Model-free and Model-based Auto-Scaling Techniques for Distributed Applications

Introduction to Auto-Scaling

Gabriele Russo Russo

University of Rome Tor Vergata, Italy

June 2023 – Roma Tre University

Info

Gabriele Russo Russo
Research Fellow at *University of Rome Tor Vergata*

Main research interests:

- Cloud & Edge Computing
- Run-time Adaptation of Distributed Applications
- Function-as-a-Service Systems

Info (2)

Course composed of 4 lectures:

- Mon, June 5 (14:00-16:00)
- Wed, June 7 (14:00-16:00)
- Mon, June 10 (14:00-16:00)
- Wed, June 12 (14:00-18:00)

Slides will be available on my website

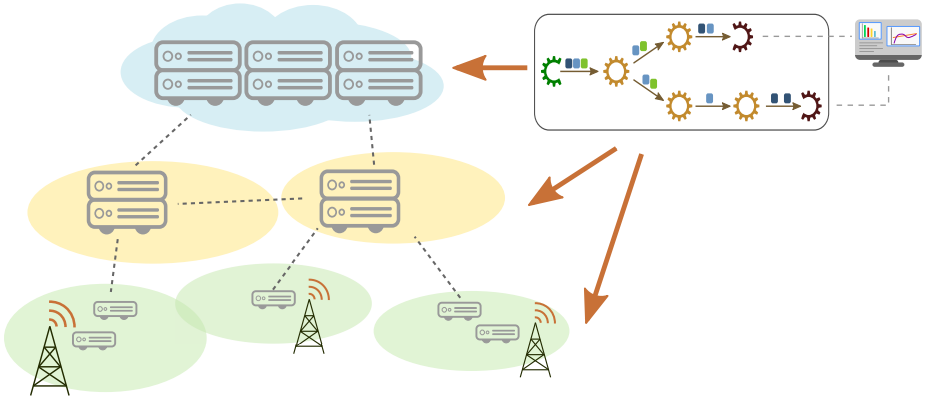
Q&A: interrupt me at any time for questions!

Agenda

- Application auto-scaling
  - Introduction
  - Key challenges
  - Architectures
- Model-based and model-free auto-scaling policies
  - Threshold-based heuristics
  - Reinforcement learning

Scenario

Distributed applications in heterogeneous computing environments with Quality-of-Service requirements



Cloud Computing

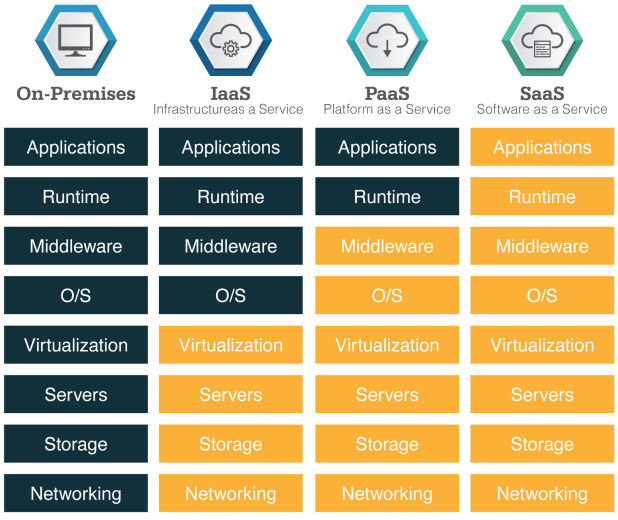
"[...] enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort [...]"
NIST definition

- Global spending on cloud computing services to reach $1.3 trillion by 2025 (International Data Corporation)
- Enterprise IT spending for the shift to the cloud to reach $1.8 trillion by 2025 (Gartner)

Cloud Computing (2)

- On-demand, self-service
- Broad network access
- Resource pooling and multi-tenancy
- Rapid (and automatic) provisioning
- Pay-as-you-go

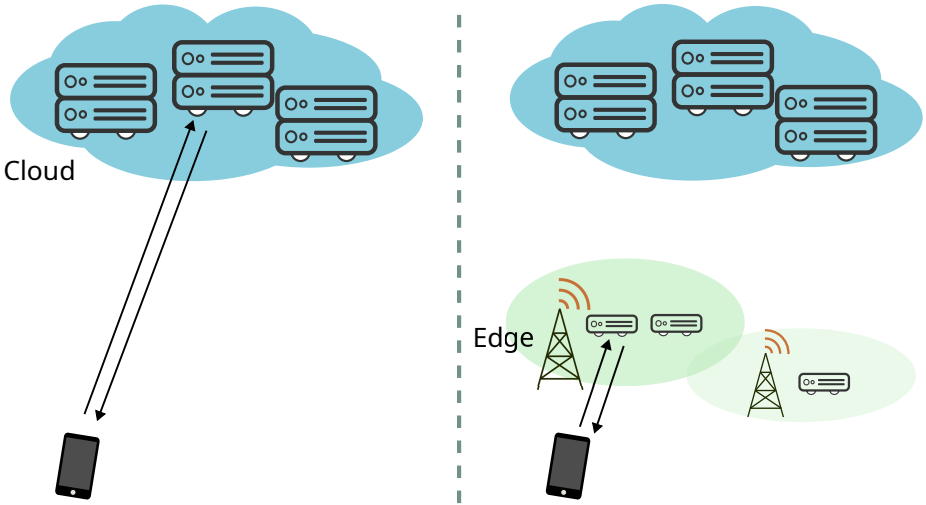
Cloud Service Models



Cloud Deployment Models

- Public: infrastructure provisioned for open use by the general public
  - Amazon Web Services
  - Google Cloud Platform
  - Microsoft Azure
  - …
- Private: infrastructure provisioned for exclusive use by a single organization comprising multiple consumers
- Hybrid: composition of two or more distinct cloud infrastructures

From Cloud to Edge Computing



From Cloud to Edge Computing (2)

Advantages:

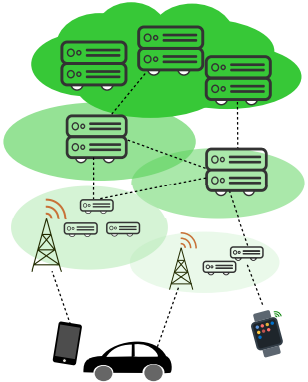
- Reduced network latency
- No need to transfer data to cloud data centers

But...

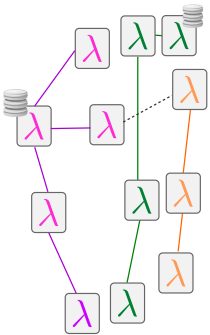
- Reduced computational capacity
- Heterogeneous computing resources
- Limited/unstable network connectivity

Compute Continuum

- We are moving towards a continuum of computational resources: from IoT devices to cloud servers
  - a.k.a. “Cloud-to-Thing Continuum”
- Resources are even more heterogeneous and ubiquitous



Computing Infrastructure



Applications

Challenges: Deployment

Deploying applications in these environments is challenging

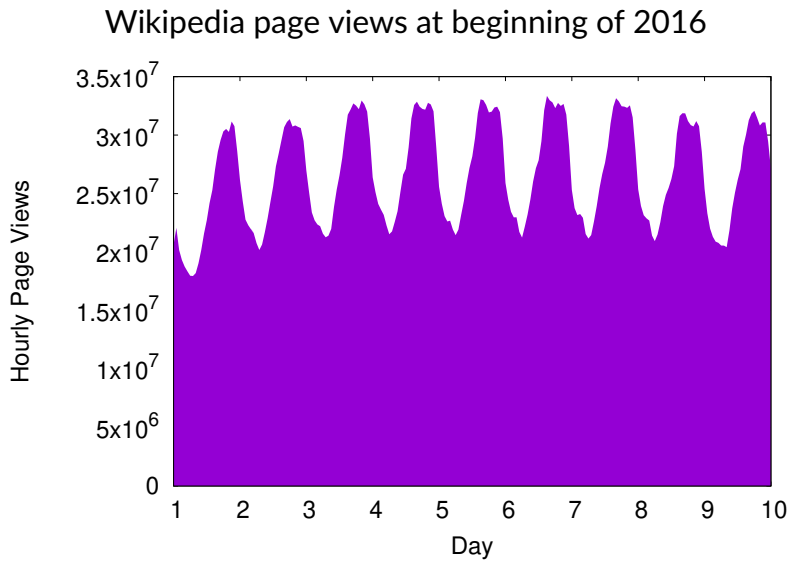
- Where to deploy each application component?
  - Hardware requirements (e.g., GPU)
  - Latency
  - Security and privacy issues
  - Energy
- How to provision enough resources to each component (e.g., memory, CPU shares)?
  - Traditional solution: over-provisioning

Challenges at Run-Time

- Many applications are long-running
- Working conditions likely change during execution!

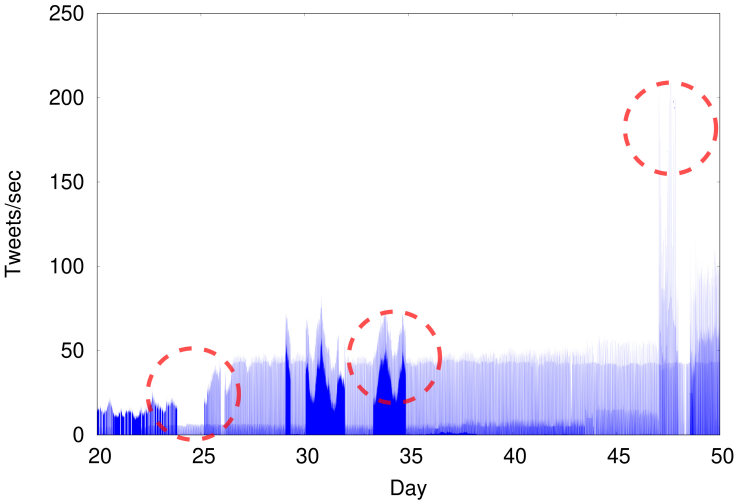
- Workloads
- Mobile user location
- Infrastructure load
  - Both for computing hosts and network!
- Resource prices
- …

Example: Workload Variability



Example: Workload Variability (2)

Tweets about “COVID” at beginning of 2020



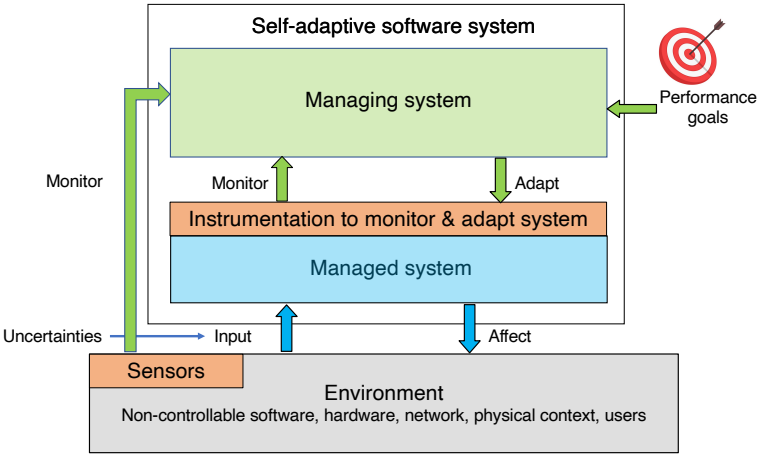
Run-Time Adaptation

- How can applications deal with changes at run-time?
- Applications need to adapt to changing working conditions
  - Migrating components
  - Provisioning more/less resources
  - …

Self-Adaptation

- Scale of modern applications and infrastructures
- Cannot rely on humans alone!
- Applications must be able to self-adapt!

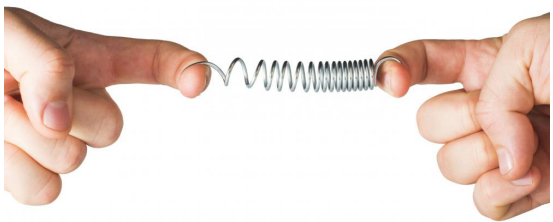
Self-Adaptive System



Reference book: “An Introduction to Self-Adaptive Systems” [Weyns 2020]

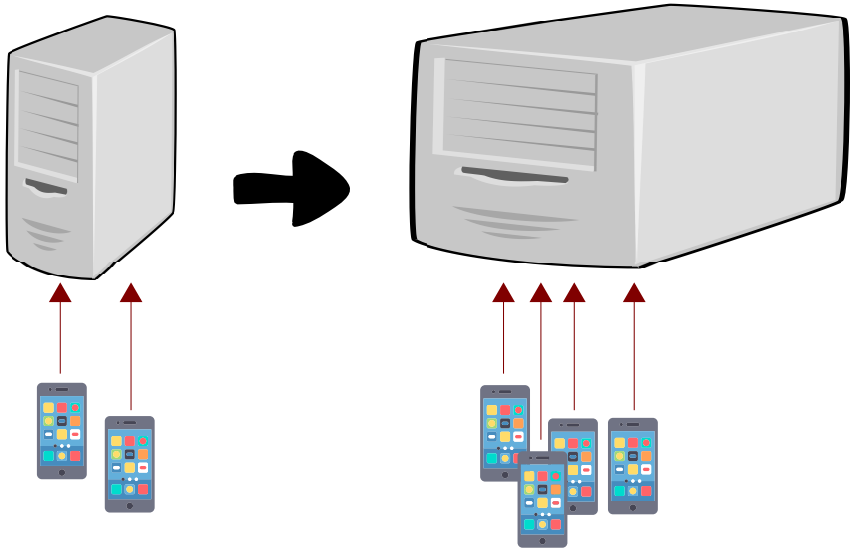
Elasticity

- We focus on a specific aspect of application adaptation: **elasticity**
- Essential feature of modern applications and infrastructures



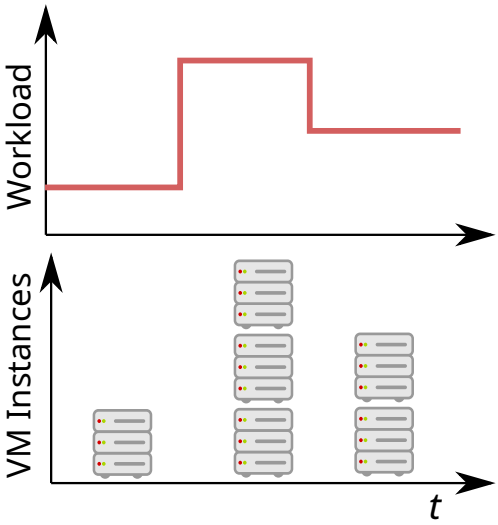
In physics: a material property capturing the capability of returning to its original state after a deformation

Elastic Computing Systems?



Example: Cloud VMs

- Consider a simple web application
- We can acquire virtual machines (VMs) from a cloud provider to deploy the application server(s)
- We can **elastically** acquire/release VMs at any time based on current demand
- e.g., “whenever the average CPU utilization of the VMs exceeds 70%, create a new VM”



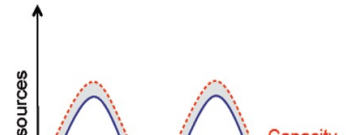
Elasticity: Definition

Herbst, Kounev, and Reussner 2013

Degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible.

Herbst, Kounev, and Reussner 2013

Degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible.



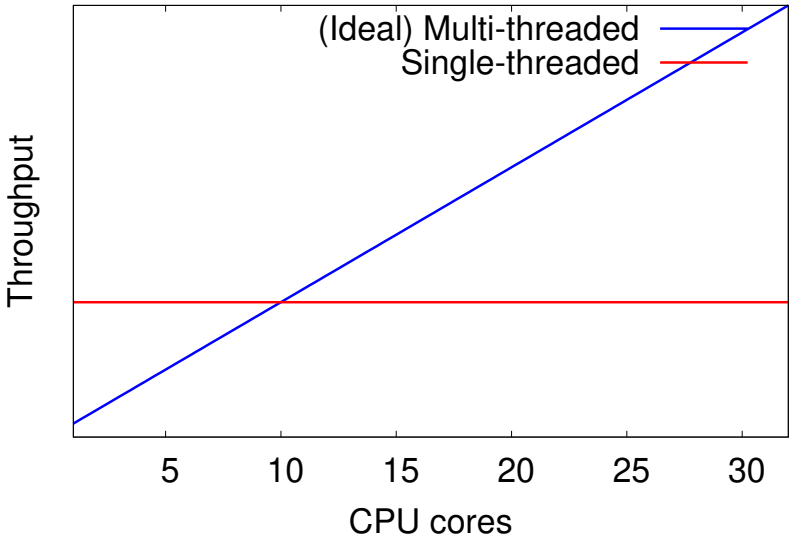
Scalability

- We are often concerned with the scalability of software systems

Scalability

Ability of a system to sustain increasing workloads with adequate performance provided that hardware resources are added

Scalability: Example



Scalability vs. Elasticity

- Without a proper adaptation process, a scalable system cannot behave in an elastic manner
- It does not make sense to reason about elasticity of a system that is not scalable!
- We need both!

Auto-Scaling vs. Elasticity

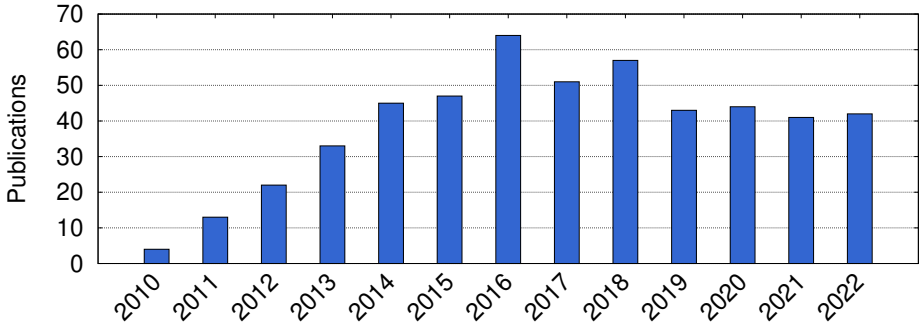
- Most of the time (including in these lectures!) the terms are used interchangeably
- To be more precise, elasticity refers to a system property that we obtain thanks to proper auto-scaling
- According to [Al-Dhuraibi et al. 2018]:

Auto-scaling = Scalability + Automation

Elasticity = Auto-scaling + Optimization

Still a Hot Topic?

- Publications retrieved from DBLP searching for “auto-scaling”, “auto-scaler”, “cloud elasticity” in the title
- So, a lot of publications not included!



Auto-Scaling: Key Issues

5 W's + 1 H

- Auto-scaling solutions differ in many aspects!
- We will look at the most relevant features and issues
- We use the 5 W's + 1 H approach
  - What
  - Why
  - When
  - Who
  - Where
  - How

What - Scaling Mechanisms

Horizontal scaling

- Adding/removing instances of computing resources (scaling out/in)
  - Virtual machines
  - Containers
  - Threads

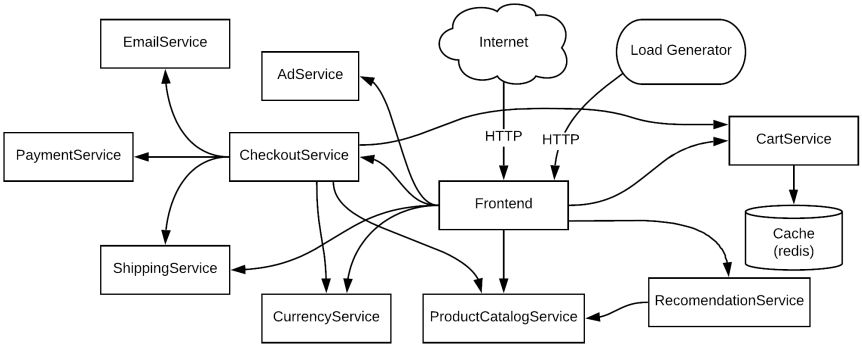
Vertical scaling

- Increasing/decreasing characteristics of computing resources (scaling up/down)
  - Memory
  - CPU shares
  - Network bandwidth

Example: AWS Auto Scaling Group

- Recall the example with the web application deployed in the Cloud
- If we are using Amazon Web Services (AWS), we can leverage the Auto Scaling Group service
- We first need to define a Launch Configuration
  - i.e., specification of how new VMs should be started (OS image, VM type, …)
- We create an Auto Scaling Group based on the Launch Configuration
- We specify the desired policy for the group

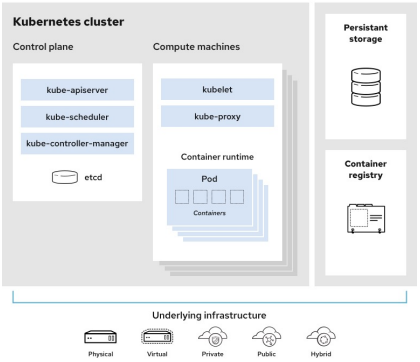
Example: Microservice-based Application

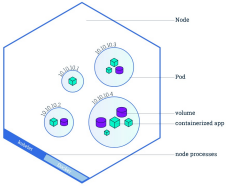


e.g., Google Online Boutique

Example: Microservices in Kubernetes

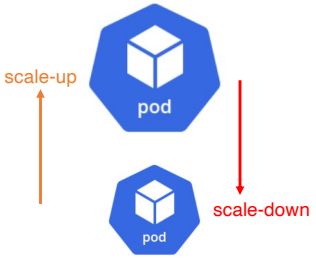
- Container orchestration platform for automating deployment, scaling, and management of containerized applications
  - Pod: group of one or more application containers ≈ 1 microservice
  - Multiple pods can run on a worker node





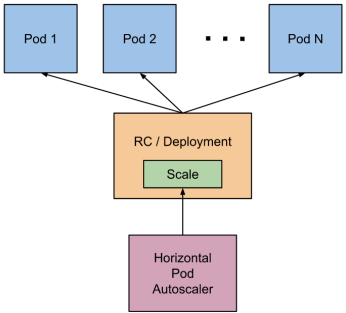
Example: Auto-Scaling in Kubernetes

- Auto-scaling at node and pod granularity
- Cluster Autoscaler: adjusts size of Kubernetes cluster
- Vertical Pod Autoscaler (VPA): scales amount of pod resources (CPU, memory)
  - Based on historical resource usage of pods [Wang, Ferlin, and Chiesa 2021]
  - Requires restarting pods



Example: Auto-Scaling in Kubernetes

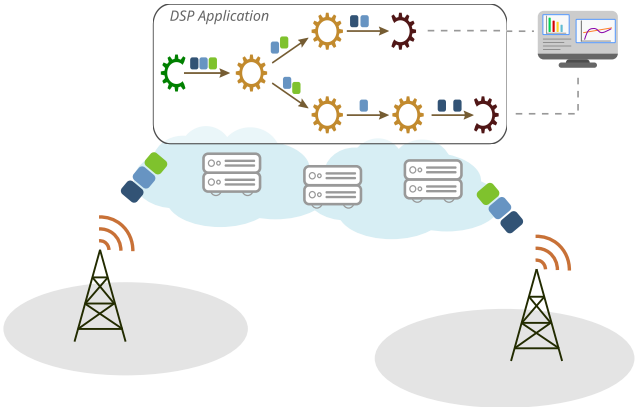
- Horizontal Pod Autoscaler (HPA): scales number of pods [Kubernetes 2021]
  - Based on observed CPU utilization (or some other application-provided metrics)
  - Creates new pods without affecting existing ones



Example: DVFS

- Dynamic Voltage and Frequency Scaling enables significant power savings in modern CPUs
- Voltage and Frequency of CPU cores adjusted on-the-fly
- An example of vertical scaling
- Usually managed at hardware/OS level
  - e.g., frequency scaled based on current workload
- Some auto-scaling solutions rely on this mechanism (e.g., Arroba et al. 2015)

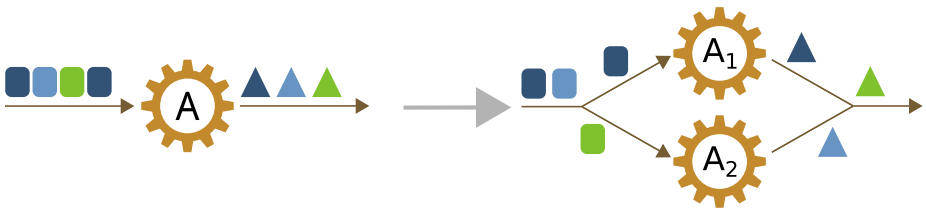
Example: Data Stream Processing



- Data stream processing (DSP): processing high-volume data flows in (near) real-time
- Popular frameworks: Apache Flink, Spark Streaming, …

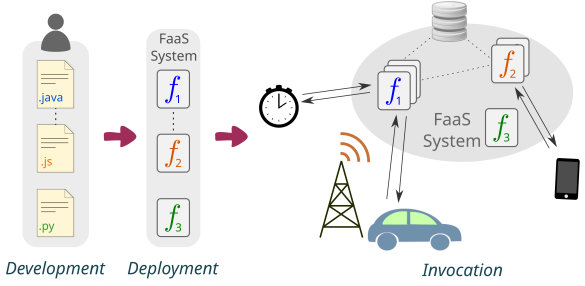
Example: Data Stream Processing (2)

- DSP apps usually defined as DAGs
- Each vertex is an operator, which transforms/analyzes incoming streams and outputs a new stream
- Horizontal and vertical scaling to face varying workloads



Example: Serverless Functions

- Function-as-a-Service (FaaS) is a trending paradigm for Cloud (and Edge) application development and deployment
- Supported by major Cloud providers
  - AWS Lambda
  - Google Cloud Functions
  - Azure Functions



Example: Serverless Functions (2)

- Functions executed in a serverless fashion
  - All operational issues managed by the provider
- Containers created on-demand to run functions
  - To isolate different functions
  - Startup much faster than a VM!
- Scaling to zero and to “infinity”
  - “Cold start” issue
  - Containers kept “warm” for a few minuted

Example: Elastic Bandwidth Allocation

- Software-defined Networking (SDN)
  - complete separation of network data plane and control plane
  - programmatic and dynamic configuration via software controllers
- Bandwidth allocation to multiple flows can be elastically adjusted

See, e.g., [Rodrigues et al. 2014]

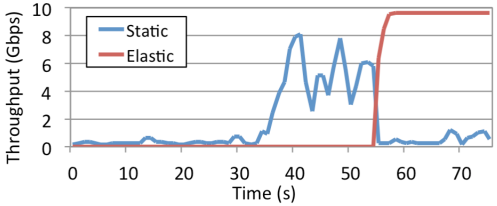


Figure 10: Elastic bandwidth allocation for large transfers. Throughput is limited by contention, and extra resources allow better application-level network performance.

Combining Mechanisms

- Elasticity may involve multiple scaling mechanisms
- Across dimensions: horizontal + vertical scaling
- Across stack layers: multi-level elasticity
  - Application-level (e.g., threads)
  - Infrastructure-level (e.g., virtual machines)
- Usually with different time scales

Complementary Mechanisms

- Auto-scaling is often coupled with additional system adaptation mechanisms
- Load balancing
- Migration

Load Balancing

- Distributing a set of tasks over a set of computing units (e.g., VMs), with the aim of making their overall processing more efficient
- Fundamental to take advantage of multiple parallel computing units, e.g., created through horizontal scaling
- Naive approach: round-robin
- More advanced techniques are load-aware

Migration

- Moving one or more software instances across distributed computing nodes
  - A “software instance” may be a whole VM or container!
  - We are often able to perform such migrations live (i.e., without interrupting the execution)
- Migration is important to balance the load across infrastructure nodes
- Can be coupled with vertical scaling
  - e.g., you cannot scale up the memory of a VM beyond the physical limit of the current node
  - you can migrate to a more powerful machine before scaling up

Stateful Applications

- Auto-scaling requires extra care if scaled components (e.g., threads) keep some internal state
  - e.g., data stream processing operators may store partial results in memory
- State should be re-distributed to the parallel instances after scaling in/out
- Common solution: key partitioning
  - State associated with a key
  - Each replica responsible for a set of keys
  - Load balancing impacted!
- Migration is more challenging as well!

Why

- Why bothering with auto-scaling?
- We are usually concerned with minimizing/maximizing some objectives
  - e.g., resource usage cost, application performance
- …or satisfying some constraints
  - e.g., keep energy consumption under a certain level
- Auto-scaling often involves multiple conflicting objectives
  - If we only cared about performance, we would simply over-provision
  - If we only cared about cost, we would shut down the servers :-)

Key Objectives

Auto-scaling objectives usually revolve around:

- **Cost** for acquiring computing resources from a provider
  - e.g., cost per minute of using a cloud VM
- **Performance** of the system
  - e.g., response time of a web application
- **Energy** consumption
- **Availability** of the system
  - some scaling mechanisms may cause the system to be unavailable during reconfiguration (e.g., rebooting a VM to change the number of available CPU cores)

Popular Performance Metrics

Application-oriented

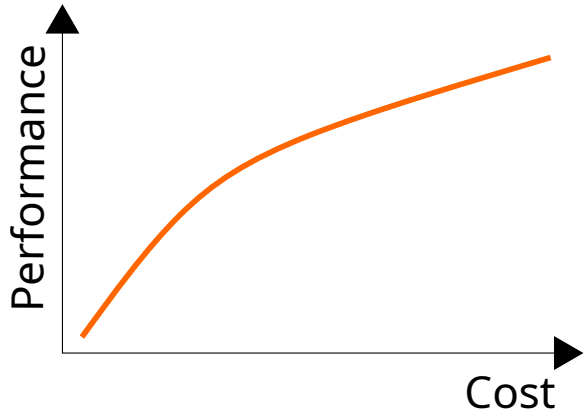
- Response time: time since a request arrives and a response is sent
  - e.g., time it takes for a web server to serve an incoming HTTP request
- Throughput: number of completed requests/tasks per unit of time
  - e.g., transactions per second in a database system

System-oriented

- Utilization: percentage of a resource that is currently being in use
  - CPU: % of time processor is busy
  - Memory: % of available memory in use
- Queue length: number of requests waiting for being served

Multiple Objectives

- Auto-scalers often seek for a trade-off among conflicting objectives
- Classic example: cost vs. performance



When

- When is the auto-scaler activated?
- **Triggered by events**
  - e.g., auto-scaling in FaaS systems is activated upon arrival of new invocation requests
- **Periodically** every X units of time
  - e.g., every 10 seconds we adjust CPU frequency based on load
  - How often?

Time Scale

- It rarely makes sense to make auto-scaling decisions faster than we can apply (and evaluate) them
- But it cannot be slower than workload variations!

Example: time to scale-out (approximate)

Physical server	5+ min
VM	1 min
Container	100ms–1s
Thread	< 1ms

Who

- Who is responsible for auto-scaling?
- It depends …
- Responsibility often shared by resource provider and customers
  - e.g., AWS takes care of operational issues, but the user decides the policy to use
- Sometimes, fully managed by the provider/infrastructure
  - e.g., serverless systems
  - e.g., DVFS
- Sometimes, fully managed by applications

Where

- Where are auto-scaling decisions made?
- Different control architectures can be adopted
- Centralized vs. decentralized control
- We will come back to this issue later!

How

- How are auto-scaling decisions made?
  - Which is the policy in use?
- Reactive vs. Proactive
- Model-free vs. Model-based
- Several techniques have been used:
  - Control theory
  - Queueing theory
  - Machine learning
  - Mathematical optimization
  - Heuristics

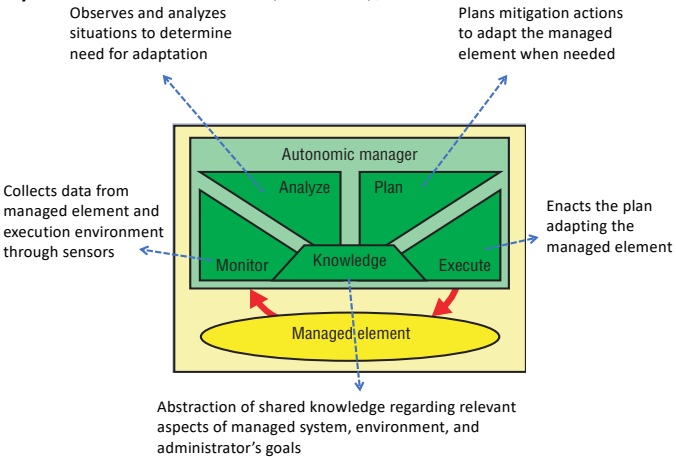
Example: Threshold-Based Scaling Policy

- The most popular scaling policy (AWS EC2 Auto Scaling, Kubernetes HPA, ...)
- Define high and low thresholds on some metric(s) (e.g., CPU utilization, memory utilization, response time)
- Compare measured metrics against thresholds and change replicas number if thresholds are exceeded
- Example of rules: dynamic scaling policy in EC2 Auto Scaling [AWS 2021]
  - Scale-out rule: if average CPU utilization of all replicas >70% in last 1 min, add 1 new replica
  - Scale-in rule: if average CPU utilization of all replicas is <35% in last 5 min, remove 1 replica
- Simple and intuitive policy ✓
- Thresholds definition: subjective and prone to uncertainty X

Control Architectures for Auto-Scaling

Reference Architecture for Self-Adaptive Systems

- Monitor-Analyze-Plan-Execute (MAPE), with shared Knowledge (MAPE-K)



Seminal paper by [Kephart and Chess 2003]

MAPE: Monitor & Analyze Phases

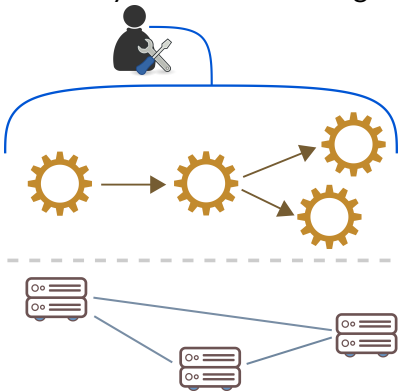
- Monitor: main design options
  - *When*: continuously, on demand
  - *What*: resources, workload, application performance, …
  - *How*: architecture (centralized vs. decentralized), methodology (active vs. passive)
  - *Where* to store monitored data and how (e.g., some pre-processing)
- Analyze: main design options
  - *When*: event- or time-triggered
  - *How*: reactive vs. proactive
    - Reactive: in reaction to events that have already occurred (e.g., scale-out to react to workload increase)
    - Proactive: based on prediction so to plan adaptation actions in advance (e.g., scale-out before workload increase effectively occurs)

MAPE: Plan Phase

- The most studied MAPE phase
- A variety of methodologies (to manage cloud/edge applications performance at run-time)
  - Optimization theory
  - Queueing theory
  - Heuristics
  - Meta-heuristics
  - Control theory
  - Machine learning (including reinforcement learning)
- Surveys: [Lorido-Botran, Miguel-Alonso, and Lozano 2014; Chen, Bahsoon, and Yao 2018; Al-Dhuraibi et al. 2018]

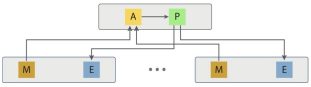
Designing the Architecture of the Managing System

- How to design the control architecture for cloud/edge applications
- Centralized MAPE
  - All MAPE components on same node
  - Simple, global view on master ✓
  - Lack of scalability and resiliency in multi-cloud/edge environments X

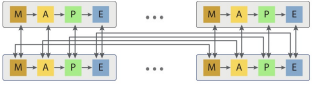


Designing the Architecture of the Managing System

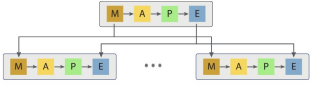
- Decentralized MAPE
  - How to distribute MAPE phases
  - Many patterns, including master-worker, fully decentralized, and hierarchical [Weyns et al. 2013]
  - No clear winner, depending on system, environment and application features and requirements



Master-worker

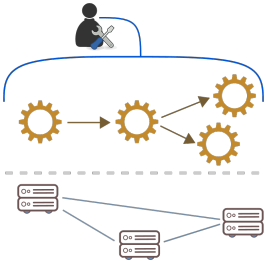


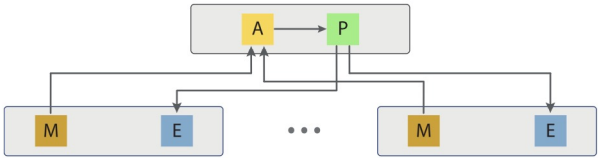
Fully decentralized (e.g., coordinated)



Hierarchical

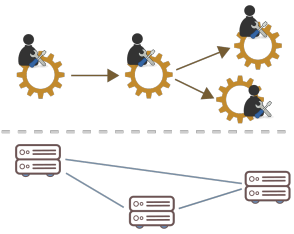
Decentralized MAPE: Master-Worker

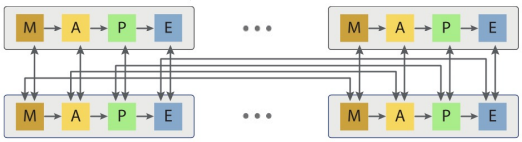




- Decentralize M and E on workers, keep A and P centralized on master
- Global view on master ✓
- Communication overhead, bottleneck risk and single point of failure on master X

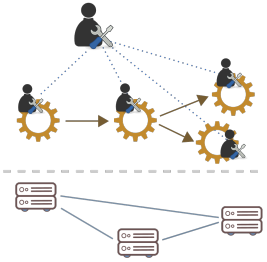
Decentralized MAPE: Fully Decentralized

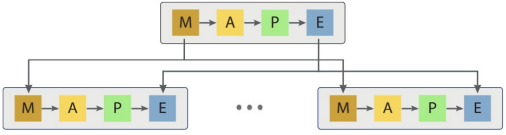




- Multiple control loops, each one in charge of some part of the controlled system, possibly coordinated through interaction (or no interaction at all)
- Cloud/edge applications: distinct control loop per application component
- Improved scalability ✓
- More difficult to take joint adaptation decisions X

Decentralized MAPE: Hierarchical

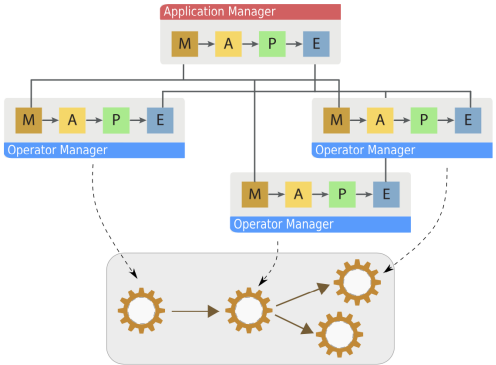




- Multiple MAPE loops, which can operate at different time scales and with separation of concerns
- Top-level MAPE can achieve global goals ✓
- Non-trivial to identify multiple levels of control, depends on system characteristics X
- Cloud/edge applications:
  - Top-level MAPE: entire application
  - Bottom-level MAPE: application component

Hierarchical Decentralized MAPE for DSP Applications

- Elastic and Distributed DSP Framework (EDF): hierarchical distributed architecture based on Apache Storm for controlling elasticity of DSP applications [Cardellini et al. 2018]
- Application Manager + Operator Manager



References I

- Arroba, Patricia et al. (2015). "DVFS-Aware Consolidation for Energy-Efficient Clouds". In: *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pp. 494–495. DOI: 10.1109/PACT.2015.59.
- AWS (2021). *Amazon EC2 Auto Scaling*. https://aws.amazon.com/ec2/autoscaling/.
- Cardellini, V. et al. (2018). "Decentralized Self-Adaptation for Elastic Data Stream Processing". In: *Future Gener. Comput. Syst.* 87, pp. 171–185. DOI: https://doi.org/10.1016/j.future.2018.05.025.
- Chen, T., R. Bahsoon, and X. Yao (2018). "A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems". In: *ACM Comput. Surv.* 51.3.
- Al-Dhuraibi, Y. et al. (2018). "Elasticity in Cloud Computing: State of the Art and Research Challenges". In: *IEEE Trans. Serv. Comput.* 11, pp. 430–447. DOI: 10.1109/TSC.2017.2711009.

References II

- Herbst, N. R., S. Kounev, and R. Reussner (2013). “Elasticity in Cloud Computing: What It Is, and What It Is Not”. In: *Proc. ICAC ’13)*, pp. 23–27. URL: https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst.
- Kephart, J.O. and D.M. Chess (2003). “The Vision of Autonomic Computing”. In: *IEEE Computer* 36.1, pp. 41–50. DOI: 10.1109/MC.2003.1160055.
- Kubernetes (2021). *Horizontal Pod Autoscaler*. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/.
- Lorido-Botran, T., J. Miguel-Alonso, and J. A. Lozano (2014). “A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments”. In: *J. Grid Comput.* 12.4, pp. 559–592. DOI: 10.1007/s10723-014-9314-7.

References III

- Rodrigues, Henrique et al. (2014). “Traffic Optimization in Multi-layered WANs Using SDN”. In: *22nd IEEE Annual Symposium on High-Performance Interconnects, HOTI 2014, Mountain View, CA, USA, August 26-28, 2014*. IEEE Computer Society, pp. 71–78. DOI: 10.1109/HOTI.2014.23. URL: https://doi.org/10.1109/HOTI.2014.23.
- Wang, T., S. Ferlin, and M. Chiesa (2021). “Predicting CPU Usage for Proactive Autoscaling”. In: *Proc. EuroMLSys ’21*, pp. 31–38. DOI: 10.1145/3437984.3458831.
- Weyns, D. (2020). *An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective*. Hoboken, NJ, USA: Wiley-IEEE Computer Society Press.

References IV

Weyns, D. et al. (2013). “On Patterns for Decentralized Control in Self-Adaptive Systems”. In: *Software Engineering for Self-Adaptive Systems II*. Vol. 7475. LNCS. Springer.