Dynamic Provisioning of Multi-tier Internet Applications

Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, and Pawan Goyal

Presentation: Jeff Pound
Outline

- The problem
- Baseline approaches
- Dynamic provisioning of n-tier systems
- Experimental evaluation
- Discussion
The Problem

- Dynamically provision servers among n-tier applications to meet response time requirements.
Baseline Approaches

- Idea: Use single tier provisioning approaches on the n-tier system
  - Scale individual tiers separately
  - Treat all n-tiers as a black box and scale together
Baseline: scale individual tiers

- Identify bottleneck tier and increase provisions to that tier
Baseline: scale individual tiers

- Identify bottleneck tier and increase provisions to that tier
Baseline: Black-box

- Treat the entire system as a black-box and provision when end-to-end response time is below target
  - How many servers do we add?
  - At which tier(s) do we add servers?
  - Not knowing which tier is the bottleneck can cause problems, what if the bottleneck tier isn’t replicable?
Baseline: Black-box
Baseline: Black-box

- $\lambda = 3.5$ req/s
- $\lambda = 7$ req/s
- $\lambda = 3.5$ req/s
- Cap = 40 req/s
- Cap = 30 req/s
- Tier 1
- Tier 2
- Tier 3 drops 3.5 req/s
- 10.5 req/s
Provisioning for n-tiers

- Collectively model the effects of all tiers
- 3 major components:
  - Admission Control
  - Predictive Provisioning
  - Reactive Provisioning
Provisioning for n-tiers

- One time admission decision per session
- Response time guarantees (SLA)
- Tier-aware
  - Number of tiers, current capacities, constraints on replication
  - (Assume load balancer exists per tier)
Design

- Nucleus (monitor)
- Capsule (Application)
- Sentry (admission controller)
- Control Plane (provisions servers)
Provisioning Algorithm

- **Input**
  - Incoming request rate to the system
  - Service demand of request

- **Output**
  - Number of servers needed at each tier to handle the total demand on the system

- **2 questions to answer**
  - *How much* to provision and *when*?
How much?

- First, determine the capacity of an individual server (request rate)
  - Treat each server as a G/G/1 queue

\[
\lambda_i \geq \left[ s_i + \frac{\sigma_a^2 + \sigma_b^2}{2 \cdot (d_i - s_i)} \right]^{-1}
\]
How much?

- First, determine the capacity of an individual server (request rate)
  - Treat each server as a G/G/1 queue

\[ \lambda_i \geq \left[ s_i + \frac{\sigma_a^2 + \sigma_b^2}{2 \cdot (d_i - s_i)} \right]^{-1} \]

- Then, compute the number of servers need at each tier (in a single step)

\[ \eta_i = \left[ \frac{\beta_i \lambda \tau}{\lambda_i Z} \right] \]
How much?

\[ \eta_i = \begin{bmatrix} \beta_i \lambda \tau \\ \lambda_i Z \end{bmatrix} \]

\( \beta_i \) is a tier specific constant

\( \lambda_i \) is the request rate at tier \( i \)

\[ \lambda \tau \] \( \frac{1}{Z} \)

is the request rate to the system

In practice, we can estimate \( \beta \) by the ratio of requests at the tier and the request admitted to the system over the last time period.
Available Servers

- Servers are provisioned to applications from the *free pool*
- If there are not enough servers in the free pool
  - Applications that *benefit* the most are given priority
  - Benefit determined by a utility function
- If the free pool is empty, servers can be de-allocated from over-provisioned (under-loaded) applications
  - Under-loaded if observed / predicted request rates are below a threshold
When?

- Predictive
  - Provision on a time scale of hours and days

- Reactive
  - Provision on a time scale of minutes
Predictive Provisioning

- Provision in advance for predicted workloads
  - Based on previous workloads over the past hours or days
- Provision for the peak workload seen in the given time interval
  - Use the tail of the arrival rate distribution
  - Correct prediction based on previous errors
Predictive Provisioning

Hourly session arrival rate

- Sunday
- Monday
- Tuesday
- Today

**arrivals observed during noon–1PM**

- Histogram of number of arrivals
  - Probability distribution
  - High percentile

- Recent trends
  - Predicted demand
  - Correction

Prediction for noon–1PM today
Predictive Provisioning

\[ \lambda_{pred}(t) = \lambda_{pred}(t) + \sum_{i=t-h}^{t-1} \frac{\max(0, \lambda_{obs}(i) - \lambda_{pred}(i))}{h} \]

Prediction from past days (tail of distribution)

Error correction
Reactive Provisioning

- Trigger reactive provisioning if...
  - error ratio (obs / pred) differs by more than a given threshold, or
  - request drop rate (at the admission controller) is larger than a given threshold.

- (Use same equations as predictive provisioning)

*request drop rate?
Admission Control

- After provisioning is done
  - The max workload that has been provisioned for is reported to the Sentry
  - Sentry denies admission to new sessions if the arrival rate is above the specified max
Experimental Evaluation

- Experiments suggest that *predictive provisioning*, *reactive provisioning*, and *admission control* are both *necessary* and *sufficient* mechanisms for dynamic provisioning of n-tier systems.
Discussion

- How do we distribute target response time \( (R) \) over per-tier response times \( d_i \)?
- What about utilization? (Always over-provisioning)
- Why G/G/1 queuing model (M/GI/1 or M/G/1?)
- Stateful sessions need to be serviced by the same server at each tier.

Deployment issues:
- Application logs and server logs need to be processed on-line.
- Need (“simple”) modifications to web and application servers (Apache and Tomcat).
- Have to determine threshold values.
- On-line monitoring of physical resources.