Autoscaling challenges of multi-tenant SaaS platforms

Guest lecture for Aalto University course “Networking at Scale and Advanced Applications”, March 18, 2025

Lari Hotari

lari@hotari.net

Learning outcomes

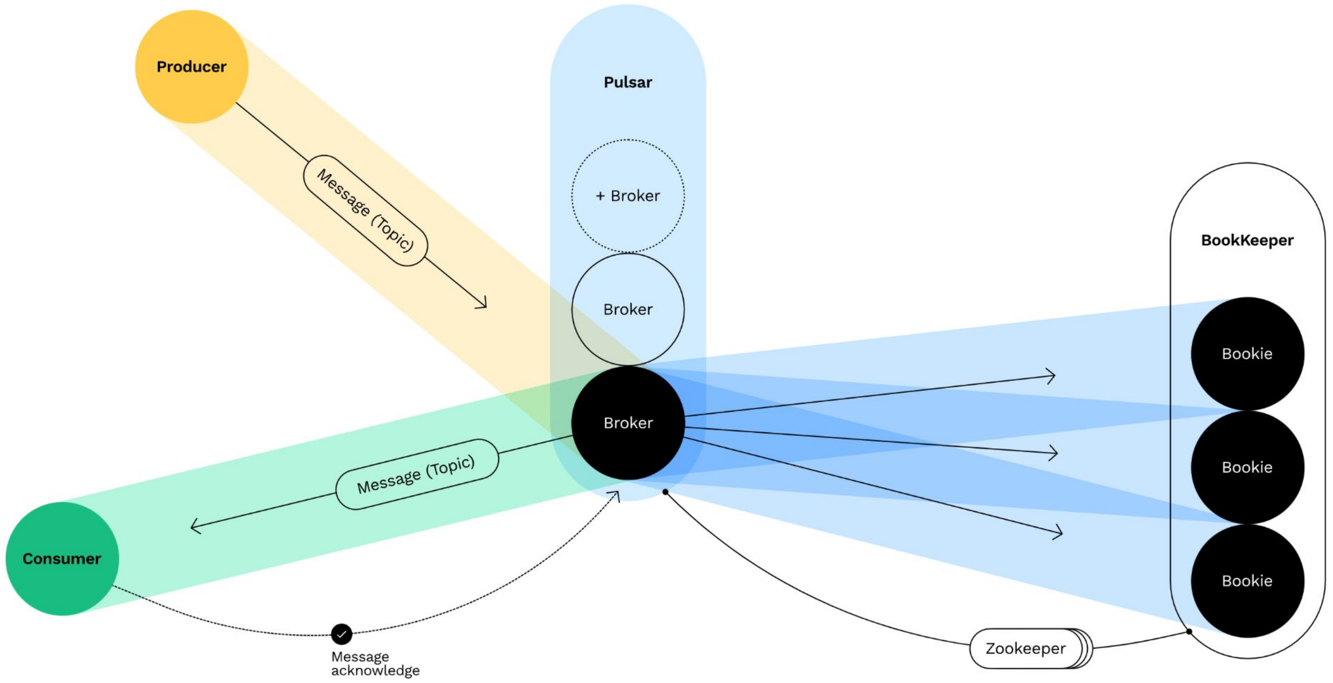
- Multi-tenancy & autoscaling
- Service provider and consumer perspectives
- Different SaaS platform types
- Capacity Unit
- Capacity management challenges
- Noisy Neighbour challenges
- Aggregated workload patterns

About me

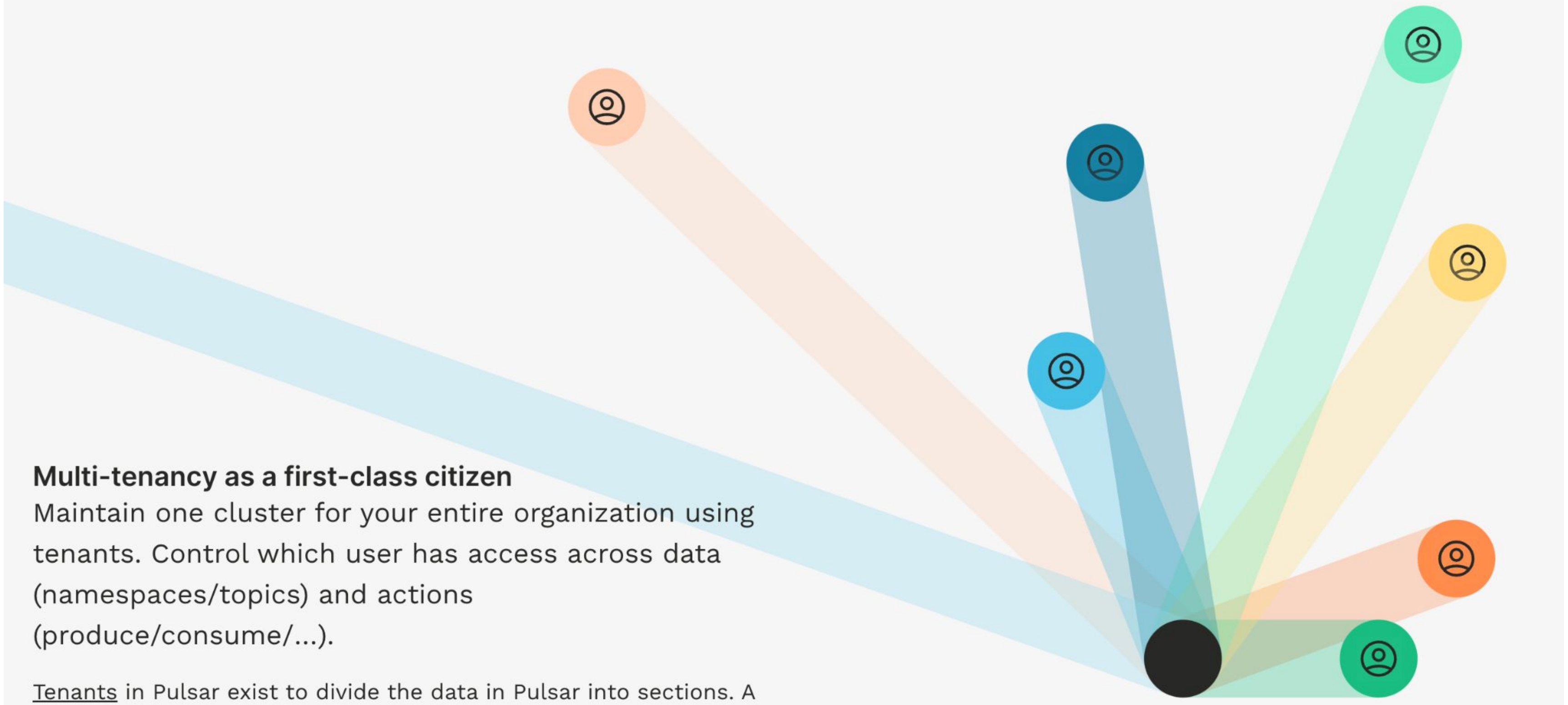
- Sr. Technologist in the Office of CTO at StreamNative.
  - Working StreamNative Cloud, powered by Apache Pulsar, an open-source, distributed messaging and streaming platform built for the cloud.
  - Prior to StreamNative, Engineering Manager of Streaming Customer Reliability Engineering at DataStax working on the Astra Streaming, powered by Apache Pulsar.
- 25+ years of experience designing, building, and operating large-scale distributed systems with small and large teams.
- Open source contributor with 5000+ commits across major projects including Apache Pulsar, Apache Bookkeeper, Grails, Groovy, Spring Framework, Spring Boot, and Gradle.
  - https://github.com/lhotari

Apache Pulsar - open-source, distributed messaging and streaming platform

Horizontally scalable with an architecture that separates compute and storage: a cluster scales to millions of messages per second across up to 1 million topics with very low latency (<10ms) from publish to consume



Provides both streaming semantics like Apache Kafka and message queue semantics like RabbitMQ or ActiveMQ



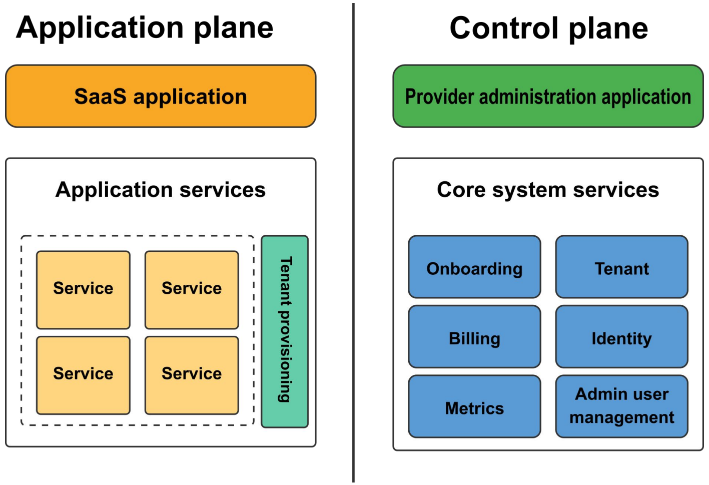
Multi-tenancy as a first-class citizen

Maintain one cluster for your entire organization using tenants. Control which user has access across data (namespaces/topics) and actions (produce/consume/...).

Tenants in Pulsar exist to divide the data in Pulsar into sections. A tenant holds namespaces which in turn hold topics and Pulsar Functions. Organizations typically give each department/team its tenant, in which they create namespaces per domain they own and topics for the implementation they need for that domain.

Multi-tenant SaaS platforms

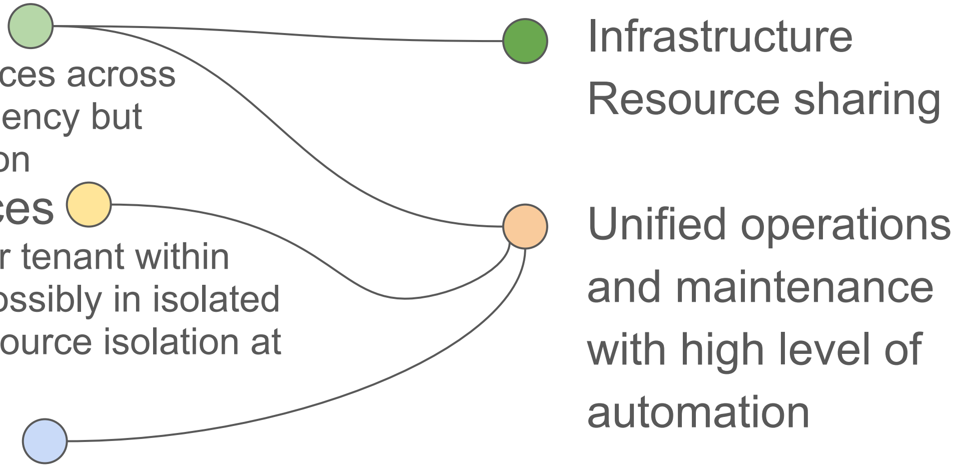
- Economies of scale by
  - Infrastructure Resource sharing
  - Unified operations and maintenance with high level of automation
- Logical isolation of tenants
- Separation of control plane and application plane



Source: Building Multi-Tenant SaaS Architectures. O'Reilly Media. 2024.

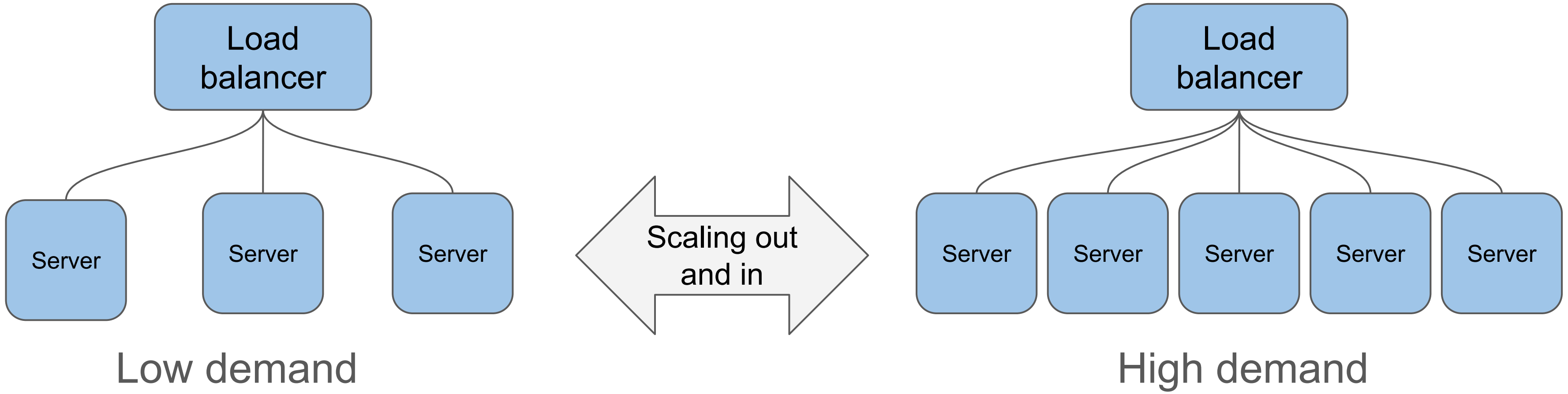
Different SaaS types based on infrastructure resource allocation

- Shared Resources Multi-tenancy
  - Pooling compute and storage resources across multiple customers, maximizing efficiency but requiring sophisticated tenant isolation
- Dedicated Infrastructure Resources
  - Provisioning dedicated resources per tenant within the cloud provider's infrastructure, possibly in isolated cloud accounts, offering stronger resource isolation at higher cost
- Bring Your Own Account (BYOC)
  - Deploying and managing services within the customer's own cloud account, providing maximum isolation while leveraging existing customer infrastructure


Infrastructure Resource sharing
Unified operations and maintenance with high level of automation

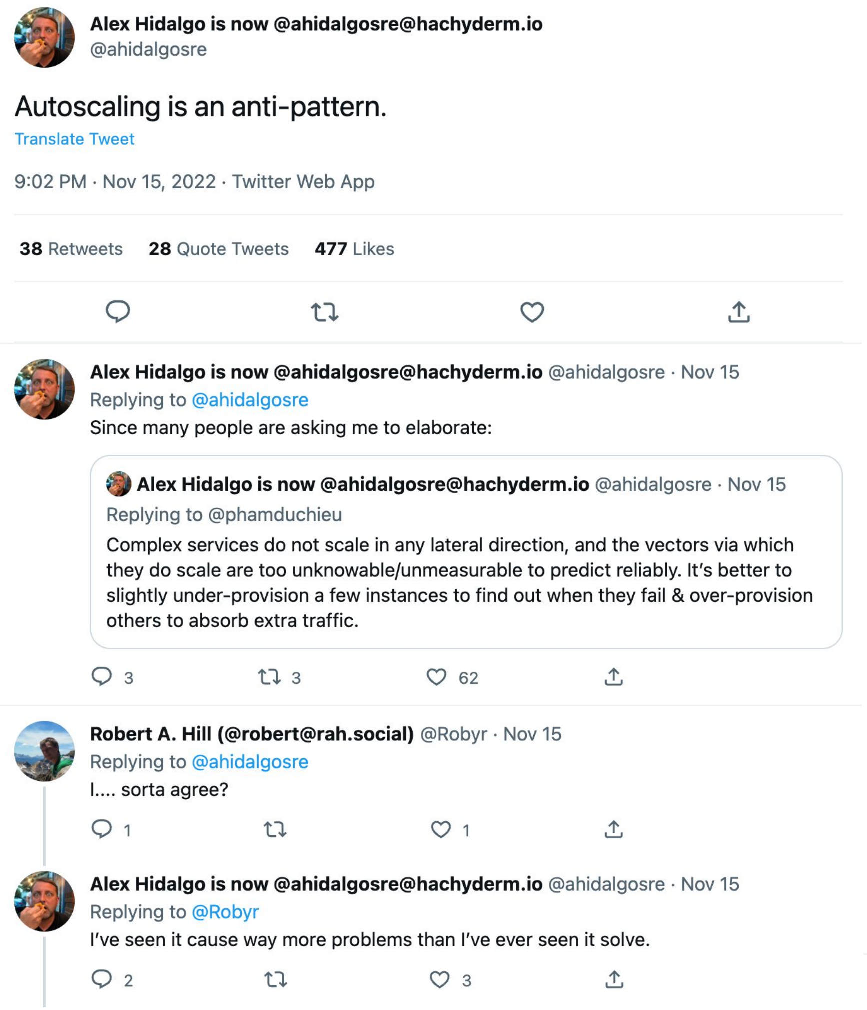
Goals of autoscaling (elasticity)

- Saving costs by avoiding over-provisioning resources for peak workloads
  - Adjusting resource allocation dynamically to match changing demands



General autoscaling challenges

- The application architecture itself must be horizontally scalable
  - Distribute workload across multiple instances (load balancing)
  - Gracefully handle instance addition or termination
    - Maintain consistency
    - Transfer or redistribute state
- Scaling operations often introduce temporary performance impacts or service disruptions


Alex Hidalgo is now @ahidalgosre@hachyderm.io
@ahidalgosre

Autoscaling is an anti-pattern.

9:02 PM · Nov 15, 2022 · Twitter Web App

38 Retweets 28 Quote Tweets 477 Likes

Alex Hidalgo is now @ahidalgosre@hachyderm.io @ahidalgosre · Nov 15
Replying to @ahidalgosre
Since many people are asking me to elaborate:

Alex Hidalgo is now @ahidalgosre@hachyderm.io @ahidalgosre · Nov 15
Replying to @phamduchieu
Complex services do not scale in any lateral direction, and the vectors via which they do scale are too unknowable/unmeasurable to predict reliably. It's better to slightly under-provision a few instances to find out when they fail & over-provision others to absorb extra traffic.

3 3 62

Robert A. Hill (@robert@rah.social) @Robyr · Nov 15
Replying to @ahidalgosre
I.... sorta agree?

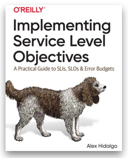
1 1

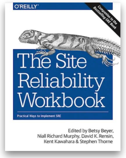
Alex Hidalgo is now @ahidalgosre@hachyderm.io @ahidalgosre · Nov 15
Replying to @Robyr
I've seen it cause way more problems than I've ever seen it solve.

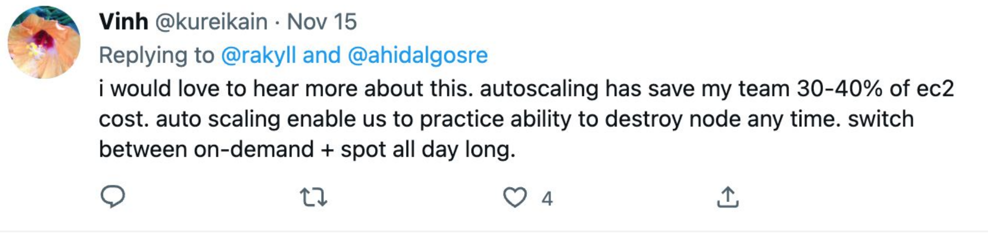
2 3

"Autoscaling is an anti-pattern: Complex services do not scale in any lateral direction, and the vectors via which they do scale are too unknowable/unmeasurable to predict reliably. It's better to slightly under-provision a few instances to find out when they fail & over-provision others to absorb extra traffic."

- *Alex Hidalgo (Site Reliability Engineer, book author)*


O'REILLY
Implementing Service Level Objectives
A Practical Guide to SLIs, SLOs & Error Budgets
Alex Hidalgo




Vinh @kureikain · Nov 15
Replying to @rakyll and @ahidalgosre
i would love to hear more about this. autoscaling has save my team 30-40% of ec2 cost. auto scaling enable us to practice ability to destroy node any time. switch between on-demand + spot all day long.

4

Source: https://web.archive.org/web/20221125003448/https://twitter.com/ahidalgosre/status/1592624122521321472

Two contrasting perspectives of Autoscaling

Against:

- “Autoscaling is an anti-pattern”
- Creates more problems than it solves in complex systems
- Unpredictable scaling behavior leads to reliability issues

For:

- “Autoscaling has saved my team 30-40% of EC2 cost."
- Provides financial benefits through resource optimization

Two sides of autoscaling in SaaS platforms

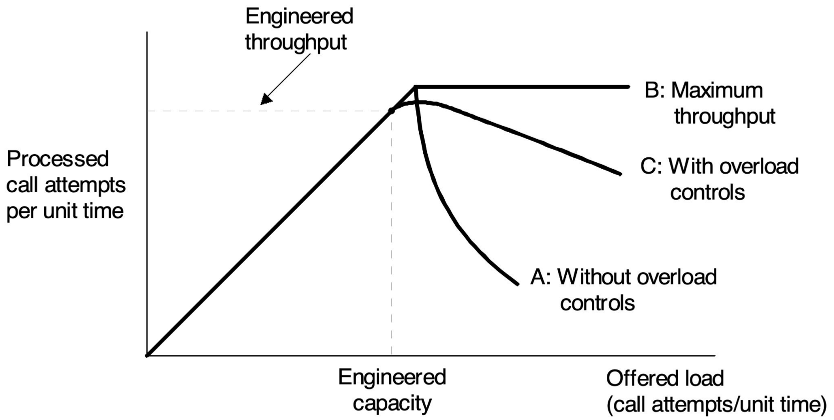
- Service Provider's Perspective
  - Capacity Management Responsibility
    - **Accountable for scaling resources** to fulfill service level agreements (SLAs) and maintain consistent performance (Quality of Service) across all service consumers (tenants)
- Service Consumer's (Tenant) Perspective
  - Outsourced Capacity Planning
    - Expectation that the provider handles all scaling complexities transparently
  - Flexible Resource Acquisition and Releasing
    - Need for clear, predictable mechanisms to **increase and decrease capacity on-demand** with proportional pricing
  - Performance Guarantees
    - Consistent service quality regardless of overall platform load or other tenant activities

Connecting the two sides of autoscaling with Capacity Units

- Capacity Unit is an abstraction specific for the application domain
- The service consumer should be able to increase and decrease capacity of the service in terms of capacity units
- The service provider increases and decreases the infrastructure capacity of the service to meet the demands of the acquired capacity units

What is capacity?

- A concept of "**engineered capacity**" has been used in performance evaluation of telecom exchanges for several decades. One example of the definition of "engineered capacity" is in ITU-T Q.543 recommendation, “Digital exchange performance design objectives” from 1993



- ITU-T Q.543 defines engineered capacity as the maximum load that the system can handle *while meeting performance (quality) requirements*. It is not necessarily the maximum throughput capacity.

Serverless

- Capacity units in truely Multi-tenant SaaS create a "serverless" experience for service consumers
  - There’s no need for the service consumer to think about underlying infrastructure resources
  - They can acquire and release capacity units in terms of the application domain specific units at a granular level without committing to dedicated servers

Autoscaling challenges in Multi-tenant SaaS platforms

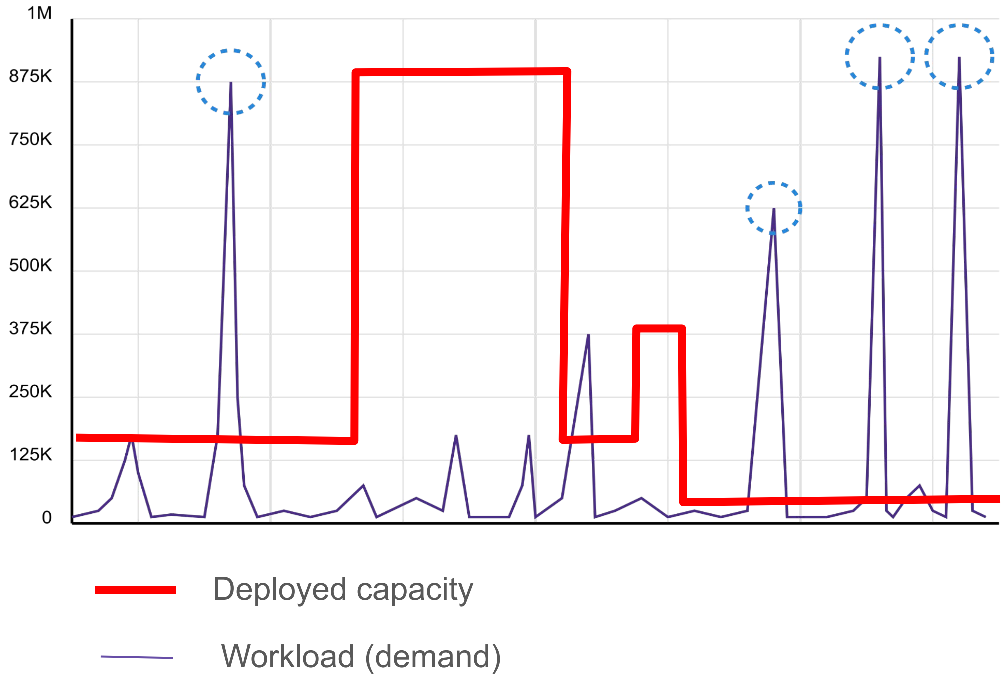
Service Provider challenges:

- Capacity management challenges
    - Adjusting resource allocation dynamically without causing
        - quality of service degradation that violates the SLA
        - excessive over-provisioning of resources
    - Lead time of adding more capacity to the system
- “Noisy neighbor” problems when resources are shared
    - Fairly serving service consumers according to acquired capacity units
        - From service consumer’s viewpoint the acquired capacity units are “guaranteed capacity” unless explicitly stated in other terms such as “burst units” or “best-effort capacity”
    - Limiting service consumers to acquired capacity units
    - Also connects to other general scaling challenges such as load balancing and overload protection

Autoscaling challenge: Lead time of adding more capacity to the system

- If the lead time to making changes to resource allocations is long, there's a challenge in autoscaling to meet the demand (the workload)
- This is especially a problem with spiky workloads

Example where capacity doesn't meet demand

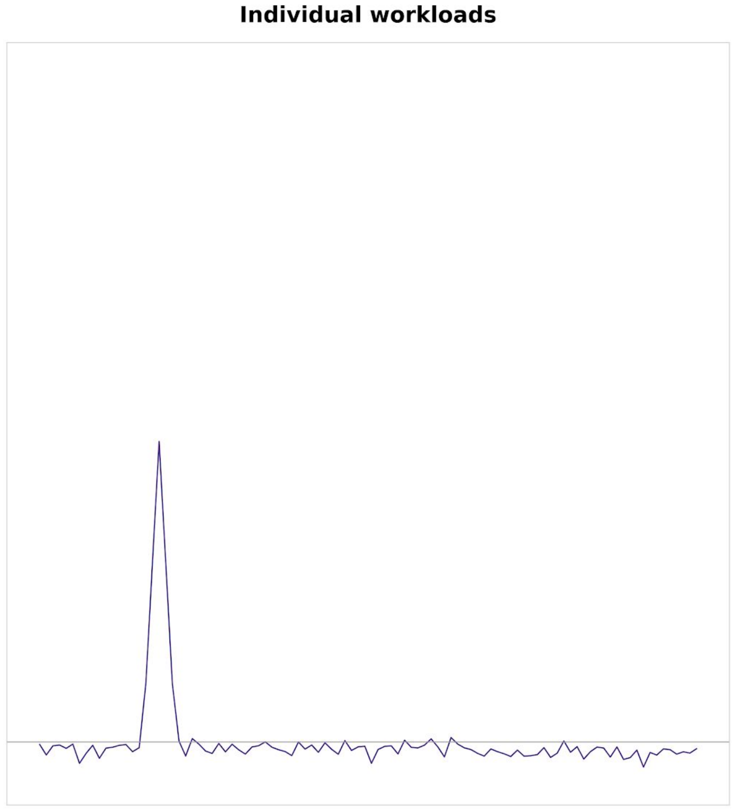


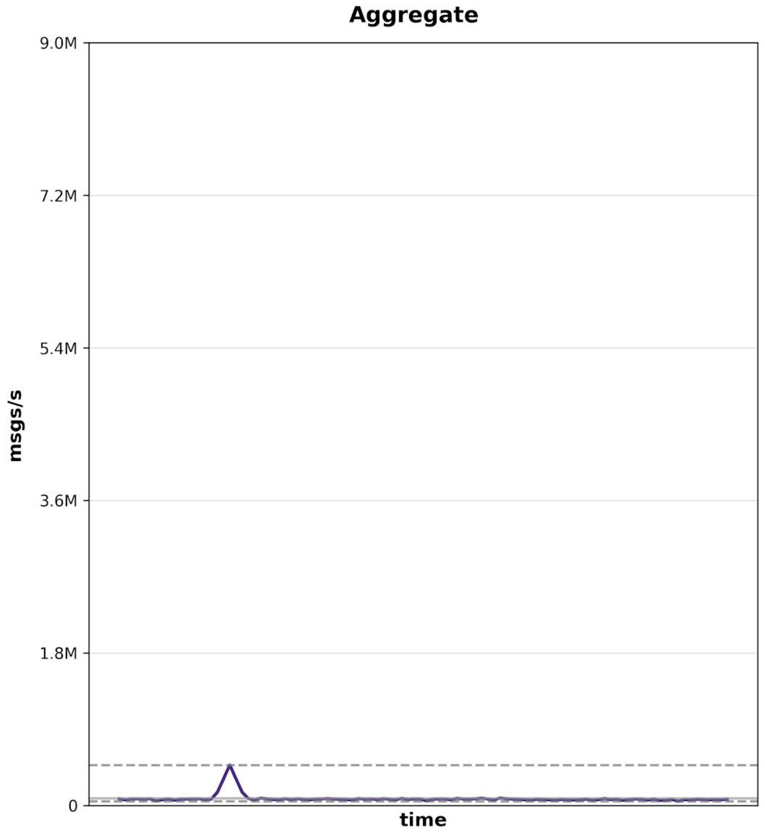
Aggregating workloads

- Under strict performance quality requirements, systems must provision for peak load, not average load

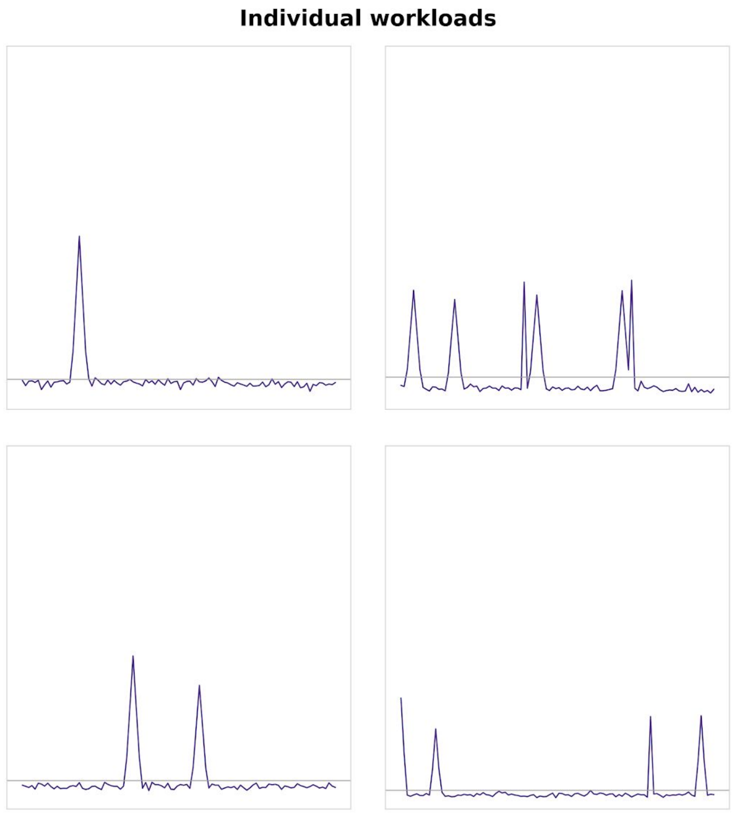
- The following slides demonstrates the effect of aggregating workloads using a Python simulation
    - Source code: https://github.com/lhotari/workload-aggregation-visualizer
- This simulation is inspired by Andrew Warfield's talk at FAST '23: "Building and Operating a Pretty Big Storage System (My Adventures in Amazon S3)"

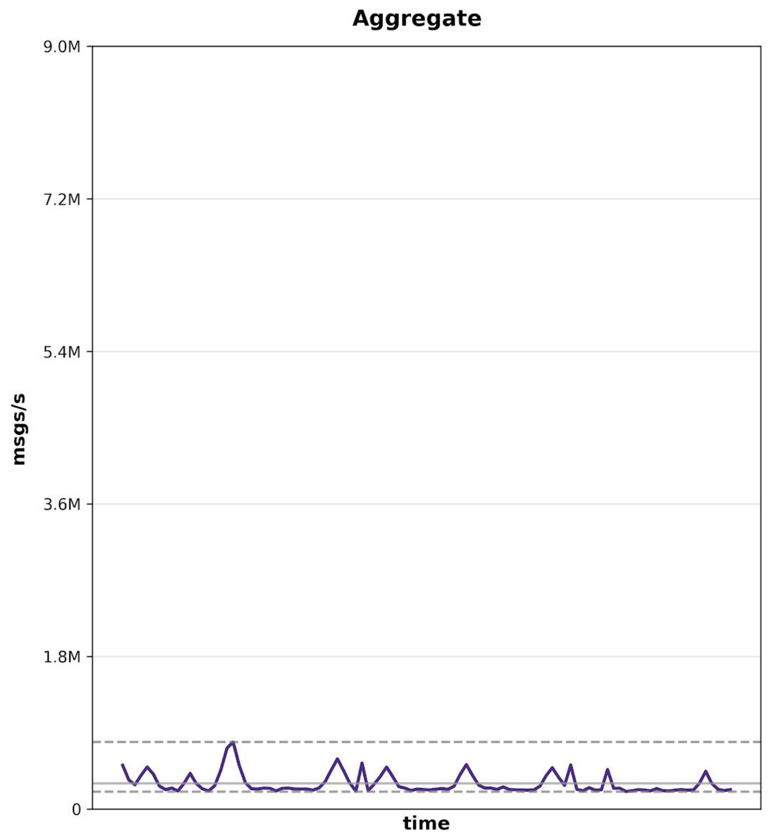
Effect of aggregating decorrelated workloads on net system load (n=1)



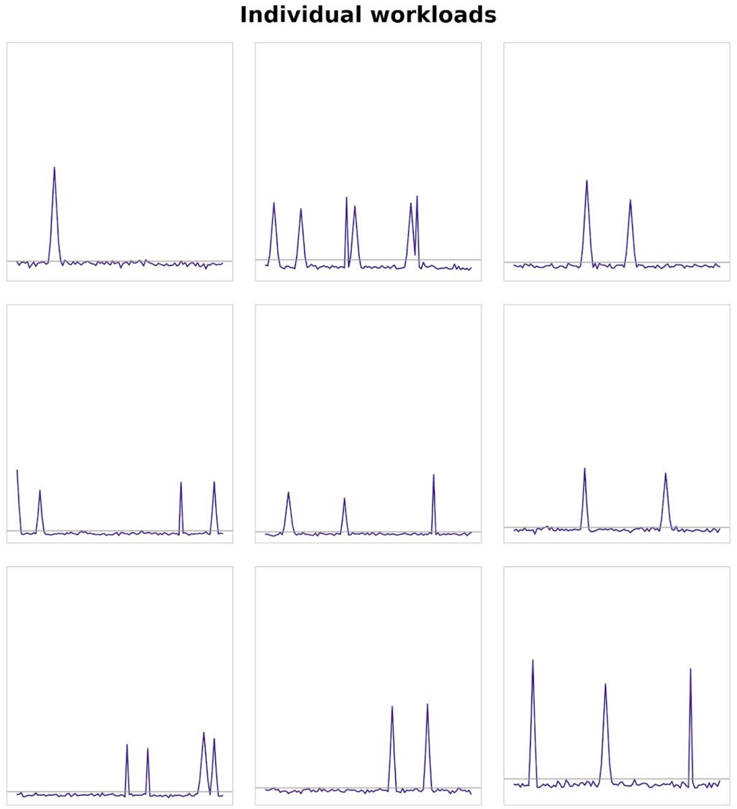


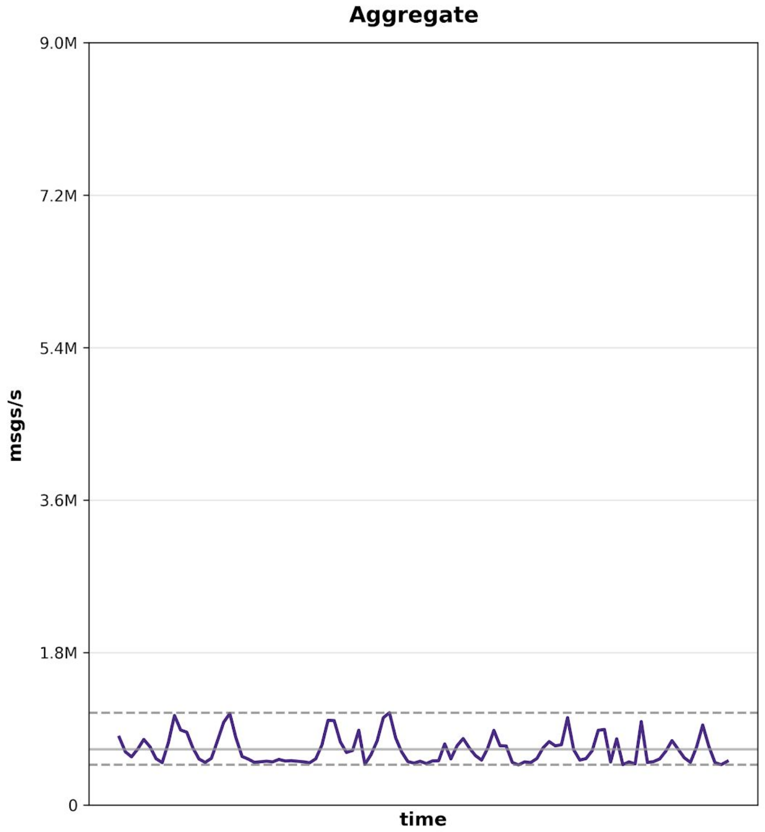
Effect of aggregating decorrelated workloads on net system load (n=4)



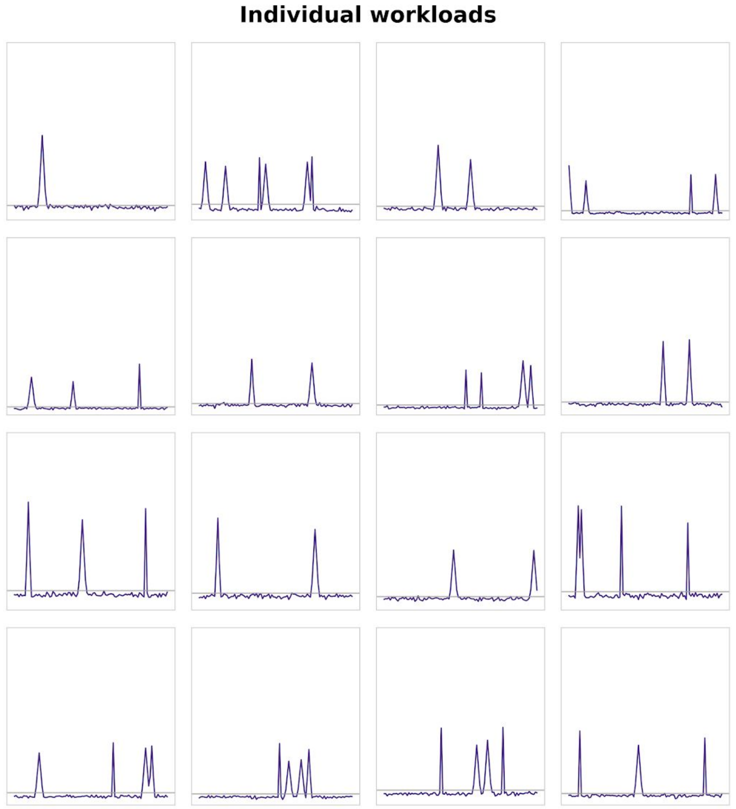


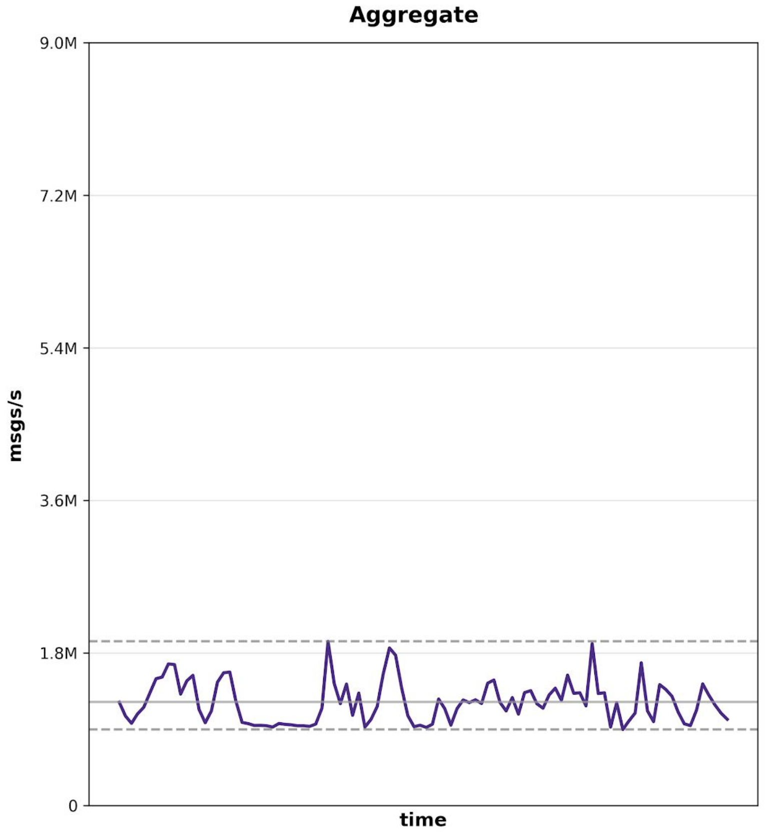
Effect of aggregating decorrelated workloads on net system load (n=9)



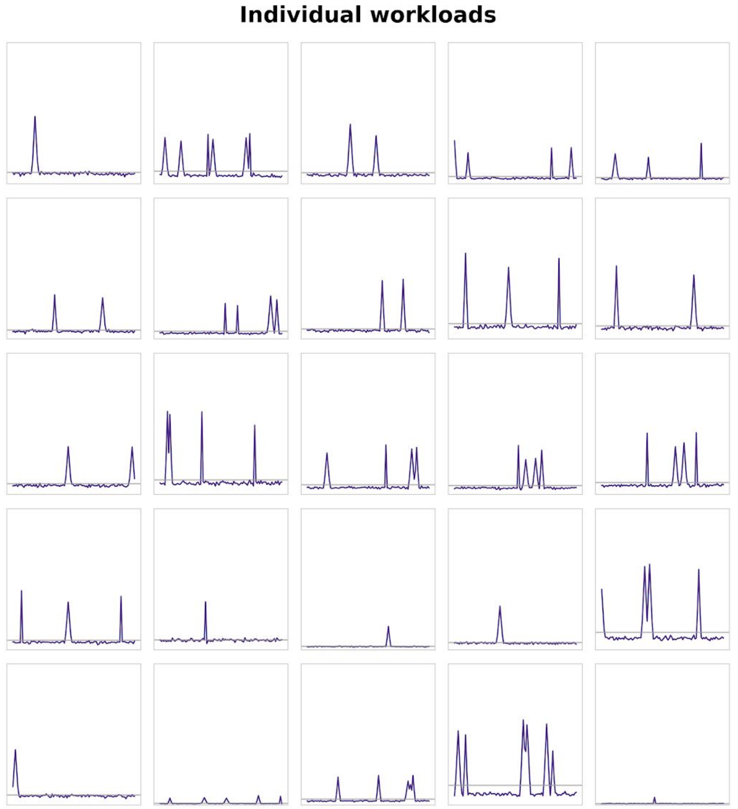


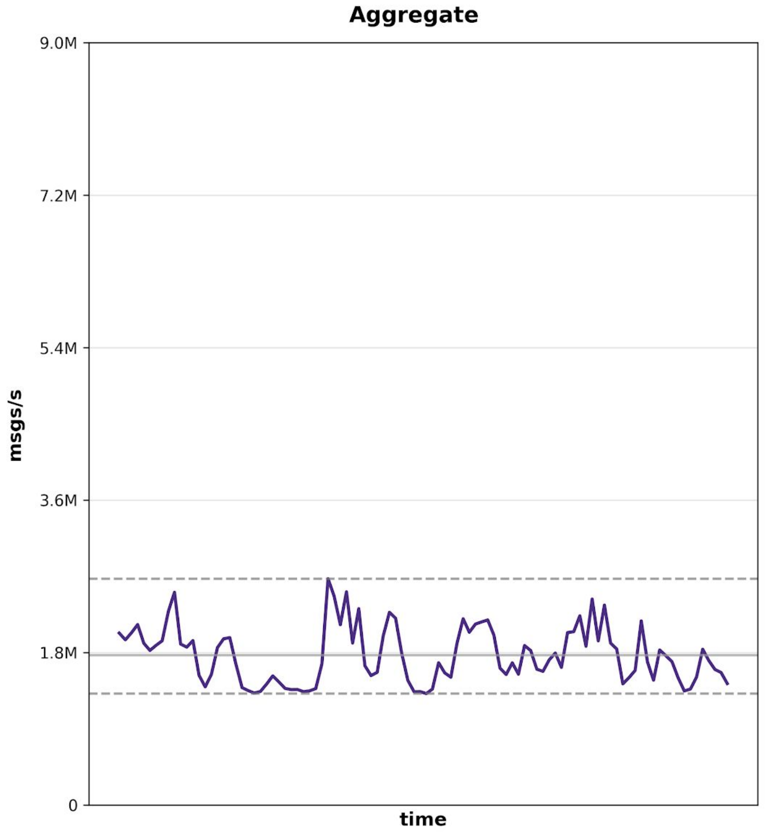
Effect of aggregating decorrelated workloads on net system load (n=16)



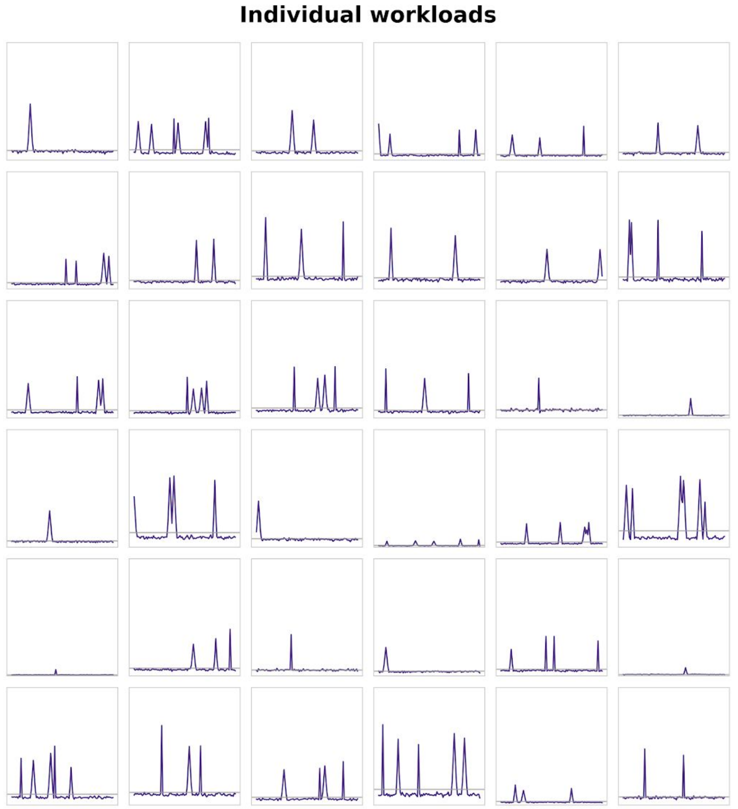


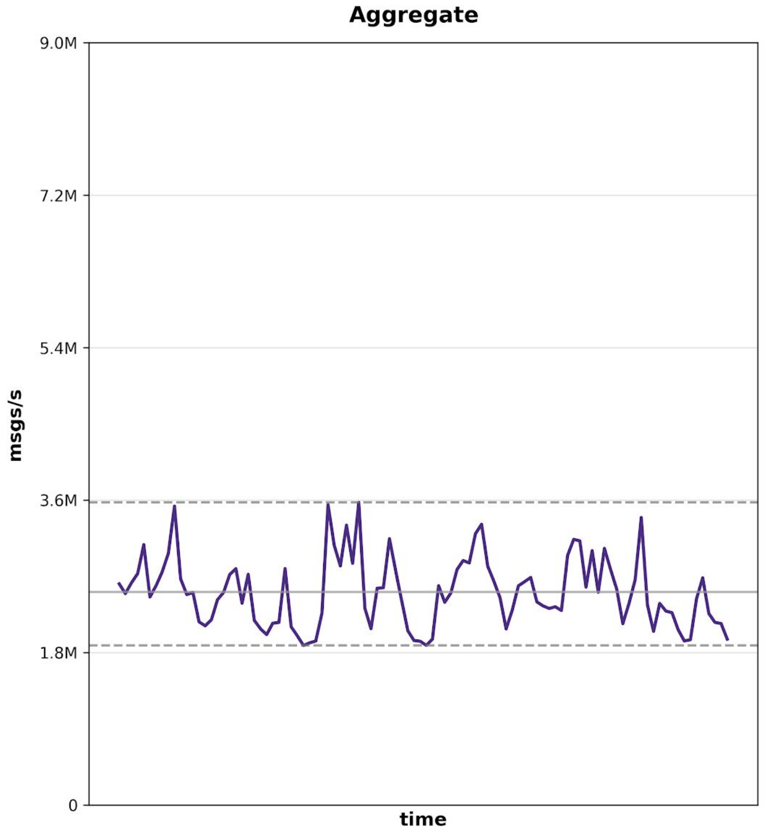
Effect of aggregating decorrelated workloads on net system load (n=25)



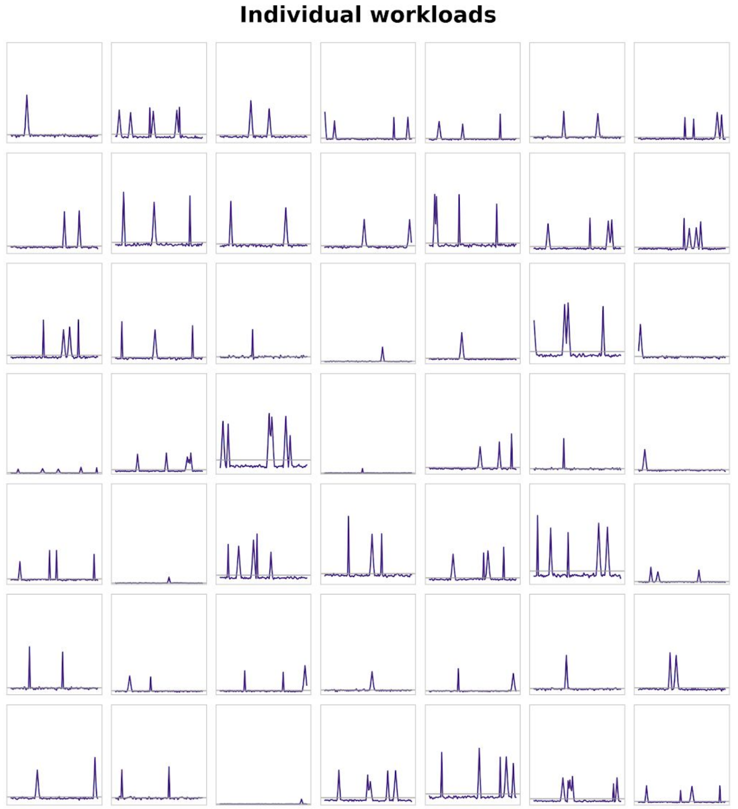


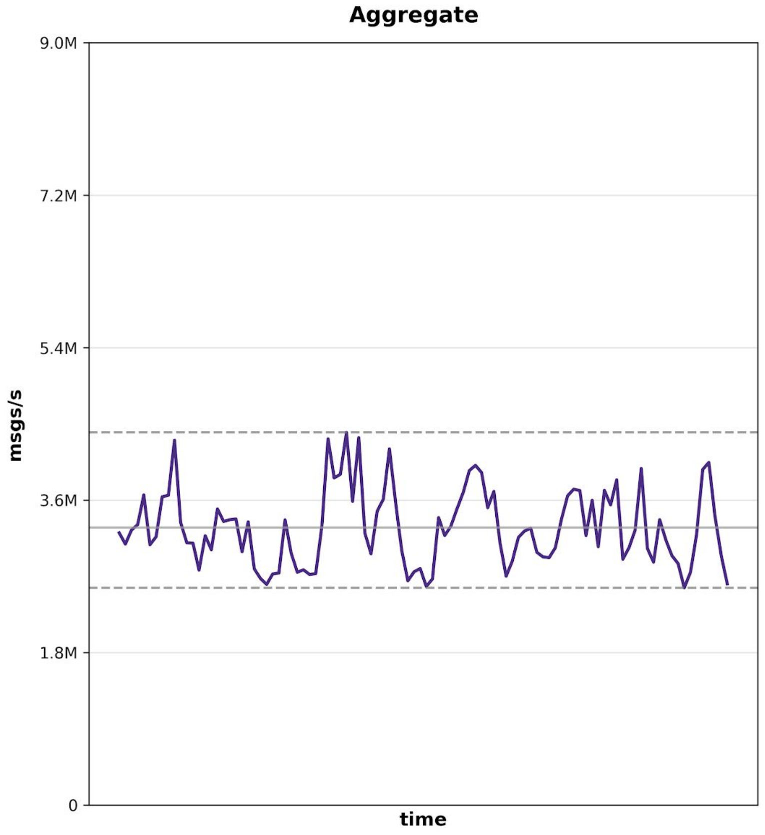
Effect of aggregating decorrelated workloads on net system load (n=36)



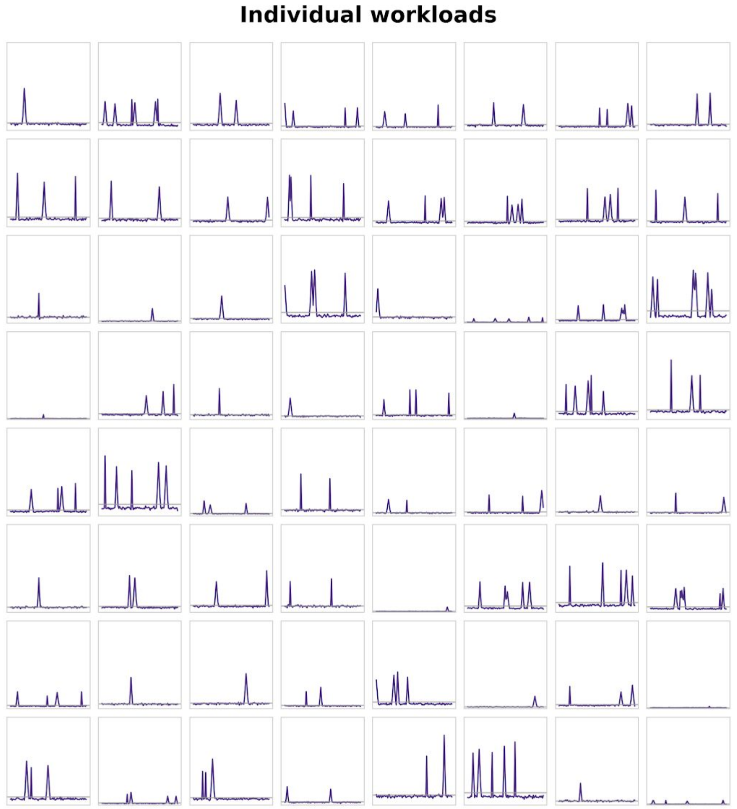


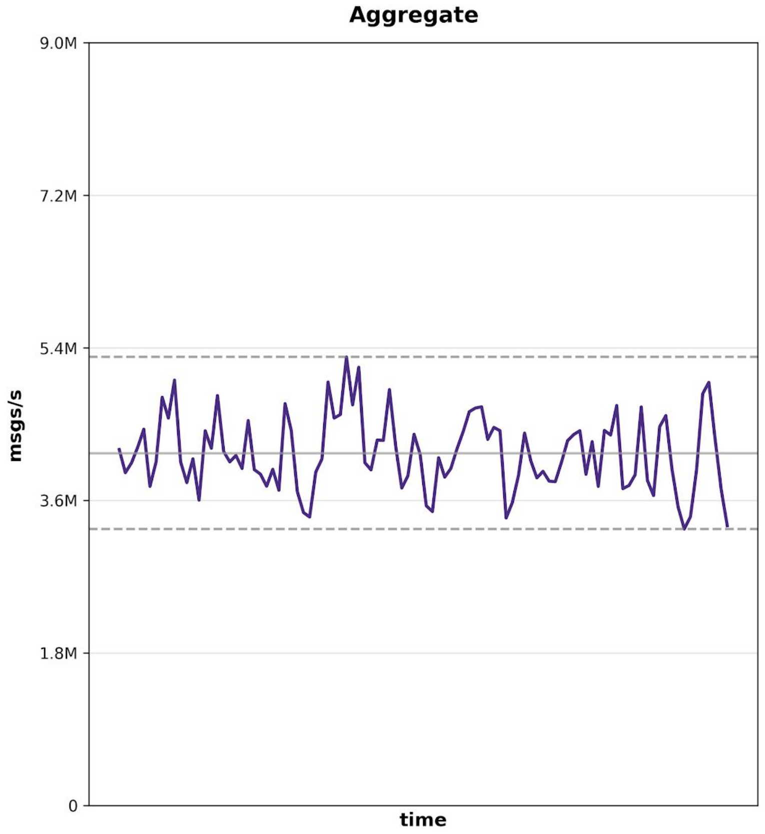
Effect of aggregating decorrelated workloads on net system load (n=49)



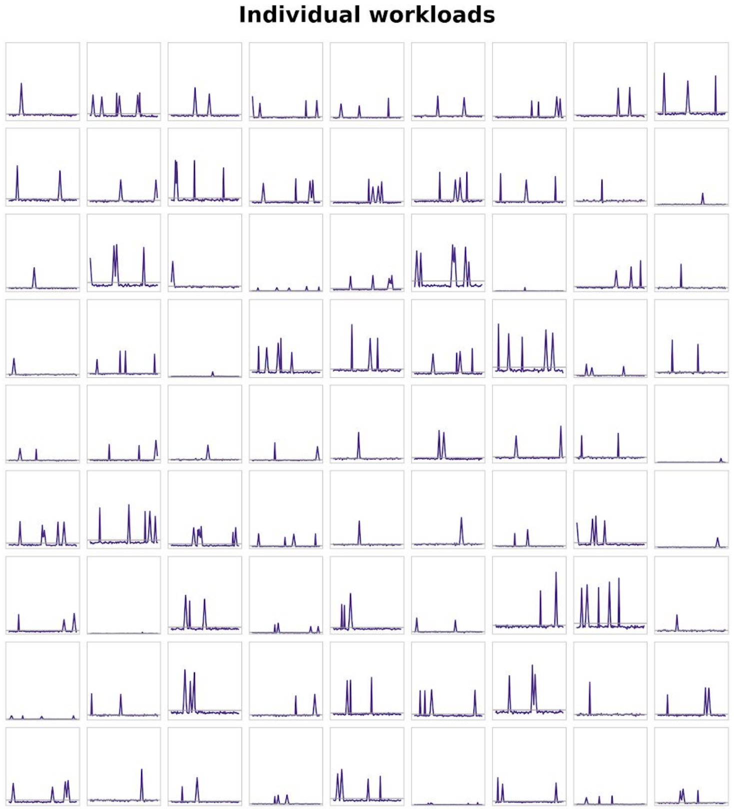


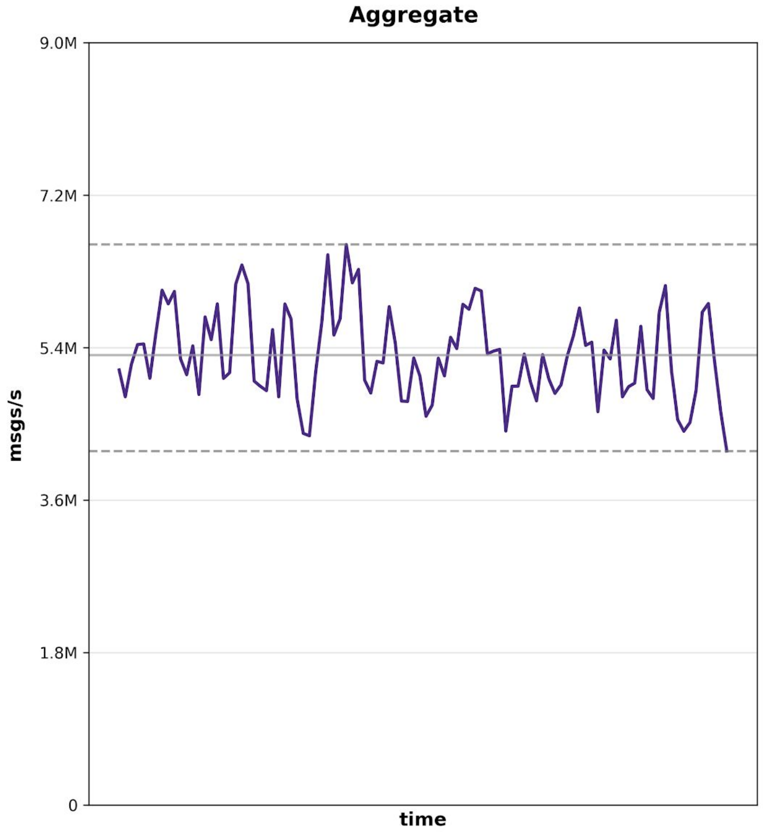
Effect of aggregating decorrelated workloads on net system load (n=64)



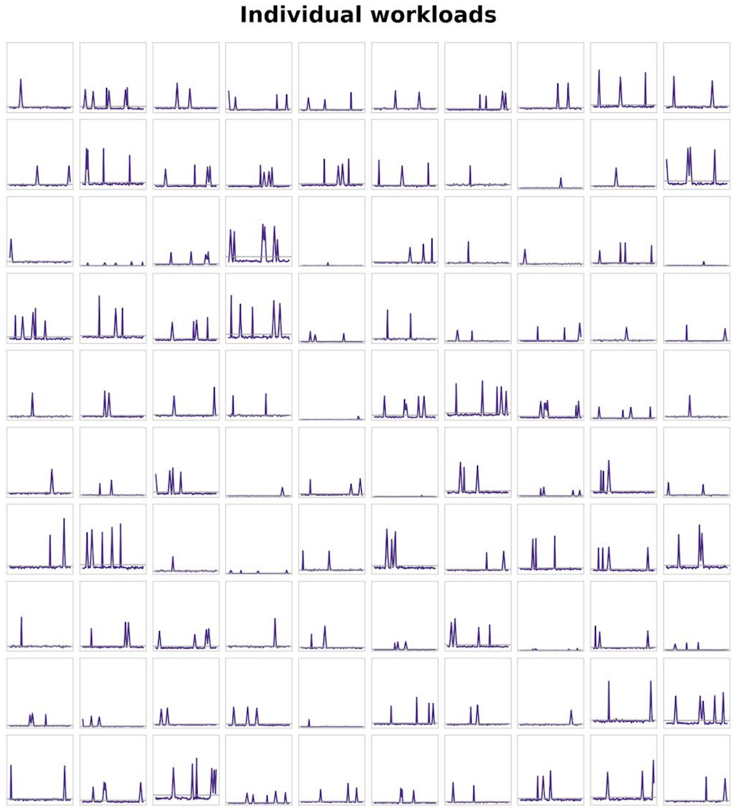


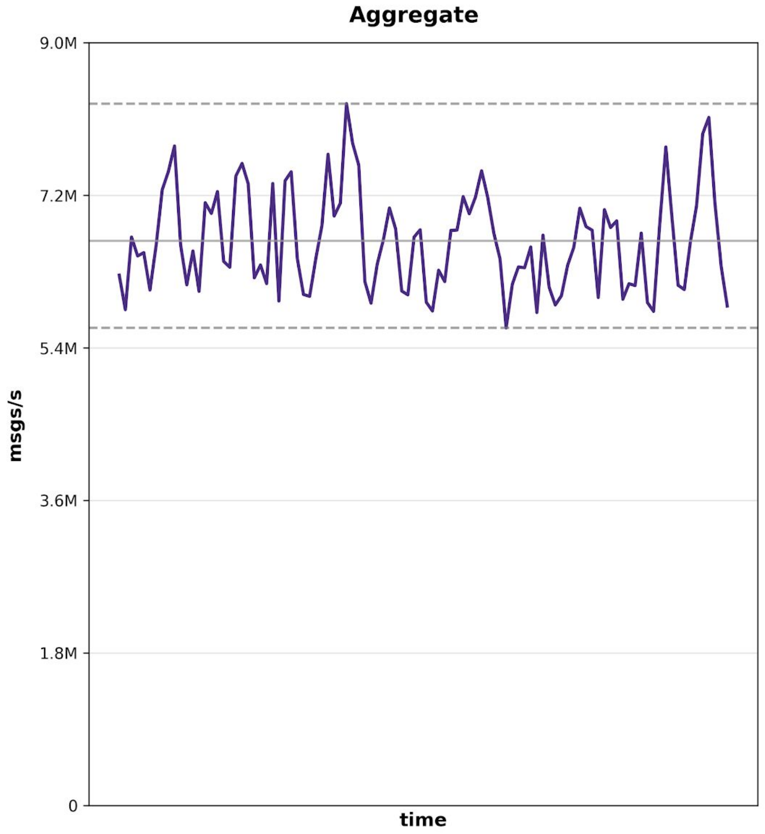
Effect of aggregating decorrelated workloads on net system load (n=81)

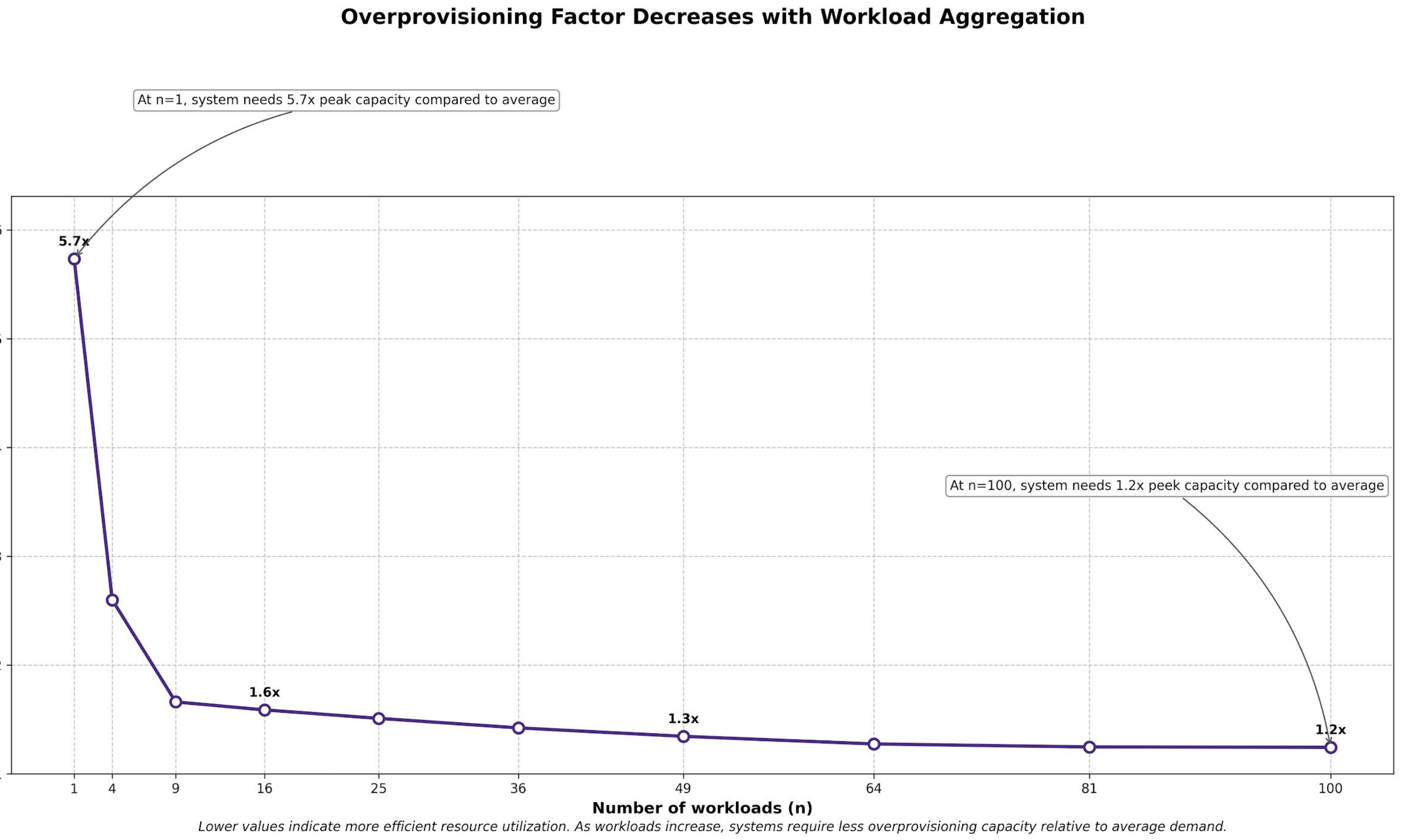




Effect of aggregating decorrelated workloads on net system load (n=100)







Lower values indicate more efficient resource utilization. As workloads increase, system relatively requires less overprovisioning capacity. → Potentially higher profit margin for the service provider.

Correlated workloads

Note that multiple workloads from the same application, customer or industry tend to behave as a single correlated workload. For example, in financial services:

- Daily patterns: Trading volume spikes at market open and closing
- Quarterly earnings reports trigger coordinated processing bursts
- Scheduled economic data releases (e.g., Fed announcements, ECB policy decisions) create predictable traffic surges

Conclusions about workload aggregation

- As workloads increase, system relatively requires less overprovisioning capacity
  - Higher profit margin for the service provider, this is the economies of scale from resource sharing
- There needs to be a sufficient amount of decorrelated work loads
- Some workloads might be needlessly correlated or spiky
  - This can be addressed in many ways

Thoughts around multi-tenancy and shared resources

- There’s a significant benefit in reducing infrastructure cost by resource sharing
- Why aren’t there more truly multi-tenant messaging and database services?
    - The challenges of autoscaling and resource sharing haven’t been addressed for the particular service
    - There’s an alternative good-enough solution by autoscaling infrastructure resources acquired from a cloud provider

Key takeaways & Q&A

- Multi-tenancy & autoscaling
  - Multi-tenancy benefits: shared resources and shared operations
- Service provider and consumer perspectives in Multi-tenant SaaS autoscaling
  - Service Consumer wants to “pay more” for more capacity without thinking about the infrastructure details
- Different SaaS platform types and how it impacts autoscaling
  - Many platform types don’t do resource sharing since there are unsolved challenges or good enough solutions that don’t are sufficient although resource sharing would potentially result in better resource utilization and higher profit margins for the service provider
- Capacity Unit in autoscaling
  - Optimally serverless
- Capacity management challenges in Multi-tenant SaaS
  - Noisy neighbour, long lead time to acquiring more resources
- Noisy Neighbour challenges in Multi-tenant SaaS
  - Unsolved problems in many applications, usually handled by eliminating the problem with resource isolation
  - Solutions for this problem would be a good topic for another talk
- Aggregated workload patterns & economies of scale of resource sharing
  - The relative amount of resource overprovisioning can be reduced when resources are shared, as long as workloads are decorrelated
- Q & A
  - If you come up with questions later, email lari@hotari.net

Related resources

- Elhemali, M., Gallagher, N., Gordon, N., Idziorek, J., Krog, R., Lazier, C., Mo, E., Mritunjai, A., Perianayagam, S., Rath, T., Sivasubramanian, S., Sorenson III, J.C., Sosothikul, S., Terry, D., & Vig, A. (2022). **Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service.** Retrieved from https://www.amazon.science/publications/amazon-dynamodb-a-scalable-predictably-performant-and-fully-managed-nosql-database-service
  - Conference presentation on YouTube
- Ed Huang's blog post series
  - The Road To Serverless: Intro & Why
  - The Road To Serverless: Storage Engine
  - The Road To Serverless: Multi-tenancy
- Jack Vanlightly's blog post series
  - The Architecture of Serverless Data Systems
- Povzner, A., Mahajan, P., Gustafson, J., Rao, J., Juma, I., Min, F., Sridharan, S., Bhatia, N., Attaluri, G., Chandra, A., Kozlovski, S., Sivaram, R., Bradstreet, L., Barrett, B., Shah, D., Jacot, D., Arthur, D., Dagostino, R., McCabe, C., Obili, M. R., Prakasam, K., Sancio, J. G., Singh, V., Nikhil, A., & Gupta, K. (2023). **Kora: A Cloud-Native Event Streaming Platform for Kafka.** Proceedings of the VLDB Endowment, 16(12), 3822-3834. https://doi.org/10.14778/3611540.3611567
- AWS blog posts by David Yanacek
  - Fairness in multi-tenant systems
  - Using load shedding to avoid overload