Engineering Autonomic Systems

Joseph L. Hellerstein
Google
June 17, 2009
Keynote, International Conference on Autonomic Computing
Autonomic Computing (AC)

- Self-managing systems
  - Configuration, healing, optimization, protection, ...

- Core technology: Policy-directed automation
  - Key requirements
    - Simple, declarative policy
    - Automation is robust to change

- Engineering autonomic systems requires methodologies for developing policy-directed automation
Agenda

- When use policy-directed automation?
- How design policy-directed automation?
- How assess policy-directed automation?
- Related work
- Challenges & research
When Automate Software Distribution?

Manual Software Distribution

2 hours per install target

*Brown & Hellerstein, HotOS 2005.*
Automating Software Distribution

Automated Software Distribution

Diagram showing the process of automated software distribution.
Automating Software Distribution

Automated Software Distribution

Operation

Software Package
- Obtain Source Distribution
- Perform Pilot Install
- Identify Customizations
- Create Deployment Wrapper
- Test Wrapper
- Publish Package

System Administrator
- Research Available Packages
- Select Targets
- Invoke Installer
- Check Results
- Package Problem?
- Package?
- End
- Diagnose Endpoint
- Repair Endpoint

Infrastructure Maintainer
- Invoke Wrapper
- Validate Prereqs.
- Prereqs Met?
- Configure Installer
- Perform Installation
- Verify Installation
- Log Results
- Remove Install Remnants

Automation

Manual Software Distribution

Off-Prep

Obtain Source Distribution
Update Prerequisites
Pilot
Configure Installer
Perform Installation
Verify Installation
Log Results
Remove Install Remnants
Notify Problem
# Automating Software Distribution

## Automated Software Distribution

<table>
<thead>
<tr>
<th>Software Package</th>
<th>Operation</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain Source Dist'n</td>
<td>Perform Pilot Install</td>
<td>Identify Affected Packages</td>
</tr>
<tr>
<td>Identify Customizations</td>
<td>Create Deployment Wrapper</td>
<td>Select Targets</td>
</tr>
<tr>
<td>Test Wrapper</td>
<td>OK?</td>
<td>Y</td>
</tr>
<tr>
<td>Publish Package</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Administrator</th>
<th>Select Targets</th>
<th>Invoke Installer</th>
<th>Check Results</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req. Package Exists?</td>
<td>N</td>
<td>Y</td>
<td>End</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Infrastructure Maintainer</th>
<th>Diagnose Endpoint</th>
<th>Repair Endpoint</th>
<th>Automation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upgrade Distribution Servers</td>
<td>Upgrade Endpoints</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Automating Software Distribution

Automated Software Distribution

<table>
<thead>
<tr>
<th>Operation</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify</td>
<td>Identify Affected Packages</td>
</tr>
<tr>
<td>Create</td>
<td>Wrapper Change Required?</td>
</tr>
<tr>
<td>Test</td>
<td></td>
</tr>
<tr>
<td>Publish Package</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20–40 hours per software package
Cost vs. Benefit of Automation

- **Cost of manual install**
  - 2 hours per install target

- **Cost of automation**
  - 20–40 hours per package
    - Prepare, test, distribute packages

- **Break even at 10 to 20 targets**
Consider the **total benefit** of automation, not just the **savings** while operating automatically.
Steps for Designing Policy-Directed Automation

- Requirements
  - Can be used on real products, by real software engineers, and contributes real value

- Examples in which I’ve been involved
  - Utilities throttling in IBM DB2
    - Control rate of background work
  - Self-tuning memory management for IBM DB2
    - Optimize the allocation of memory
  - Microsoft .NET thread pool
    - Optimize performance for 1B computers with diverse workloads

*Diao, Hellerstein, Parekh, Accepted in IGI Publishers.*
Design Steps

1. Qualitative Modeling
2. Measurability & Controllability
3. Quantitative Modeling
4. Controller Design
5. Implementation

*Originally proposed by Sujay Parekh in his PhD thesis.
Design Steps

1. Qualitative Modeling
2. Measurability & Controllability
3. Quantitative Modeling

Formulate Policy
Construct Block Diagram

Controller Design

5. Implementation
What is the policy for throttling utilities?
- Cannot measure progress of utilities

Policy
There should be no more than an x% performance degradation of production work as a result of executing administrative utilities
Qualitative Modeling

Construct Block Diagram

Diagram shows relationships in policy-directed automation
- Policy
- Target system – control input, measured output
- Controller
- Other elements

R: Impact Limit
E: Error
U: Control knob
Y: Pageometer (pages/sec)
θ: Model parameters
Y*: Baseline perf
Problem – load balance memory pools

Solves optimization problem with regulatory control

- Regulates to the average load

*Diao et al., RTAS 2004.*
Design Steps

1. Qualitative Modeling

2. Measurability & Controllability

3. Quantitative Modeling

4. Controller Design

5. Implementation
Qualitative Modeling: Measurability & Controllability

- Measurability issue
  - No metric for %degradation

![Diagram](image)
Qualitative Modeling: Measurability & Controllability

- Measurability issue
  - No metric for %degradation
Qualitative Modeling: Measurability & Controllability

- Need “knob” that regulates the trade-off between production throughput and utility progress
Qualitative Modeling: Measurability & Controllability

- Need “knob” that regulates the trade-off between production throughput and utility progress

New knob: Utility Sleep Time on each cycle.
Design Steps

1. Qualitative Modeling
2. Measurability & Controllability
3. Quantitative Modeling
4. Controller Design
5. Implementation
Quantitative models are needed for
- Estimating throughput ($Y$) of DB2
- Estimating % degradation

Much literature on models for systems
- Queuing, machine learning, ...
- Personal experience
  - Simple linear models are very robust (If include some adaptation)
Design Steps

1. Qualitative Modeling
2. Measurability & Controllability
3. Quantitative Modeling
4. Controller Design
5. Implementation
Controller Design: System ID

Control Setting $u(k)$ \rightarrow \text{Target System} \rightarrow \text{Measured Output} $y(k)$

Dynamical Model of Target System

$y(k + 1) = ay(k) + bu(k)$

$= a^2 y(k - 1) + abu(k - 1) + bu(k)$

Transfer Function $S(z)$

$zY(z) = aY(z) + bU(z)$

$S(z) = \frac{Y(z)}{U(z)} = \frac{b}{z - a}$

$a$ is the pole of $S(z)$
Controller Design: Controller Construction

\[ r(k) + e(k) \xrightarrow{\text{Controller} \ G(z)} u(k) \xrightarrow{\text{Target System} \ S(z)} y(k) \]

\[ F_R(z) = \frac{Y(z)}{R(z)} = \text{Closed Loop Transfer Function} \]

**Integral Control Law**

\[ u(k) = u(k - 1) + K_I e(k) \]

**Poles of** \( F_R(z) \)

\[ K_I = 0.1 \quad K_I = 1 \quad K_I = 5 \]

![Graphs showing system response with different K_I values](image-url)
Design Steps

1. Qualitative Modeling
2. Measurability & Controllability
3. Quantitative Modeling
4. Controller Design
5. Implementation

The hard part.
Left as an exercise.
Step 6: Assessment
Example: .NET ThreadPool Controller

CLR = Common Language Runtime

CLR VM Services
- Extension Model
- Garbage Collection
- ThreadPool
...

Host Environment
(OS, DBMS, WebBrowser)

ThreadPool Operation
- CLR Appl uses QueueUserWorkItem() to initiate asynchronous work
- CLR ThreadPool places work in a queue until thread is available
- ThreadPool Controller regulates concurrency level to optimize throughput

Finds concurrency level (# threads) that maximizes throughput
  ◦ Uses “hill climbing” with statistics (stochastic gradient analysis)

Distributed to almost all of the 1B windows machines
Challenges In Assessment

- Test matrix is very large (100K to 10M test cases) since
  - .NET runs on very diverse machines and workloads
  - Control algorithm has many parameters
    - Control gain (how much to increase/decrease # threads)
    - Significance level for comparing throughputs
    - Magnitude of a random move if “stuck”
    - Minimum number of completions (data points) to make a decision

- Need efficient test environment
**Approaches to Assessment**

**Real Threadpool Real Controller**
- **W** (W=Workload)
  - Benchmark Driver
  - Synthetic Work
  - Threadpool
  - Controller
  - Metrics
  - Allocations
- **P** (P=Parameters)
  - **Accurate** but **slow** (months to years)

**Model Threadpool Model Controller**
- **W**
  - Model of Threadpool & Controller
  - **Fast** (seconds to minutes); **inaccurate** (difficult to model)

**Model Threadpool Real Controller**
- **W**
  - Model of Threadpool
  - **Reasonably fast** (minutes to hours) and **acceptable** accuracy

**Metrics**
- **Allocations**

**Synthetic Work**
- **Work** (W=Workload)

**Threadpool Model**
- **Throughput** vs **Concurrency** graph
Summary of Methodology

General flow is top to bottom
Lots of iteration and re-do

Step 0: Determine if Automation is desirable
- Qualitative Modeling
  - Formulate Control Objective
  - Construct Block Diagram

Step 1: Assess & Ensure Controllability, Observability
- Quantitative Modeling
  - System Identification
  - Choose Mathematical Representation

Step 2: Controller Design
- Select Control Law
- Parameterize Controller

Step 3: Implement Controller

Step 4: Controller Assessment
## Some Related Work

<table>
<thead>
<tr>
<th>Area of Innovation</th>
<th>Component Models</th>
<th>System Models</th>
<th>Controller Design</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Area of Innovation</td>
<td>Single Server Systems</td>
<td>Multiple Servers</td>
<td>Networks</td>
<td>Power</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------</td>
<td>------------------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>Component Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controller Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Some Related Work

<table>
<thead>
<tr>
<th>Area of Innovation</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single Server Systems</strong></td>
<td><strong>Multiple Servers</strong></td>
</tr>
<tr>
<td>Component Models</td>
<td>RWK’03</td>
</tr>
<tr>
<td>Systems Models</td>
<td>PGHTJB’02</td>
</tr>
<tr>
<td>Controller Design</td>
<td>ASS’01, LSA’01, PRDCHLH’04, HASL’07</td>
</tr>
<tr>
<td>Software</td>
<td>AB’99, EHM’08</td>
</tr>
</tbody>
</table>
Challenges and Research

Cost/benefit analysis

Control patterns
• CS problem, control objective, metrics, controller, component interactions, model/controller adaptation

Integration of multiple, interacting controllers
Design for controllability
• Control mechanisms with little dead–time, linear gains

Teach control basics to SW engineers
Integration with software development tools
• Model construction
• Design of experiments for model evaluation
• Model assessment
• Controller design

Automated controller generation

Benchmarks for adaptive systems
Efficient unit testing of controllers

Case studies, Applications beyond performance/availability/power