60017 PERFORMANCE ENGINEERING

Bottlenecks in Distributed Systems

Last lecture

- Workloads in computer systems
  - Log files
  - User behavior models

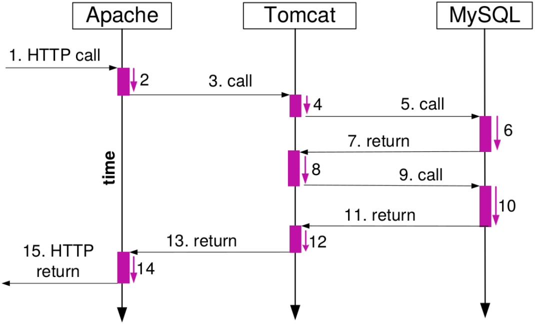
This lecture

- Service demand
- Utilization law
- Bottleneck analysis

Overview

- Typical questions involving computational resources include:
  - Bottleneck analysis: Which resources limit the scalability of a given distributed application?
  - What-if analysis: How will changes in user request rates or resource speed impact performance?
- Performance of individual resources (e.g., CPUs) affected by:
  - Arrival rate of requests
  - Service time of the request at the resource
  - Contention at the resource (e.g., jobs contend the CPU)
  - ...
- How to predict and analyse resource utilizations?

Example: resources in a three-tier application



Source: Wang *et al*, CLOUD 2013

- Serving a web page request typically involves multiple servers
- Each server (e.g., MySQL) is a resource
- A request can visit multiple times a resource, receiving a variable service time at each visit, until completing.
- We here focus on the percentage of time each resource is busy and do not attempt to quantify overheads due to contention.

Service classes

- We assume that the IT system:
  - offers C types of services (service classes), *e.g.*, C web pages.
  - processes requests using M resources, *e.g.*, M servers.
- In order to complete, a request of class c
  - makes on average k_{ic} calls to resource i
  - requires a mean service time of S_{ic} seconds for each call to i
- The demand $D_{ic} = k_{ic}S_{ic}$ is the service time accumulated by a class-c request through its visits to resource i.
- In the presence of contention/queueing, the demand represents only the effective processing time received at the resource, neglecting overheads due to other jobs.

Operational analysis

- Let us now investigate the relationship between demands and resource utilization.
- Suppose to monitor a resource for an observation period of T seconds collecting:
  - A_c: total number of arrived requests of class c
  - B_{ic}: total time resource i is busy processing class-c requests
- Averaging over the observation period we obtain:
  - $\lambda_c = A_c/T$: average arrival rate of requests of class c
  - $U_{ic} = B_{ic}/T$: utilization of resource i due to class-c requests

Operational analysis

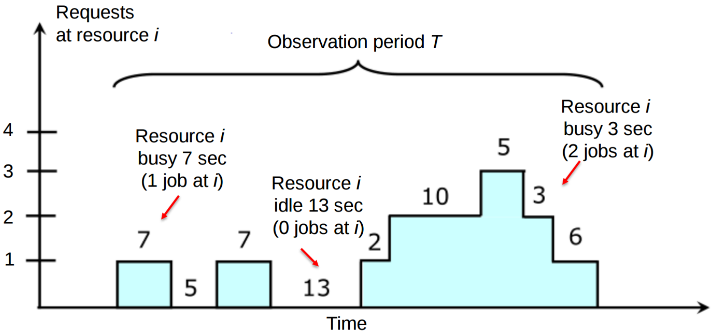
- If requests never fail and the pending requests are always bounded in number, then λ_c is also equal to the class-c throughput X_c, as T grows large because

$$\lambda_c = \lim_{T\to\infty} \frac{A_c}{T} = \lim_{T\to\infty} \frac{\text{completed}(c) + \text{pending}(c)}{T} = \lim_{T\to\infty} \frac{\text{completed}(c)}{T} = X_c$$

  since the number of pending requests remains finite, if the system is stable.
- An unstable system cannot cope with the rate of arrivals, so that eventually its backlog grows unbounded.
- Throughout, we assume that the system is stable.

Example: busy time, arrivals, completions

Assume a single workload class ($C = 1$). We omit class indices.



$$B_{i1} = 7 + 7 + 2 + 10 + 5 + 3 + 6 = 40s$$

$$U_i = B_{i1}/T = 40/58 = 0.689 \ (68.9\%)$$

$$\lambda = \text{upward transitions}/T = 5/58 \ tps$$

$$X = \text{downward transitions}/T = 5/58 \ tps = \lambda$$

Utilization law

- We now observe that

$$D_{ic} = B_{ic}/A_c$$

  since B_{ic}/A_c averages the total processing time of class-c requests at i (which is the definition of D_{ic}).

- What is the relationship between utilization and demands?

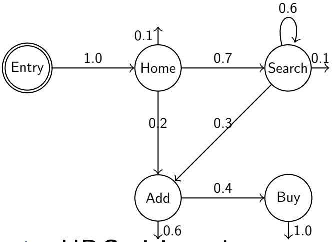
$$U_{ic} = \frac{B_{ic}}{T} = \frac{A_c}{T}\frac{B_{ic}}{A_c} = \lambda_c D_{ic}$$

- For a system with C workload classes, the total resource utilization U_i is thus given by the utilization law

$$U_i = \sum_{c=1}^{C} U_{ic} = \sum_{c=1}^{C} \lambda_c D_{ic} = \sum_{c=1}^{C} X_c D_{ic}$$

Example: utilization from UBGs

- Users arrive at rate $\lambda = 1$ sessions/sec



D_{ic} [sec]	Home	Search	Add	Buy
Web	0.1	0.2	0.4	0.1
Tomcat	0.0	0.2	0.3	0.7
MySQL	0.0	0.8	0.2	0.1

- UBG visit ratios:

$$V_H = 1,\ V_A = 0.725,\ V_S = 1.75,\ V_B = 0.29$$

- Request rates:

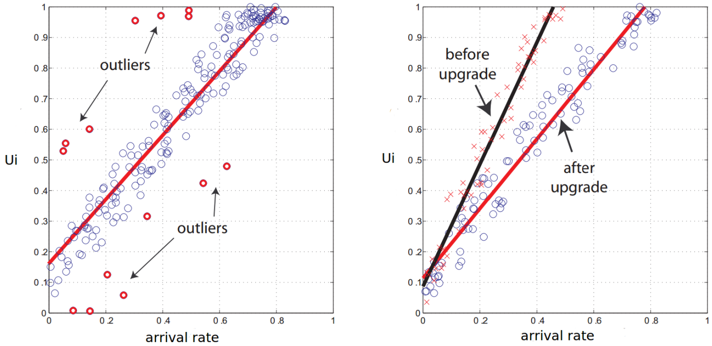
$$\lambda_H = \lambda V_H,\ \ \lambda_A = \lambda V_A,\ \ \lambda_S = \lambda V_S,\ \ \lambda_B = \lambda V_B$$

- Utilization:

$$U_W = \lambda_H D_{WH} + \lambda_A D_{WA} + \lambda_S D_{WS} + \lambda_B D_{WB} = 0.974 \ \ (97.4\%)$$

Demand estimation from utilization samples

- How to estimate demand values in practice?
- The utilization law describes a hyperplane with slopes D_{ic}.



- Multivariate linear regression fits hyperplanes to samples of U_i and λ_c (or X_c), $\forall c$, returning the demands D_{ic}.
- The estimated demand values are hardware dependent and in general change after hardware upgrades.

Bottleneck analysis

- Demands can be used to determine the resources that limit scalability, called the bottlenecks.
- The bottlenecks are oversubscribed resources, which struggle to complete the backlog of pending requests.
- Bottlenecks often run near 100% utilization (saturation), thus over time they tend to pile up a large backlog of requests.
- Bottleneck analysis wishes to answer these questions:
  - As we increase the arrival rates of requests, which resource(s) will saturate first?
  - What is the maximum arrival rate that the system can sustain under a given request mix?

Bottleneck analysis

- For a system with M resources, we can describe resource usage in terms of the system of linear equations

$$U_i = \sum_{c=1}^{C} \lambda_c D_{ic} \qquad i = 1, \ldots, M$$

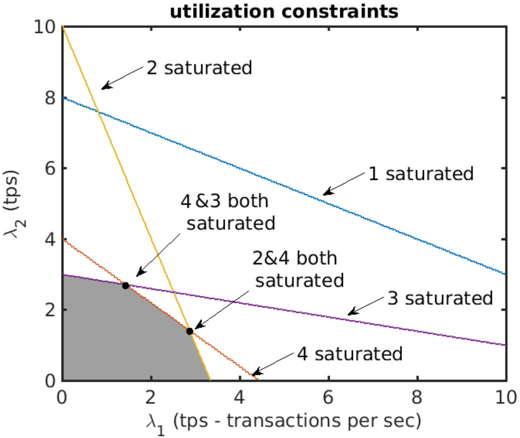
  where we require $U_i \leq 1$ for each resource.
- Resource j can saturate if and only if there exists a combination of arrival rates $(\lambda_1, \ldots, \lambda_C)$ such that $U_j = 1$.
- Wether a resource can saturate may be verified using the following linear program (LP) defined over the variables λ_c

$$\begin{aligned} U_j^{\text{max}} = \text{maximize} \quad & \textstyle\sum_{c=1}^{C} \lambda_c D_{jc} \\ \text{subject to} \quad & \textstyle\sum_{c=1}^{C} \lambda_c D_{ic} \leq 1, \quad i = 1, ..., M \\ & \lambda_c \geq 0, \qquad c = 1, ..., C \end{aligned}$$

  If the LP has optimal value $U_j^{\text{max}} = 1$, then j can saturate.

Example: sustainable arrival rates

Assume $C = 2$ workload classes and $M = 4$ resources.



Each line represents the boundary of $U_i \leq 1$, for given i and D_{ic}.

Example: sustainable arrival rates

In the example:

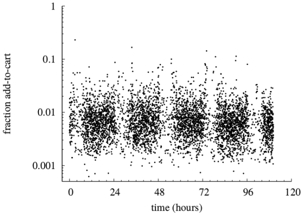
- Each point (λ_1, λ_2) is a possible arrival rate to the system.
- The shaded region indicates the sustainable arrival rates.
- Server 1 can never become a bottleneck. Never upgrade it!
- Server 2 is the class-1 bottleneck (largest demand in class 1).
- Server 3 is the class-2 bottleneck (largest demand in class 2).
- Server 4 is a bottleneck only for some arrival rates.
- Servers 2,3,4 are thus potential bottlenecks for this system.
- Some mixes lead multiple resources to become bottlenecks simultaneously, e.g., $2 + 3$ or $2 + 4$.

Bottleneck mitigation

- If the system has a single potential bottleneck, we can remove the bottleneck by upgrading its hardware or scale up the VM.
- However, it is important to realise that arrival rates change over time and so the active bottlenecks.
  - Arrival rates change visibly through the day, week and month.
  - Recent studies found that the class mix also varies frequently.
- Therefore IT systems may display different bottlenecks over time, requiring adaptive resource management to cope with transient need of capacity at a certain resource.
  - Since cloud computing allows to acquire and release resources via APIs, it facilitates adaptive resource management.

Example: time-varying transaction mixes

Mix of requests in execution in a real e-commerce website:

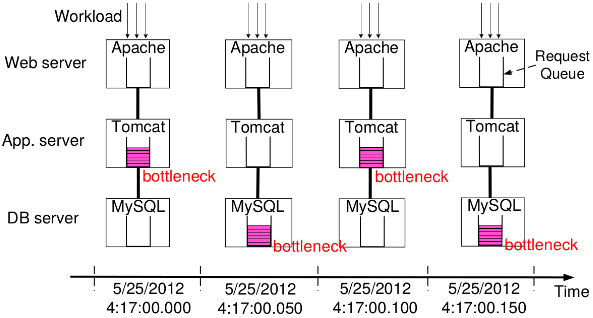


Source: Stewart *et al*, Eurosys 2007

- Each point represent the fraction of requests of the two popular transaction types seen in the website during a 5-minute monitoring interval
- If the two requests stress different resources (e.g., Tomcat, MySQL), the resource usage of these resources will also fluctuate over time

Bottleneck switches

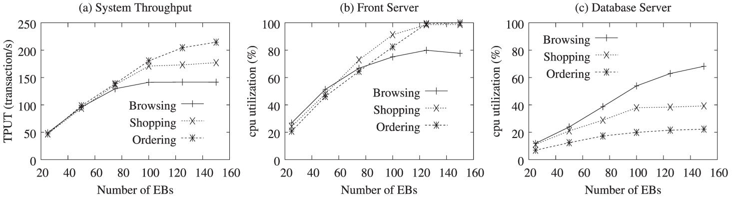
- If there are multiple potential bottlenecks, rapid changes in the transaction mix can lead to observe bottleneck switches.
- Bottlenecks can migrate between resources very rapidly, even within milliseconds (millibottlenecks).
- If bottleneck switches occur at ms scale, they can be invisible to monitoring, which aggregates data at s or min resolution.



Sources: Wang *et al.*, CLOUD 2013

Diagnosing bottleneck switches

- Rapid bottleneck switches typically degrade performance.
  - Transactions more likely to find queues along the flow.
- A symptom that this is happening is that throughput grows slowly, even if the system is lightly utilised.
- Example: TPC-W's browsing mix illustrates rapid bottleneck migration between application server and DB.



Sources: Casale *et al.*, IEEE TSE 2012