Managing Web server performance with AutoTune agents

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

Jangwon Han
Seongwon Park

2011-12-21
Managing Web server performance with AutoTune agents

Contents

- Introduction
- Apache Web server and performance tuning
- Self-tuning AutoTune agents
- Conclusion
Managing Web server performance with AutoTune agents

Introduction

- Managing the performance of servers which have dynamic workload is challenging
  - System administrators must frequently tune servers manually to maintain optimal settings

- Autonomic controllers can reduce the work of system administrators and provide guaranteed control performance.
Introduction

In this paper

- Propose an agent-based solution
  - Automates the ongoing system tuning
  - Automatically designs an appropriate tuning mechanism for the target system

- Study the problem of controlling CPU and memory utilization of an Apache Web server using the application-level tuning parameters MaxClients and KeepAlive
Apache Web server and performance tuning

- Apache Web server
  - The most popular Web server
  - Version 1.3.x of the server on the UNIX

- Processes of Apache Web server
  - Master process
    - Monitors the health of the worker processes and manages their creation and destruction
  - A pool of worker processes
    - Communicate with Web clients and generate response
    - One worker process can handle at most one connection
Worker process connection

Worker process has three states

- **Idle**
  - No TCP connection form the client

- **Waiting**
  - After TCP connection is accepted, The worker process is waiting for a HTTP

- **Busy**
  - The worker process is busy in processing the client request
Worker process connection

- TCP connection remains open between consecutive HTTP request
  - Eliminates the overhead for setting up one connection
  - TCP connection can be terminated by client or by the master process
Performance metric

- Utilization of various resources on the server
  - Server-side metric
  - Measure CPU and memory utilization
Tuning parameter

- MaxClient and KeepAlive
  - Increase MaxClient
    - more client request
    - Increase CPU and memory utilization
  - Decrease KeepAlive
    - Worker processes is more active
    - Increase CPU utilization

- Select the tuning parameter properly to achieve the feasible CPU and memory utilization
Results of manually tuning the Apache web server

Experiment

- Keep Alive
- MaxClients
- CPU Utilization
- Memory Utilization
Effects of dynamic workloads on manually tuned Apache server

Experiment
Self-tuning AutoTune Agents (1)

- Solution to the dynamic workload
  - Server self-tuning with multiple AutoTune agents
    - Automate the entire methodology of controller design
    - perform the on-line system control
    - Implement by using Agent Building and Learning Environment (ABLE)
      - provides a comprehensive library of intelligent reasoning and learning components
      - Java based toolkit
Self-tuning AutoTune Agents (2)

- Base AutoTune agent
  - Pre-made class in ABLE
    - Controller
      - controls adaptor by setting controller parameters and save result in data repository
    - Adaptor
      - controls target system by given control parameters and save result in data repository
Managing Web server performance with AutoTune agents

Self-tuning AutoTune Agents (3)

- AutoTune agent-based self-tuning system
  - Create three agents based on Base AutoTune agent
    - modeling agent, control design agent
      - generating controller for target system
    - run-time control agent
      - For ongoing tuning
Modeling Agent (1)

- Modeling Agent
  - To gain relationship (so called system model) between the tuning parameters and performance metrics of target system
    - System designer sets experimental controller parameters
    - Controller (Exiting Signal Generator) sets workload for adaptor
    - Adaptor applies workload to apache server
    - Results are sent to control design agent
Modeling Agent (2)

- System model is shown in matrices
  - $\text{CPU}_k$: CPU utilization at the k-th time interval
  - $\text{MEM}_k$: memory utilization at the k-th time interval
  - A, B: modeling parameters
    - Identified using the least squares method
      - A way to minimize the sum of the squares of the errors when making an equation

\[
\begin{bmatrix}
\text{CPU}_{k+1} \\
\text{MEM}_{k+1}
\end{bmatrix} = A \cdot \begin{bmatrix}
\text{CPU}_k \\
\text{MEM}_k
\end{bmatrix} + B \cdot \begin{bmatrix}
\text{KeepAlive}_k \\
\text{MaxClients}_k
\end{bmatrix}
\]

Least squared method
Control Design Agent

- Design control parameters based on the system model obtained from the modeling agent
  - System designer sets design criteria
  - Controller sets controller design to
  - Adaptor applies controller parameters
  - Result is sent to run-time control agent
Control Design Agent (2)

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

$$J(K_p, K_i) = \sum_{k=1}^{\infty} \begin{bmatrix} e_{CPU,k} & e_{MEM,k} & \nu_{CPU,k} & \nu_{MEM,k} \end{bmatrix} \cdot Q \cdot \begin{bmatrix} e_{CPU,k} \\ e_{MEM,k} \\ \nu_{CPU,k} \\ \nu_{MEM,k} \end{bmatrix}$$

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

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

$$Q = \text{diag}(q_1, q_2, q_3, q_4)$$
Run-time Control Agent (1)

- Run-time control agent
  - Makes control decisions based on feedback mechanism
    - Controller parameters from control design agent are applied
    - System administrator sets desired utilization
    - Feedback controller sets controller parameter to adaptor
    - Adaptor applies changes to apache web server
    - Results are used for feedback controller
Self-tuning AutoTune Agents

Run-time Control Agent (2)

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

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

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

Increase KeepAlive
Decrease MaxClients

e_{CPU}: + 0.2
\quad e_{Mem}: + 0.3

e_{CPU}: + 0.05
\quad e_{Mem}: + 0.02

e_{CPU}: + 0.01
\quad e_{Mem}: + 0.00
Self-tuning AutoTune Agents

Experiment Environment

- Environment: Linux 2.2.16, Apache 1.3.19
  - Workload generator
  - WAGON
    - Web trAffic GeneratOr and beNchmark
  - Web Stone
    - Web site file access distribution:

- Experiment 1
  - Static workload

- Experiment 2
  - Static workload
  - Dynamic workload
Self-tuning AutoTune Agents

Experiment 1

Results with Self-tuning AutoTune Agents
Static workloads initialized with KeepAlive = 60, MaxClients = 600
Self-tuning AutoTune Agents

Experiment 2

Results with Self-tuning AutoTune Agents
Static workloads & Dynamic workloads (starting from 20th interval) initialized with KeepAlive = 5, MaxClients = 390 aprox.
Conclusion

- Tuning server manually is annoying
  - Servers which have dynamic workloads should be frequently tuned to be maintained, optimized
  - Tuning mechanism should be re-designed when target system is changed

- AutoTune Agent-based self-tuning is a way to solve these problem
  - Automating the ongoing system tuning
  - Automatically designing an appropriate tuning mechanism for the target system