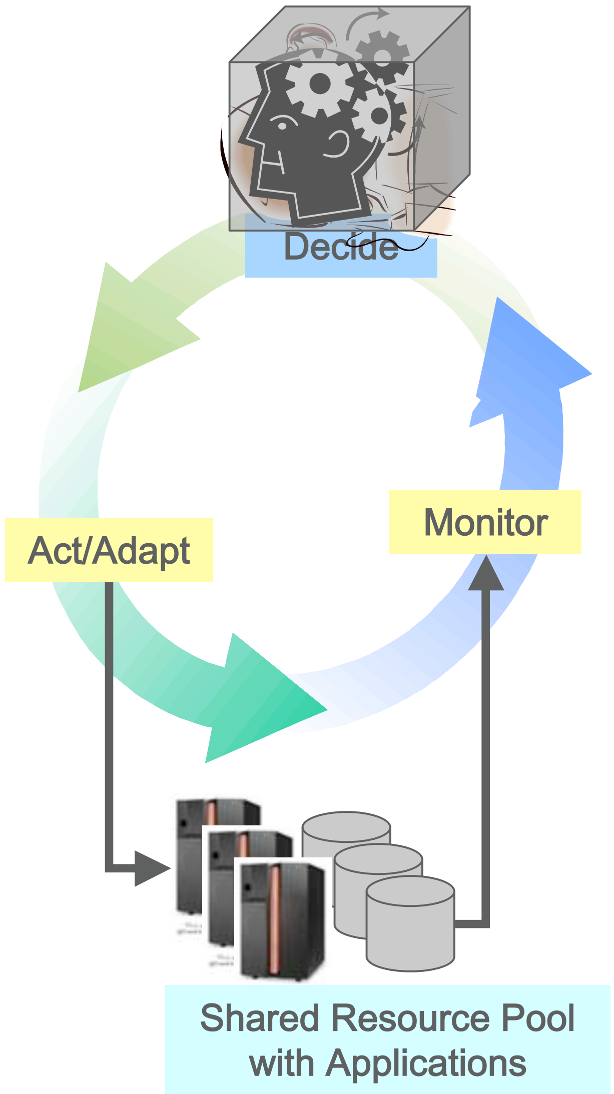
Monitor:

- resource utilization,
- response times,
- failure alarms.

Actions:

- Start/stop processes (e.g., adjust replication degree of a component).
- Migrate processes
- Adjust CPU allocation (e.g., virtual machine technology).

Current approach



Significant and obvious limitations of manual approach

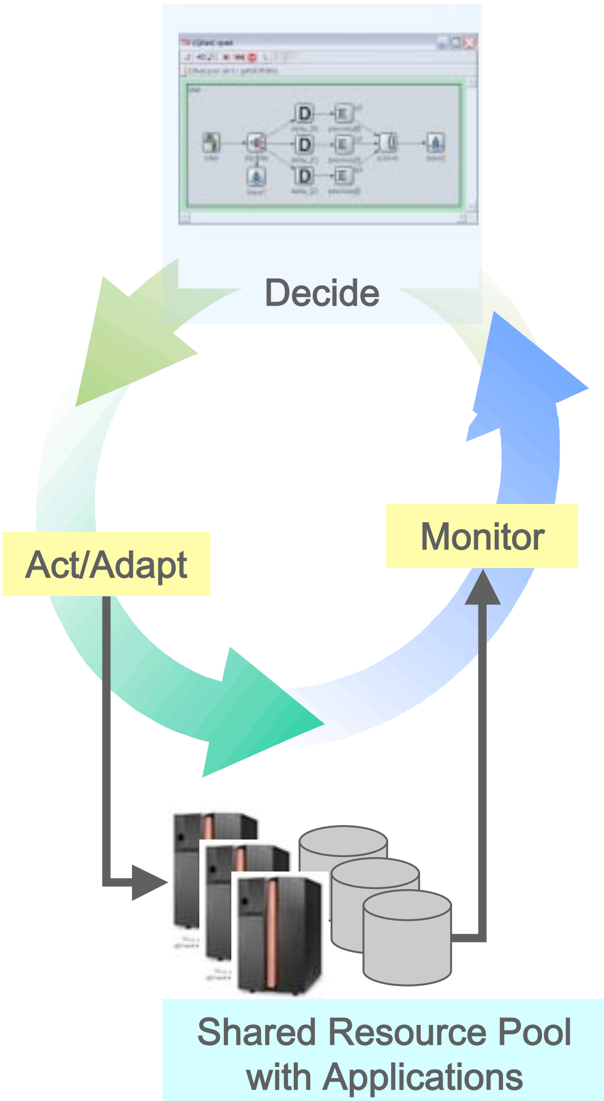
- Slow reaction time (10s of minutes).
- Difficult to consider all factors in a complex system.
- Human errors.
- Cost of 24/7 operations.

Solution: Replace operator with a rule-based management system.

Challenge: Developing rules.

➡Use stochastic models as the basic technology.

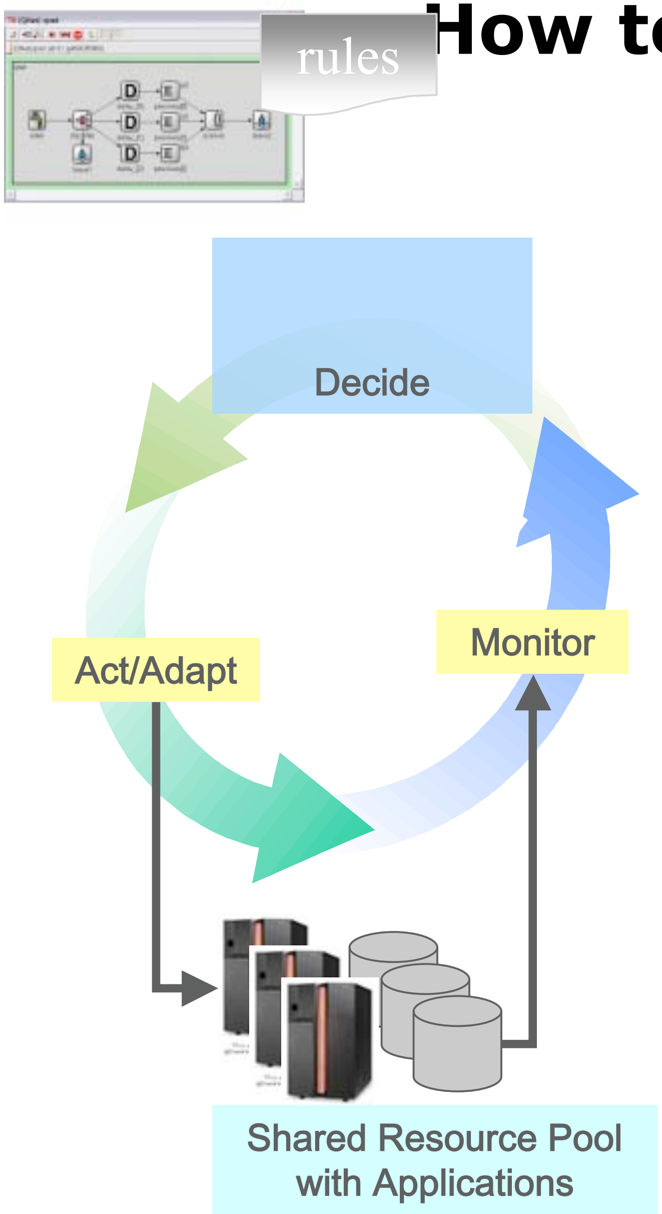
How to use models (1/2)



Model inline(MIL):

- Model(s) evaluated at runtime given current system workload as input.
- The rewards for alternative configurations can be calculated to determine a better configuration.

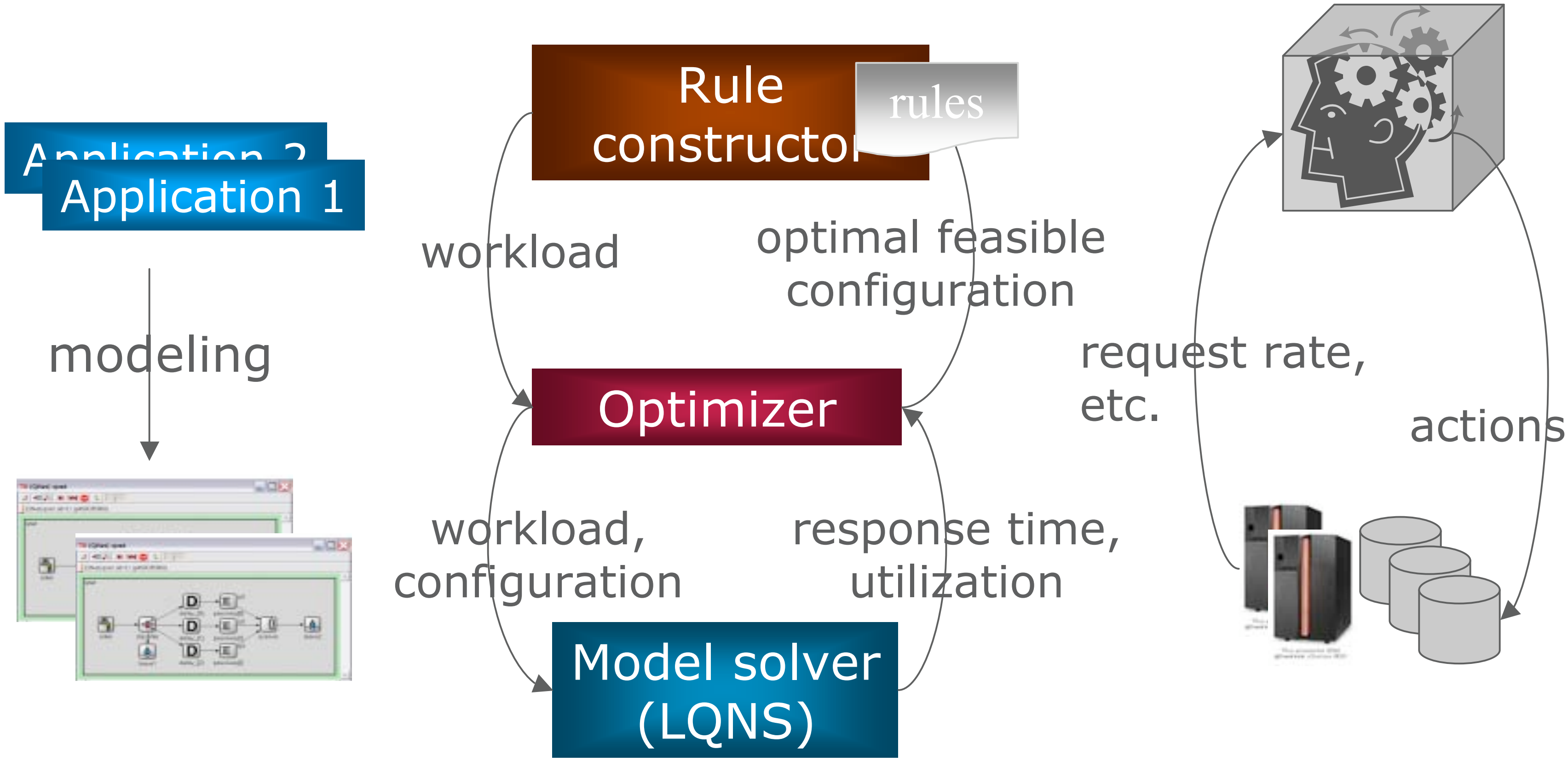
How to use models (2/2)



Model offline(MOL):

- Model(s) evaluated before system deployment using different workloads as inputs.
- Optimized configuration determined for each different workload mix.
- Adaptation rules generated based on model outputs.

MOL in action: Our Approach



Formal problem statement, then discuss steps bottom up.

Formal problem statement (1/2)

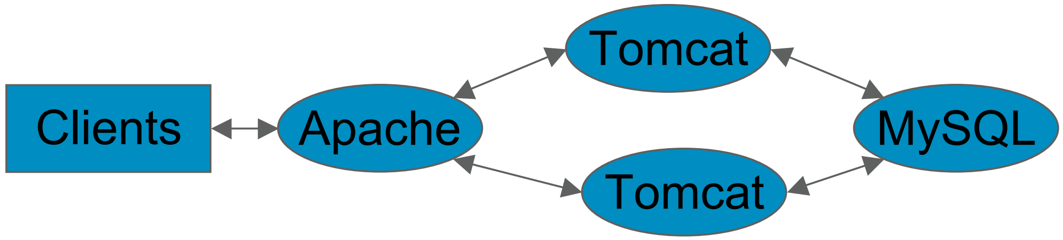
Given:

- **A set of computing resources R**
- **A set of applications A:**
  - each consists of a set of components/tiers
  - each component has a set of possible replication degrees
  - may support multiple transaction types.
- **For each transaction type,**
  - a transaction graph describes how the transaction uses application components
  - each component's service time
- **For each application, an SLA that, for each transaction type, specifies the desired (mean) response time and the reward/penalty for meeting/missing this time**

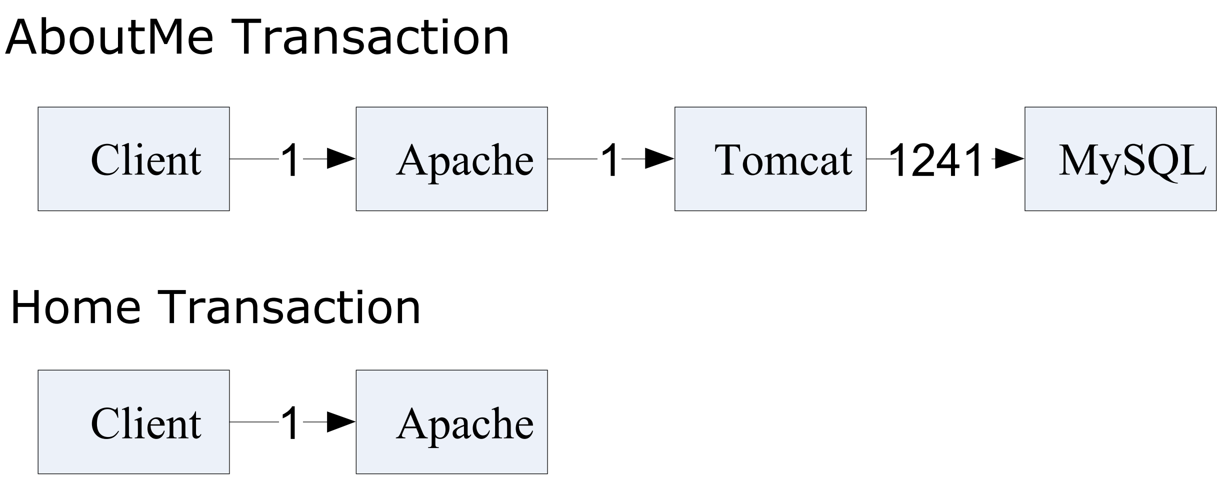
at&t

Example: RuBIS

RuBIS: a J2EE-based auction system.



26 different transaction types with very different behaviors.



at&t

Formal problem statement (2/2)

Measured at runtime:

- **Each application's workload for each transaction type**

Goal:

- **Configure the set of applications A on the resources R so that the reward with current workload is maximized**
- **Configuration:**
  1. Degree of replication for each component (of each app)
  2. Virtual machine parameters for a VM running the component (CPU fraction)
  3. Placement of VMs on the physical machines R

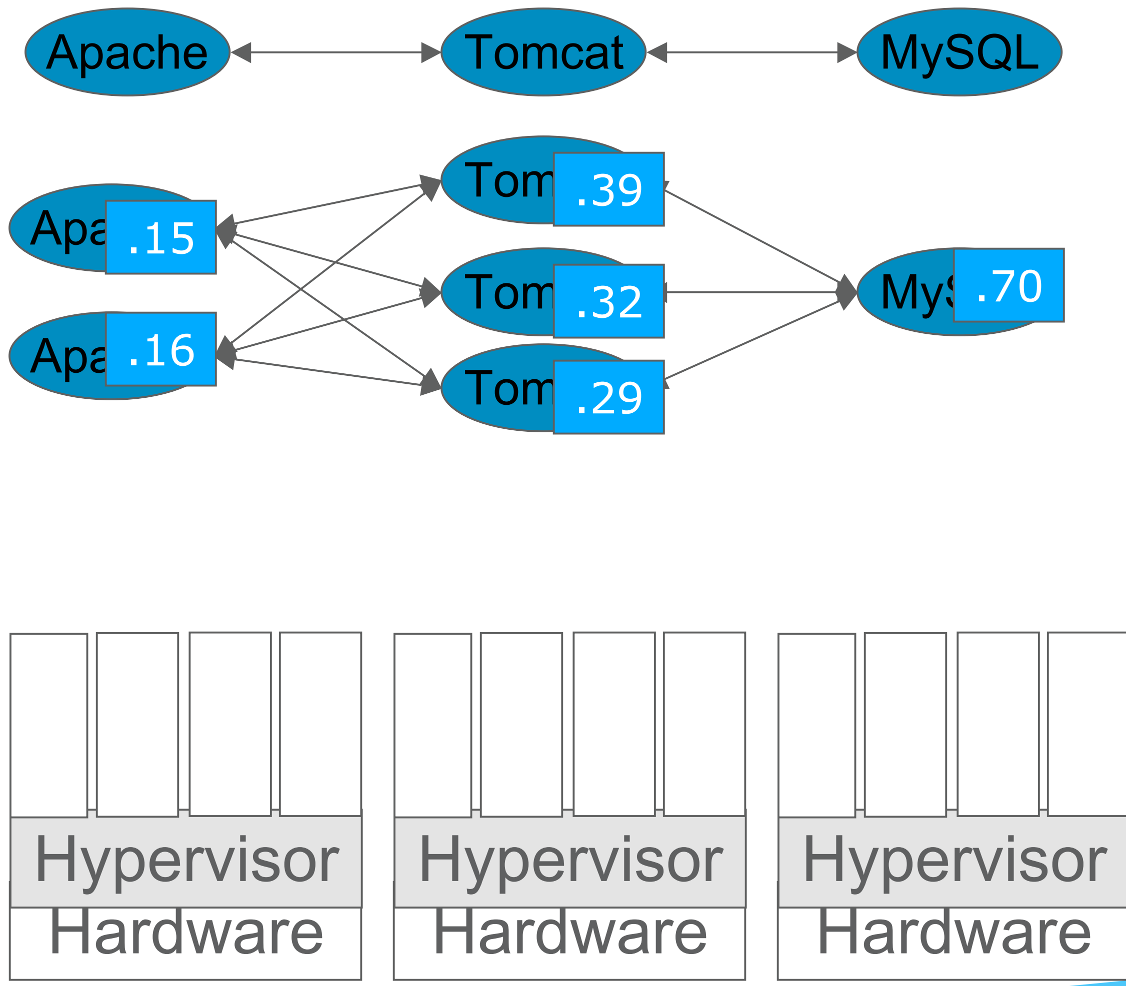
at&t

Example: RuBIS

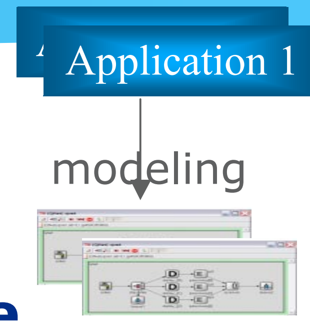
Application components:

Logical configuration:

Physical configuration:



Application Modeling



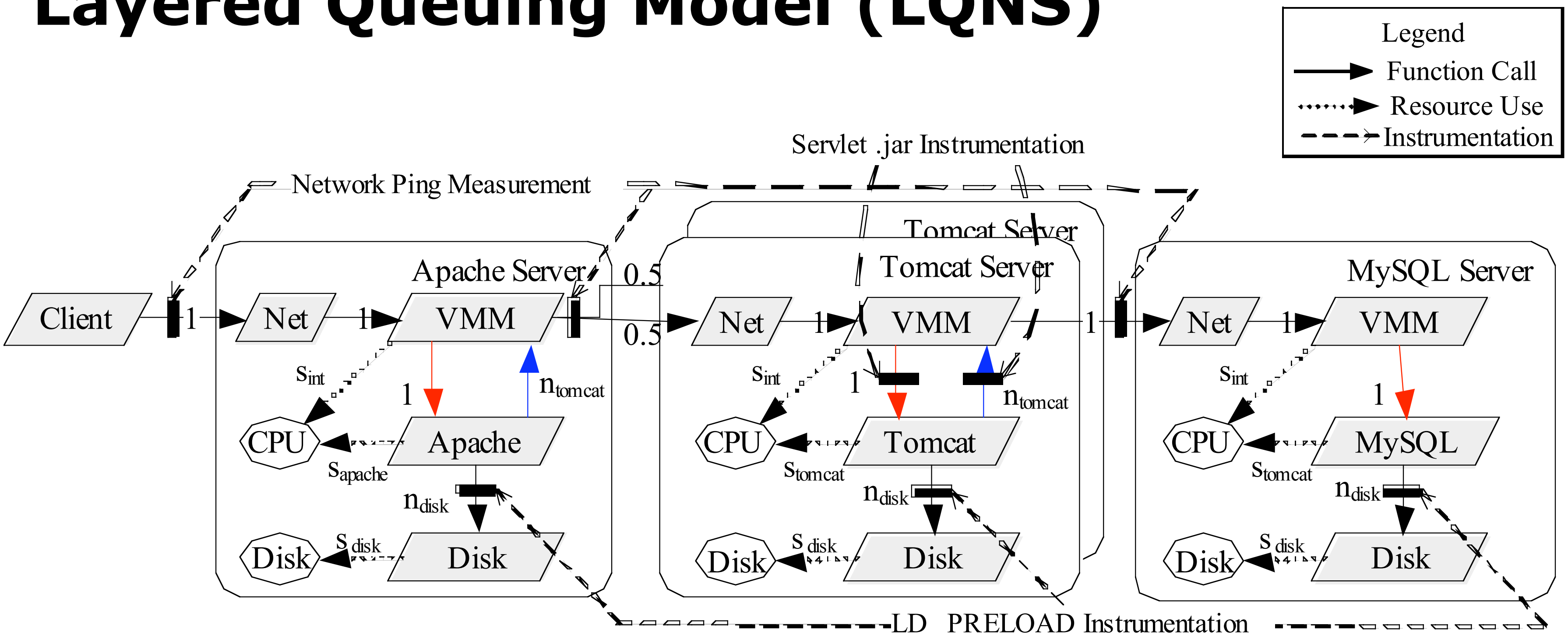
Regular queuing networks do not capture the complexity of layered, multi-tier software systems

- Multiple request classes and request fractions between classes
- Synchronous calls among servers
- Multiple interactions between servers

Layered queuing models

- Allow simultaneous resource possession
- Allow distinguishing between software (threads) and hardware bottlenecks

Layered Queuing Model (LQNS)



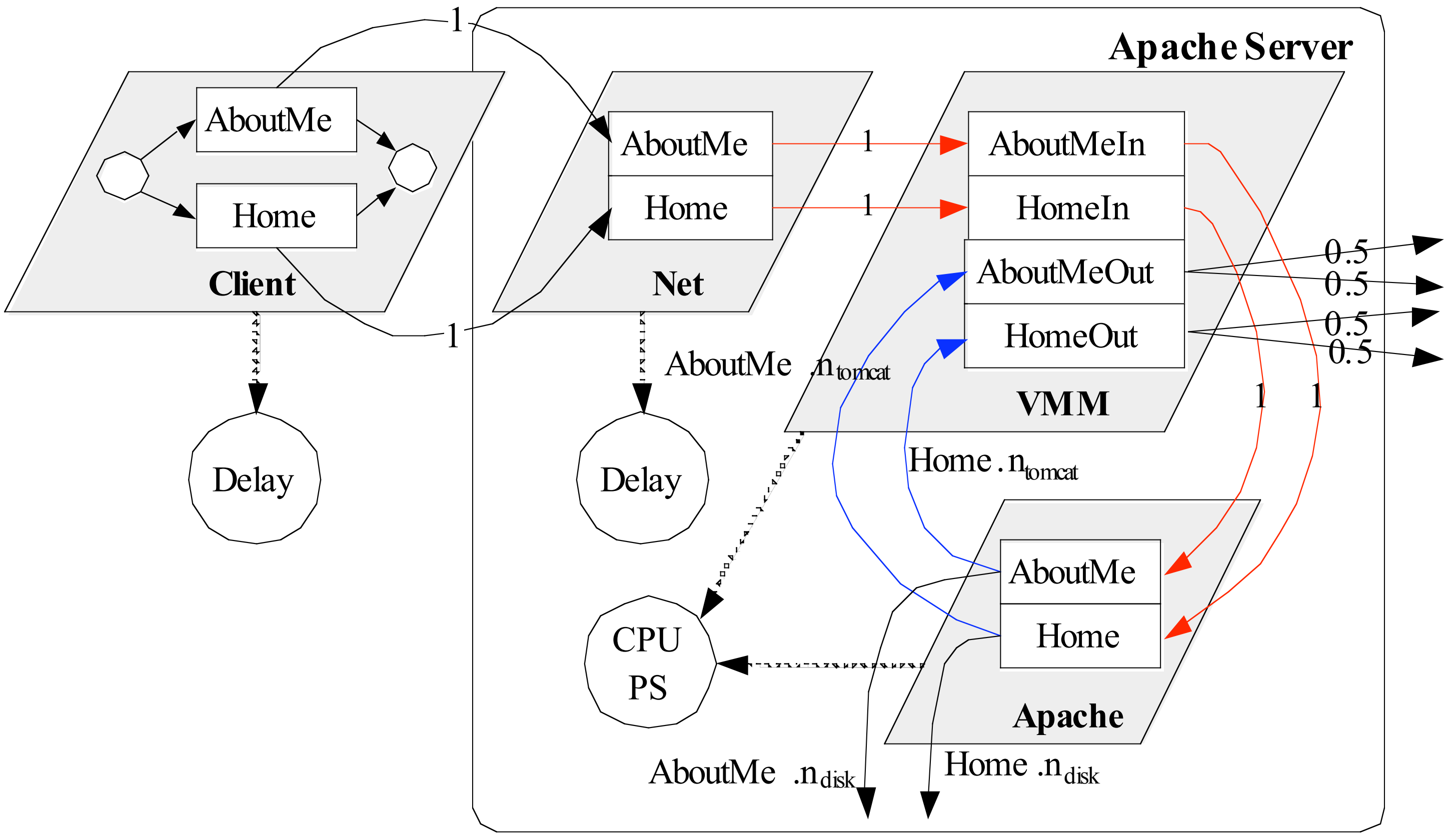
Software component replica = task

- number of servers per task = max number of threads in the component
- one entry for each transaction type

Task allocated to a virtual processor. Service time scaled by the CPU fraction of the virtual processor.

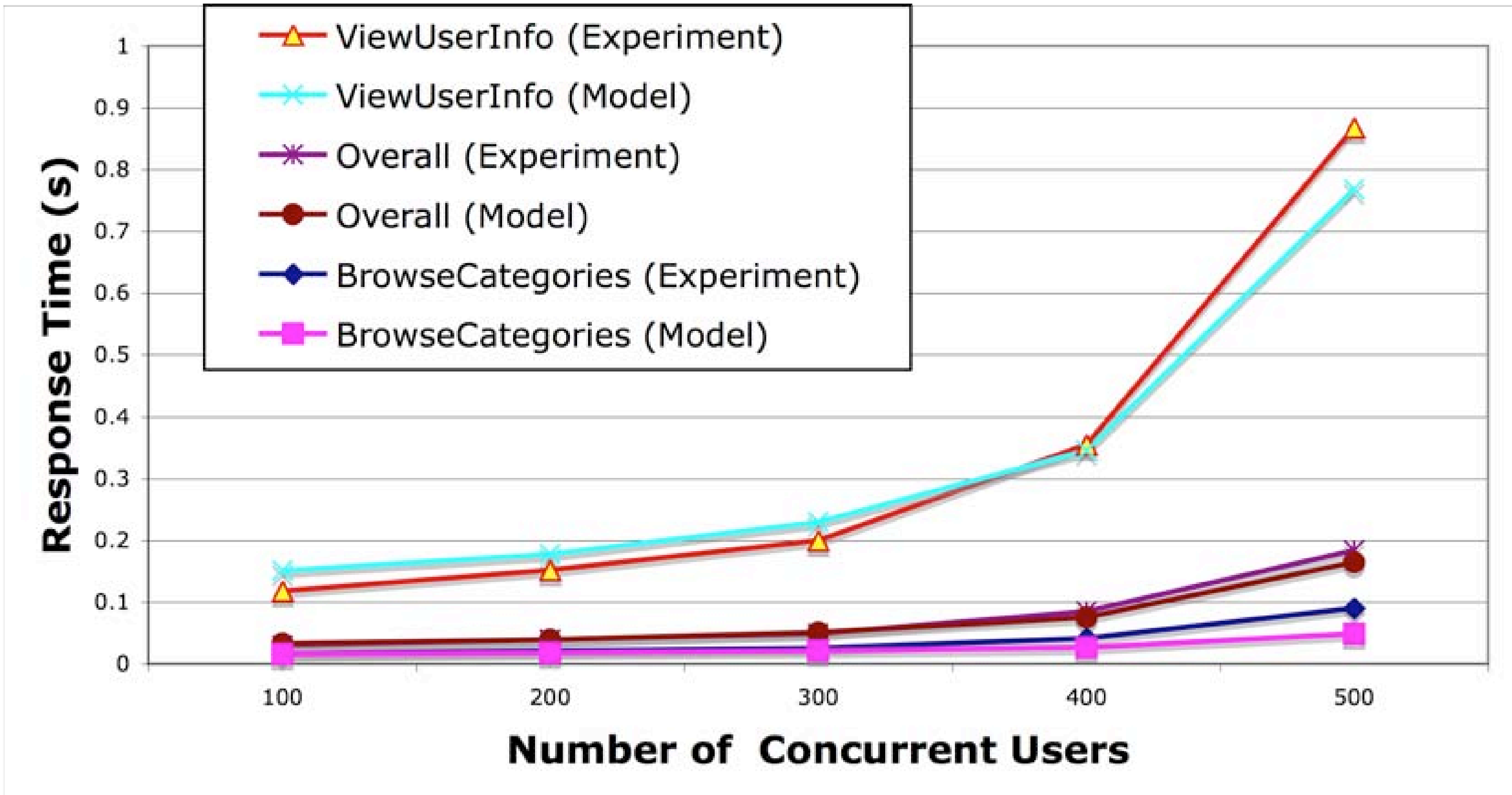
Do not model memory, disk, or network contention.

Layered Queuing Model (LQNS) - details



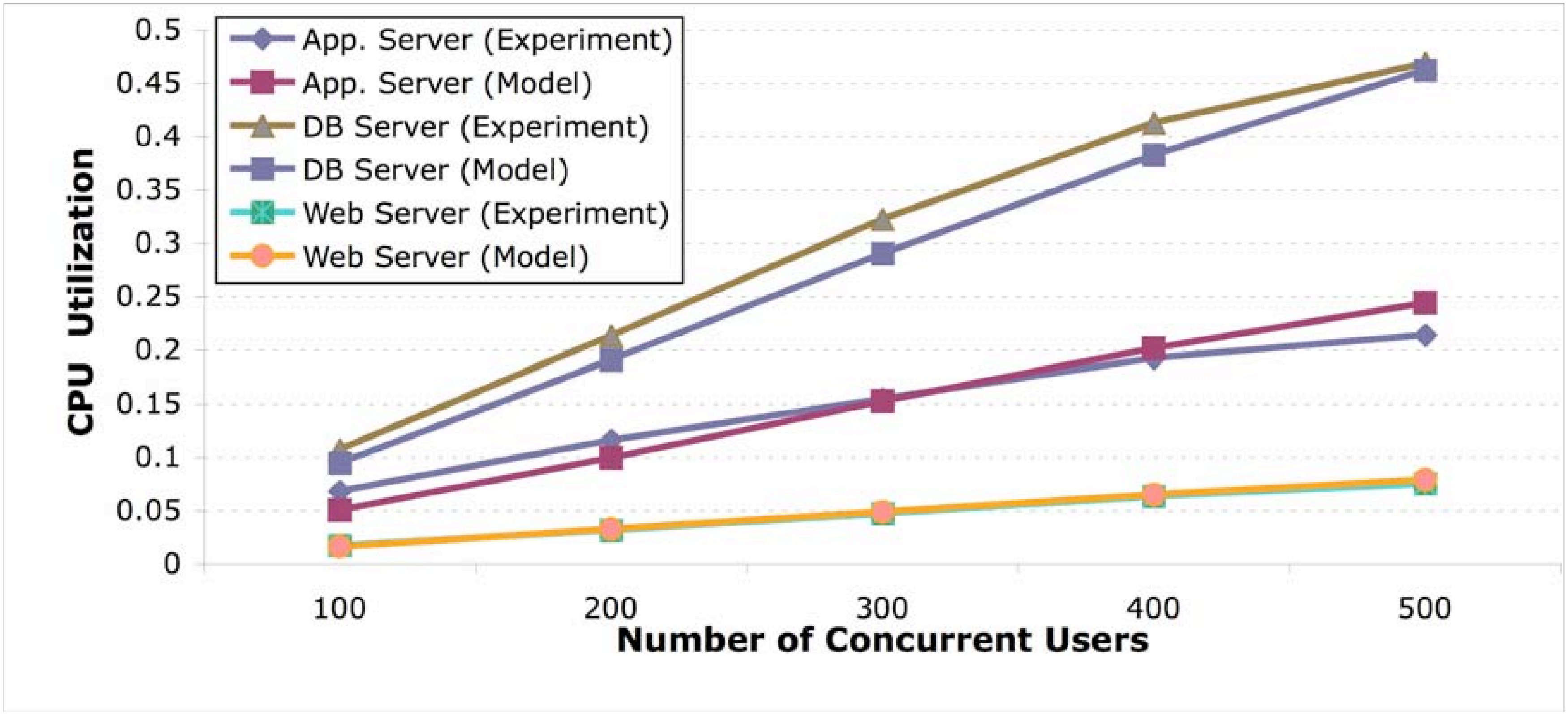
at&t

Model validation (1/3)



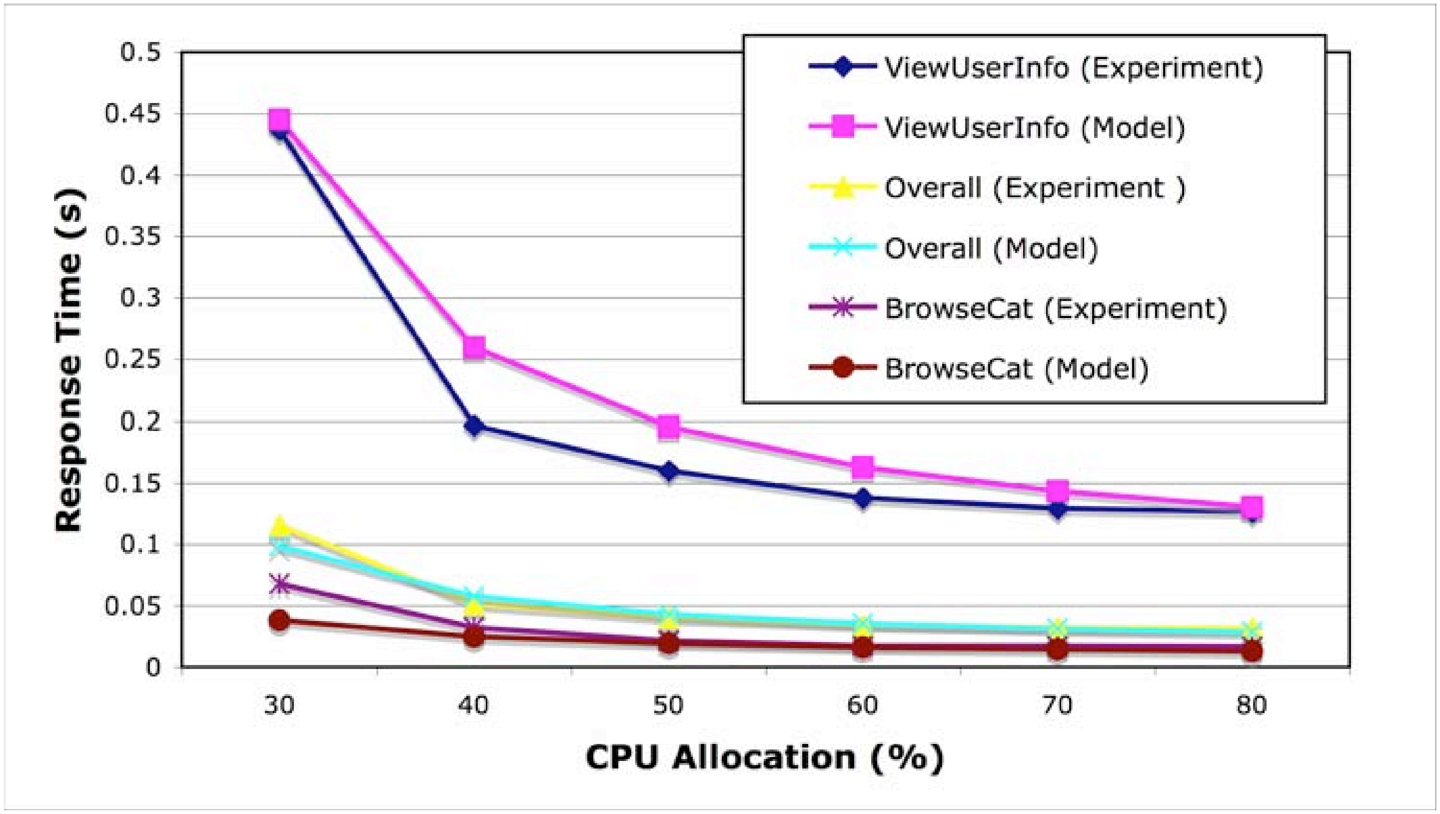
Model predicts response time at different request rates

Model validation (2/3)



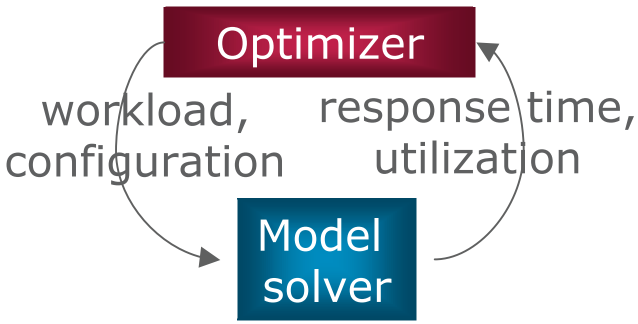
Model predicts CPU utilization at different tiers at different request rates.

Model validation (3/3)



Model predicts response time at different CPU allocations.

Optimization (1/2)



For a given workload, find the configuration with the maximum utility.

Huge parameter space to explore, NP-Complete problem.

Key techniques:

- Decouple logical configuration from physical component placement.
- Start from optimal configuration, search paths that reduce resource utilization while minimally reducing utility.

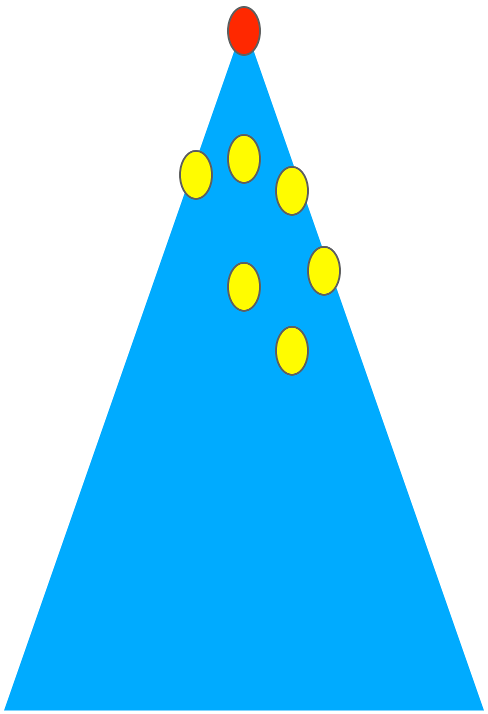
Observations:

- Utility decreases when response time increases.
- Response time increases when number of replicas is reduced.
- Response time increases when CPU fraction is reduced.

Optimization (2/2)

Optimal configuration:

- Each component of each application has the maximum number of replicas, each with 100% of a CPU of their own.
- Use model solver to get actual resource utilizations ρ and the response times (for calculating utility U).



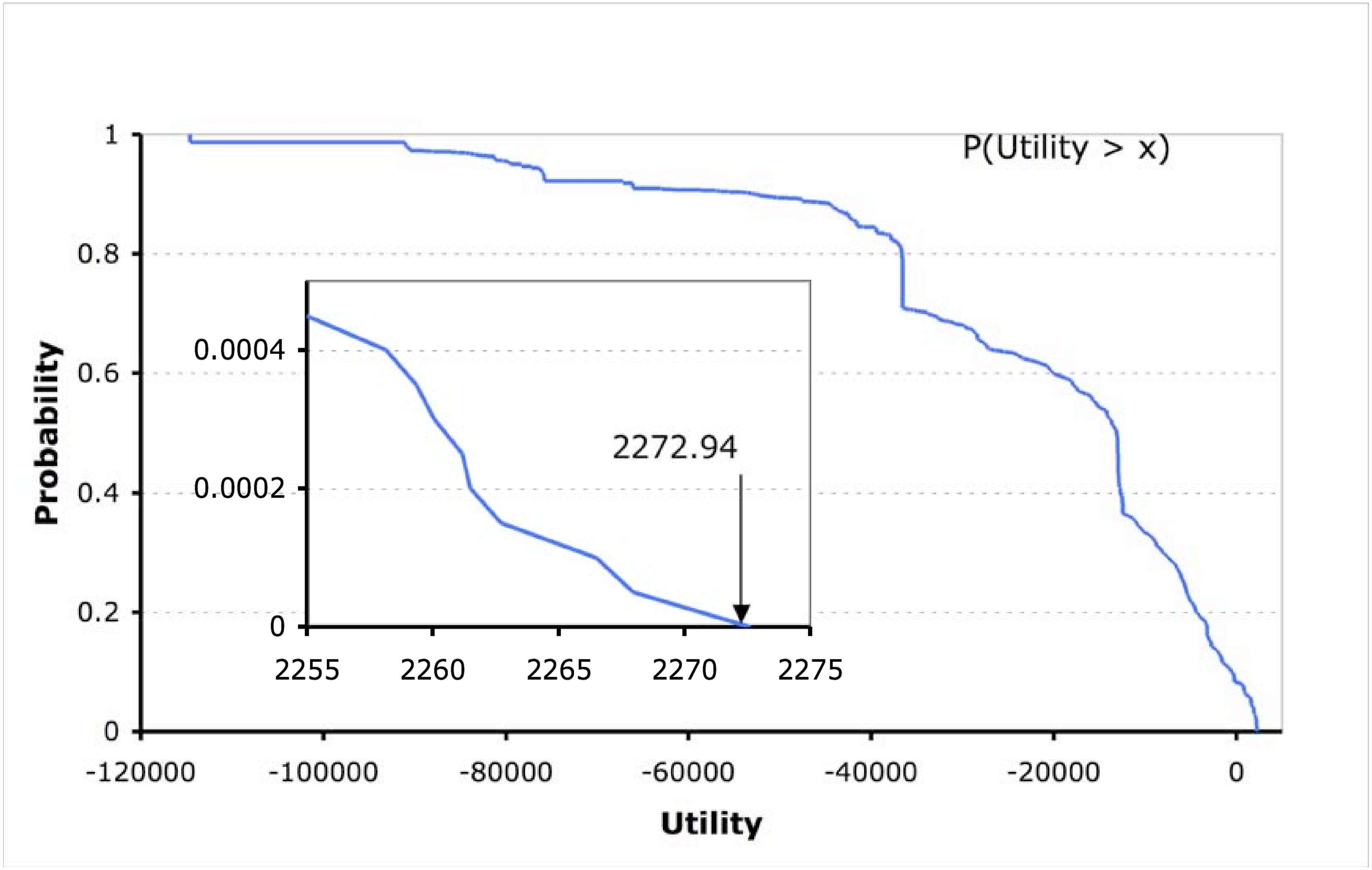
Algorithm:

1. Use *bin-packing* algorithm to find out if the utilizations ρ can be fitted in the actual resources R.
2. If not, evaluate possible alternatives for reducing utilization:
   - Reduce number of replicas for some component
   - Reduce CPU fraction for some virtual machine by 5%
3. Determine the actual utilizations and utility for the different options.
4. Choose the one that maximizes:

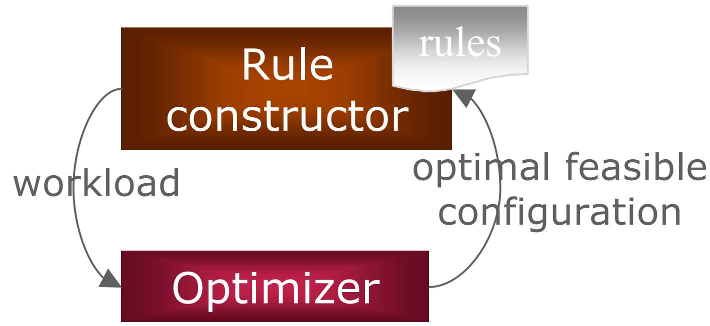
$$\frac{\sum_{i,j,k}\rho_{new} - \sum_{i,j,k}\rho_{old}}{U_{new} - U_{old}}$$

5. Repeat until configuration found

Optimality of the generated policies



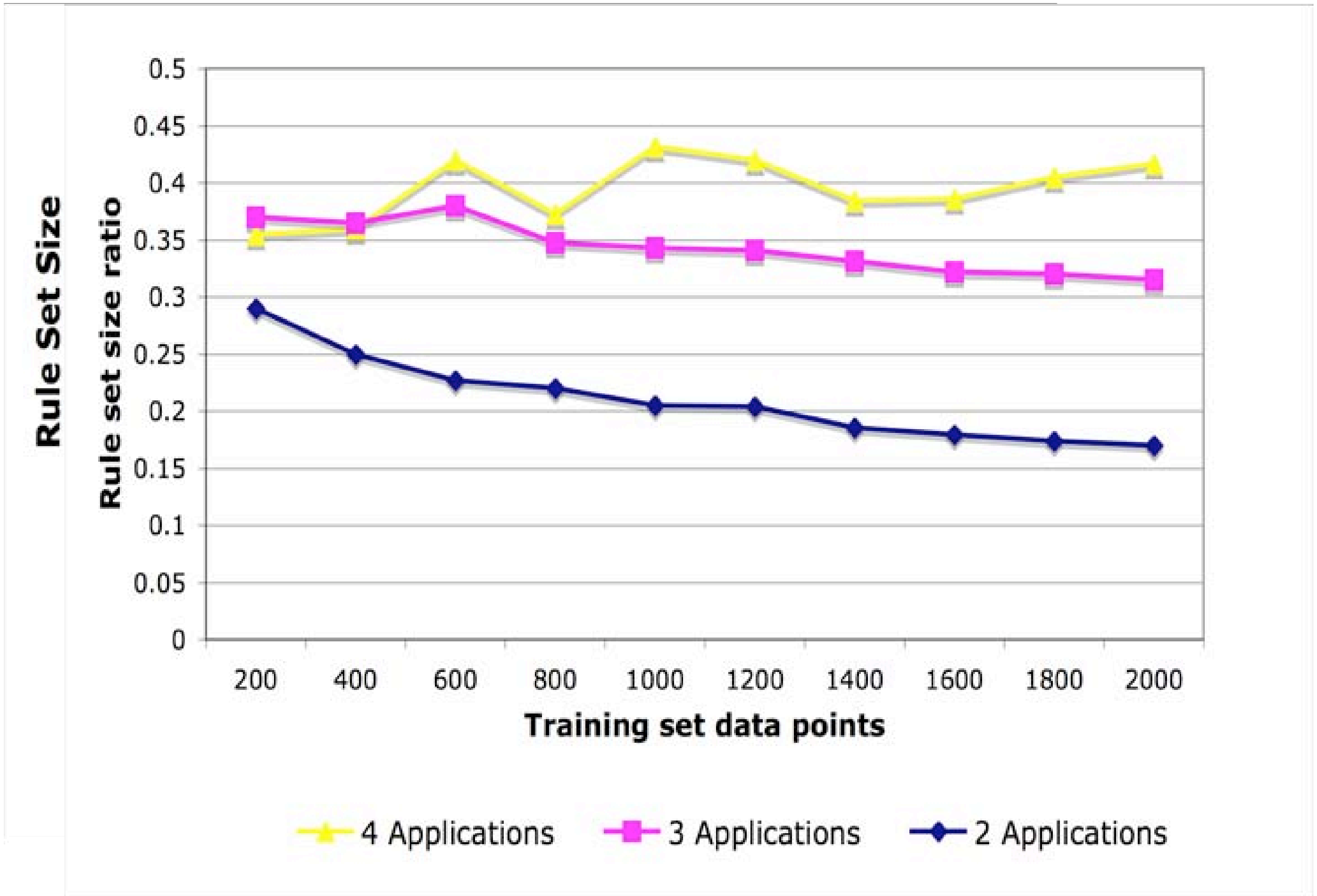
Rule Set Construction



Constructor:

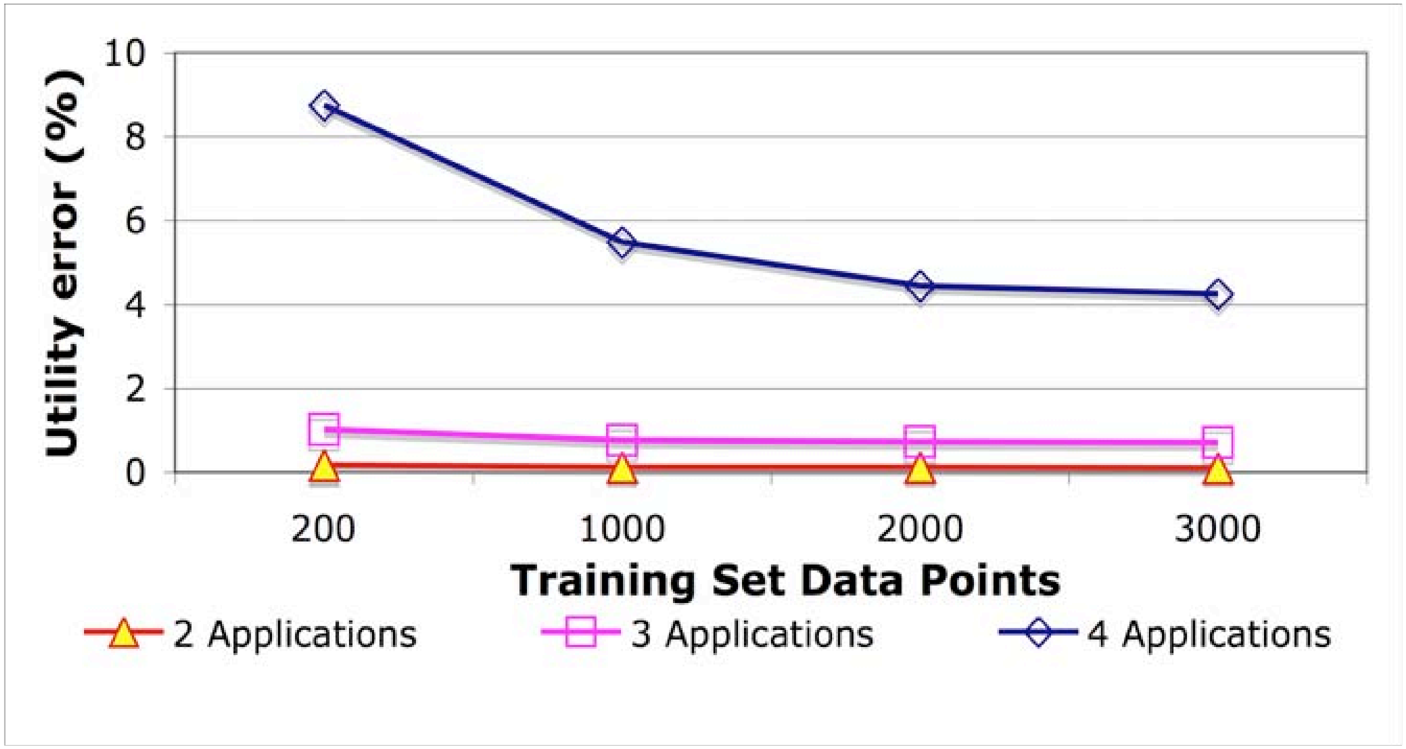
- Randomly generates a set of workloads WS based on SLA for each application.
- Invokes optimizer to find optimal configuration c for each $w \in WS$.
- Gives (w, c) pairs (*raw rule set*); still need interpolation for workloads $\notin$ WS.
- Use decision tree learner in the Weka machine learning toolkit to generate decision tree.
- Linearize into nested "if-then-else" rule set.

Rule set size



The size of the rule set increases when the number of training set data points increases.

Utility error



The utility error decreases, and then stabilizes, with number of training set data points.

Summary and Open Questions

Summary:

- Dynamic resource management crucial for server consolidation
- Development of adaptation policy rules a challenging problem
- Propose a hybrid approach based on offline modeling for rule generation

Open questions:

- Can the set of rules be simplified with minimal loss of accuracy?
- How do rules compare with human generated rules?
- Given the current configuration, how to get to the optimized configuration (at minimum cost).

Thank you!