60017 PERFORMANCE ENGINEERING

Resource scaling

This lecture

- Resource scaling
  - Vertical scaling
  - Horizontal scaling
- Autoscaling basics

Vertical and horizontal scaling

- IT systems use multiple tiers consisting of several nodes.
- We focus on IT systems on IaaS clouds, where nodes are virtual machines.
- Two main mechanisms exist to increase application capacity:
  - Vertical scaling (VS): increase or upgrade the resources within existing nodes (cores, memory, disk, ...).
  - Horizontal scaling (HS): add new nodes to one or more tiers.
- Common scaling terminology:
  - When we use VS, we scale up the application.
  - When we use HS, we scale out the application.
  - When we remove nodes, we scale down the application.
- In essence, VS throws more hardware at the problem, while HS extends the application architecture.

Vertical scaling (VS)

- VS increases the number of cores, memory, I/O devices and in general resources available to a VM.
- VS has the virtue of simplicity, as it does not involve architectural changes to the scaled application.
- With VS, the node (VM) needs to be restarted, implying some downtime for that node.
  - Most operating systems cannot cope with a dynamic upgrade to the number of cores or to the memory.
  - Restart makes VS a risky practice, what if the restarted VM fails on startup?
- Another limitation is that the application may not be able to exploit the new resources, *e.g.*, due to lack of internal parallelism.

Amdhal's law

- T_n: execution time for a job running on n cores
- T_1: execution time for a job running on a single core
- S_n: speedup, i.e., improvement due to parallelization
- Amdhal's law (1967) characterises to what extent systems can exploit parallelization (*e.g.*, multi-core).
- The law states that the speedup is limited by the fraction p of the job's execution that can be parallelised

$$S_n = \frac{T_1}{T_n} \approx \frac{T_1}{\underbrace{(1-p)T_1}_{\text{serial part}} + \underbrace{pT_1/n}_{\text{parallel part}}} = \frac{1}{(1-p) + p/n}$$

- Various refinements exist, *e.g.*, to account for communication overheads between parallel units.

Horizontal scaling (HS)

- HS adds capacity by adding nodes to the system tiers.
- Normally, HS does not result in downtime because:
  - Nodes within a tier do not interact with each other
  - Communication between tiers is handled by load balancers and message queues that can route requests to active nodes.
- HS changes the application topology. This requires some time for the application to adapt:
  - the load balancer needs to discover and start dispatching load to the new nodes.
  - the new nodes may take time to fill-up local caches, therefore performance will take some time to stabilize.
- While the application adapts, capacity needs may be fulfilled using throttling.

Horizontal scaling (HS)

- Tiers in cloud applications are often composed by homogeneous nodes.
- Each node added by HS typically adds the same amount and type of memory and cores as in the existing nodes for that tier.
- Homogeneity is a helpful simplification, otherwise with heterogeneous nodes the load balancing would be complex.
  - Assigning load balancing weights to a set of heterogeneous nodes is a difficult problem
  - The number of requests queueing in each node may be very different and unknown to the load balancer.

Horizontal scaling (HS)

- With homogeneous nodes, we can use round-robin (RR) to load balance within a tier.
  - RR dispatches requests in turn to each node in the target tier.
  - RR produces an equal arrival rate to each target node.
  - RR makes inter-arrival times between service calls more predictable, simplifying resource management.
  - RR is stateless, hence no need to monitor the target nodes.

Cloud native applications

- Not all application architectures can support HS, *e.g.*,
  - a centralised component places an upper limit to the maximum number of concurrent requests.
  - the bottleneck resources are in a tier that cannot scale, making scaling ineffective to increase performance.
- Applications designed for the cloud are called cloud native applications. They typically support both HS and VS.
- A cloud native application relies on stateless autonomous compute nodes.
  - This makes scaling up 200 nodes as easy as scaling up 2.
  - The set of resources that need to be scaled together is called scale unit.

Cloud native applications

- Stateless nodes do not imply a stateless application. Several technique exist to maintain state within a user session:
  - use a cookie (for web servers only)
  - provide state information as a service call parameter.
  - use a sticky session.
  - retrieve state from external storage
    - *e.g.*, memcache, cloud storage, ...

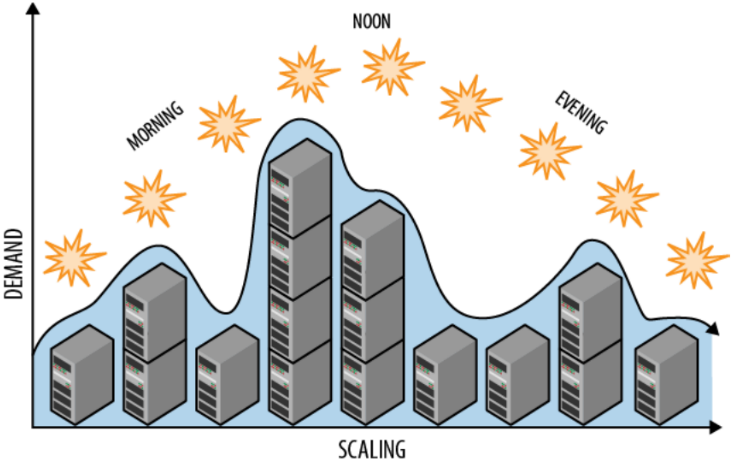
Autoscaling

- Scaling is needed to address:
  - Resource over-provisioning, which results in idle instances, incurring unnecessary costs
  - Resource under-provisioning, which hurts performance and leads to SLA violations
- However, the application workload changes over time, requiring continuous adjustment to the scaling decisions.
- Autoscaling algorithms automatically scale resources according to the workload demand.

Autoscaling setup

- Autoscaling algorithms used in public clouds are either schedule-based or rule-based.
- Schedule-based autoscaling takes into account the cyclical pattern of the daily workload.
  - Scaling actions are pre-configured based on the time of the day.
  - Schedules are manually defined and inherently immutable.
- Rule-based autoscaling relies on conditional actions triggered by a target variable, *e.g.*, CPU load exceeding a threshold.

Example: schedule-based autoscaling



Source: Wilder - Cloud Architecture Patterns - O'Reilly.

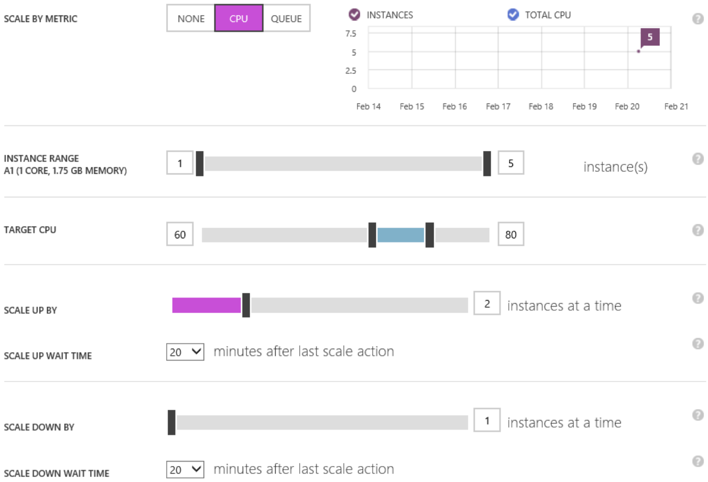
Rule-based autoscaling

- Widespread approach, based on static threshold rules.
- Let V the target variable, e.g. VM CPU utilization.
- The typical structure of a static threshold rule is:

if $V > V_{up}$ for T_{up} seconds **then**
  scale out by adding N_{up} nodes
  do nothing for S_{up} seconds
else if $V < V_{dn}$ for T_{dn} seconds **then**
  scale down by removing N_{dn} nodes
  do nothing for S_{dn} seconds
end if

where $(V_{up}, T_{up}, N_{up}, S_{up})$ and $(V_{dn}, T_{dn}, N_{dn}, S_{dn})$ are user-specified scale-up and scale-down parameters.

Example: rule-based autoscaling in Azure

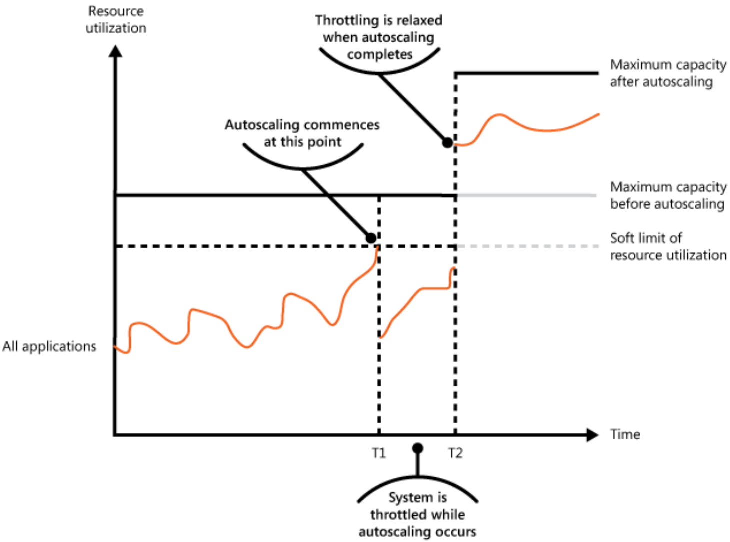


Source: Microsoft.

Resource throttling

- Static thresholds are a form of reactive scaling, where scaling actions react to changes in the target variable.
- Typically, scaling actions kick-in minutes after the workload demand surges.
  - How should we handle the shortage of capacity while we wait for scaling to happen?
- An approach consists in using throttling until the scale out operation has completed.
  - Throttling limits the rate at which a resource can be accessed.
  - Throttling can be implemented using rate limiters that distribute permits to use the resource at a bounded rate.

Example: Reactive autoscaling and throttling



Source: Microsoft.

Proactive autoscaling

- Research methods for auto-scaling algorithms are a combination of proactive and reactive techniques.
- A proactive method attempts to anticipate the workload demand, using for example time series forecasting.
- Techniques to decide how many resources to scale in forecasted scenarios are based for example on optimization, queueing theory or machine learning.