Managing Web Server Performance with AutoTune Agents

by Y. Diao, J. L. Hellerstein, S. Parekh, J. P. Bigus

Presented by Fu Kai
Contents

• Introduction
• Apache web server and performance tuning
• Server self-tuning with Autotune agents
  – Modeling Agent
  – Run-time Control Agent
  – Controller Design Agent
• Experimental assessment
• Conclusions
Introduction

- Managing the performance of E-commerce sites is challenging
  - Site content changes frequently
    - Dynamically varying workloads
  - Some applications of control theory to computing systems include
    - Flow and congestion control, differentiated caching and web service, multimedia streaming, web server performance, e-mail server control

- To maintain good performance
  - System administrators must tune their information technology environment
    - Manual effort can be time consuming and error-prone, and requires highly skilled
Introduction (Cont’d)

• Applications all provide a degree of autonomic behavior by providing algorithms
  – To automatically control some aspect of a computing system’s operation

• In this paper...
  – Proposing an agent-based solution
    • Automates the ongoing system tuning
    • Automatically designs an appropriate tuning mechanism for the target system
Apache Web Server and Performance Tuning

• The application-level tuning parameters in Apache Web server
  – MaxClients: The number of simultaneous requests that will be served
  – KeepAlive: Whether or not to allow persistent connections

• Significant parameters to effect CPU and memory utilization
  – Increasing MaxClients: Increasing both CPU and memory utilizations.
  – Decreasing KeepAlive: Allows worker process to be more active.
    • Directly results in higher CPU utilization.
    • Indirectly increases memory utilization (more clients can connect)
Results of Tuning the Apache web Server
Effects of Dynamic Workloads
Server Self-Tuning with AutoTune Agents

- **Solution**
  - Multiple agents
    - Automate the entire methodology of controller design
    - Perform the on-line system control
      - Agents are implemented using the ABLE (Agent Building and Learning Environment)
        » Java**based toolkit
        » ABLE: provides a comprehensive library of intelligent reasoning and learning components
Architecture of the autotune agents
Modeling Agent

- Quantifying the relationship between the tuning parameters and performance metrics
- $A, B : 2 \times 2$ matrices
  - Include modeling parameters
  - Can be identified using the least squares method

$$
\begin{bmatrix}
    \text{CPU}_{k+1} \\
    \text{MEM}_{k+1}
\end{bmatrix} = A \cdot \begin{bmatrix}
    \text{CPU}_k \\
    \text{MEM}_k
\end{bmatrix} + B \cdot \begin{bmatrix}
    \text{KeepAlive}_k \\
    \text{MaxClients}_k
\end{bmatrix}
$$
Run-time Control Agent

- Implementing a state feedback controller
  - To make control decisions based on feedback of errors

\[
\begin{bmatrix}
\text{KeepAlive}_k \\
\text{MaxClients}_k
\end{bmatrix} = K_P \cdot \begin{bmatrix}
\text{CPU}_k \\
\text{MEM}_k
\end{bmatrix} + K_I \cdot \sum_{j=1}^{k-1} \begin{bmatrix}
\text{CPU}_j \\
\text{MEM}_j
\end{bmatrix}
\]

- \( K_P \): Proportional control gain for fast response
- \( K_I \): Integral control gain for removing steady-state error
Controller Design Agent

- To design the parameters $K_p, K_i$
- Choosing the controller parameters based on minimizing the following quadratic cost function:

$$J(K_p, K_i) = \sum_{k=1}^{\infty} \begin{bmatrix} e_{CPU,k} \\ e_{MEM,k} \\ v_{CPU,k} \\ v_{MEM,k} \end{bmatrix} \cdot Q \cdot \begin{bmatrix} e_{CPU,k} \\ e_{MEM,k} \\ v_{CPU,k} \\ v_{MEM,k} \end{bmatrix} + \begin{bmatrix} \text{KeepAlive}_k \\ \text{MaxClients}_k \end{bmatrix} \cdot R \cdot \begin{bmatrix} \text{KeepAlive}_k \\ \text{MaxClients}_k \end{bmatrix}$$

- $v_{CPU,k} = \sum_{j=1}^{k-1} e_{CPU,j}$

$$R = \text{diag}(r_1, r_2)$$

$$Q = \text{diag}(q_1, q_2, q_3, q_4)$$

- Q and R perform some scaling functions in addition to determining a trade-off between control error and control variability
Experimental Environment

- Environment: Linux 2.2.16, Apache 1.3.19
  - Workload generator: WAGON (Web Traffic Generator and Benchmark)
  - File access distribution: Web Stone

- Dynamic workload
  - Web pages generated through CGI
  - The session following a Poisson distribution
  - A rate of 10 sessions per second
Experimental Assessment(1)

- Results of autonomically tuning the Apache Web server
Experimental Assessment (2)

- Performance of the AutoTune controller for the Apache Web server under dynamic workloads
Conclusions

• Proposing an agent-based solution
  – Automating the ongoing system tuning
  – Automatically designing an appropriate tuning mechanism for the target system

• Experiments showing
  – The feedback-driven controller to be robust and adaptable to situations other than the one for which it was designed