WoC’21
The Seventh International Workshop on Container Technologies and Container Clouds collocated with Middleware’21, Dec 2021, Quebec, Canada
6 December 2021

On The Design of SLA-Aware and Cost-Efficient Event Driven Microservices

Mazen Ezzeddine[1,2], Sébastien Tauvel[2], Françoise Baude[1], Fabrice Huet[1]
[1]Université Côte d’Azur, CNRS, I3S
Nice, France
[2]HighTech Payment Systems, HPS
Aix En Provence, France
mazen.ezzeddine@univ-cotedazur.fr

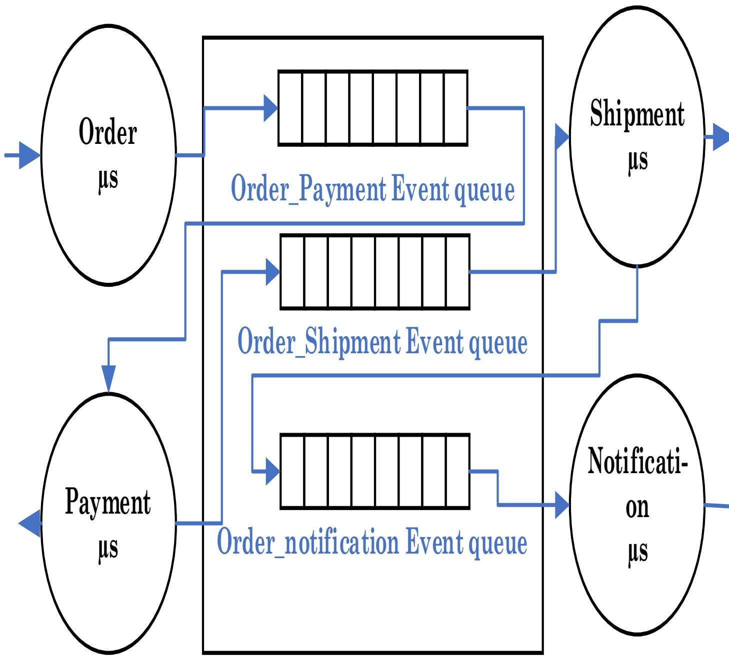
Motivation and Problem statement (1)

- <u>**Event driven microservices**</u> are widely used for architecting scalable cloud software and data systems.
  - Complete isolation and loose coupling of microservices, asynchronous queue-based event driven communication style, in addition to performance gain [1][2][3][4][5].
- <u>**SLA-aware and cost-efficient**</u> event driven microservices to achieve a desired ***tail latency*** for event processing have been rarely studied in the literature [11]
  - <u>**Autonomus scale**</u> in and out of resources to cater to the incoming workload in cost efficient manner
  - Challenges not faced in typical request-response style microservices
    - <u>**Rebalancing**</u> : distribute the load of the events waiting in the queues among the microservice replicas
      - **Consumption blocking** operation that negatively affects the SLA.

[1] K. Marcos , D. Pedro, B. Leonardo, C. Carlos, M. Lemos, A. Darlan, L. Sérgio and Y. Z. Yongluan, "From a monolithic big data system to a microservices event-driven architecture"
[2] P. Das, L. Rodrigo and Z. Yongluan, "HawkEDA: a tool for quantifying data integrity violations in event-driven microservices"
[3] Q. Xiang, P. Xin, H. Chuan, W. Hanzhang, X. Tao, L. Dewei, Z. Gang and C. Yuanfang, "No Free Lunch: Microservice Practices Reconsidered in Industry,"
[4] C. Richardson, "Building microservices: Inter-process communication in a microservices architecture
[5] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce.,"
[11] P. Chindanonda, V. Podolskiy and M. Gerndt, "Self-Adaptive Data Processing to Improve SLOs for Dynamic IoT Workloads,"

Motivation and Problem statement (2)

- Each latency-critical consumer microservice is configured with a **maximum event processing latency**
  - Maximum time an event might exhibit **waiting in the queue and processed by its consumer microservice** without violating the SLA.
- Event driven microservices architecture deployed on **Kubernetes** orchestrator.



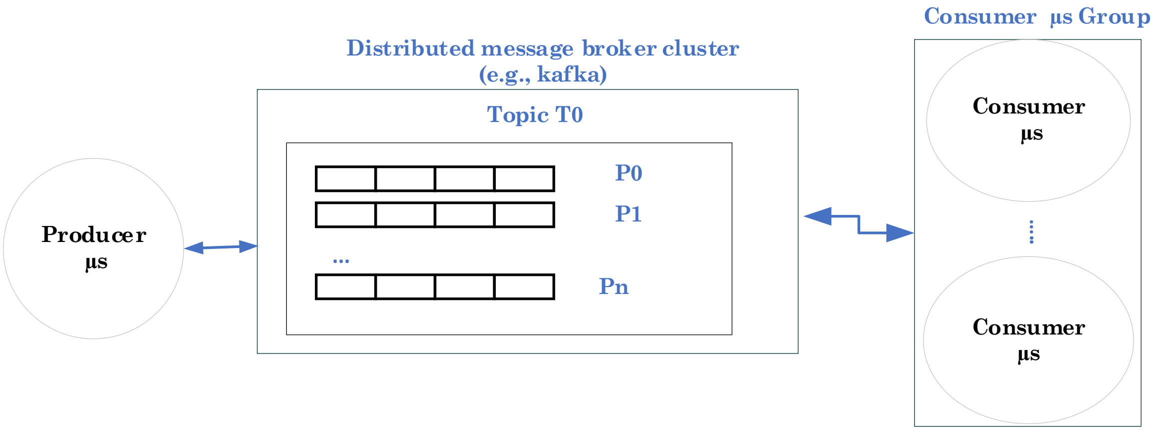
Contributions

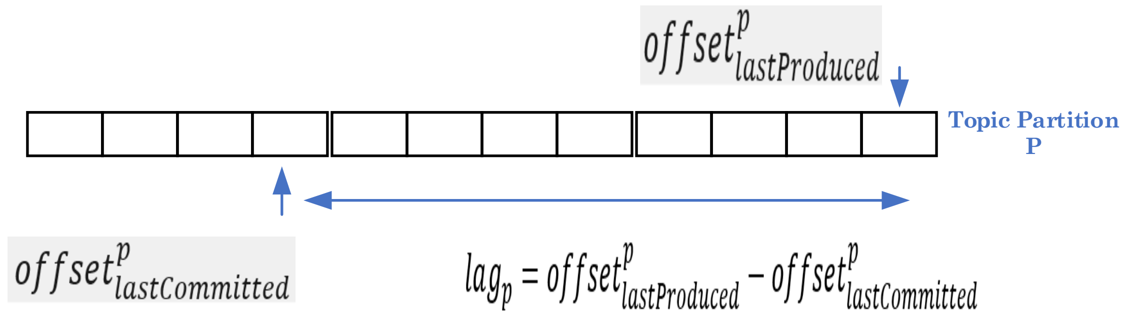
1. Design and implementation of **dynamic horizontal autoscaling framework** to meet a desired **tail latency** for **event processing time**.

2. An autoregressive workload prediction model with **online learning** using *the exponentially weighted recursive least squares algorithm* for proactive autoscaling.

3. Quantitative measurements on the cost *of consumer microservice provisioning time*, and on the cost of *blocking the consumption during rebalancing*

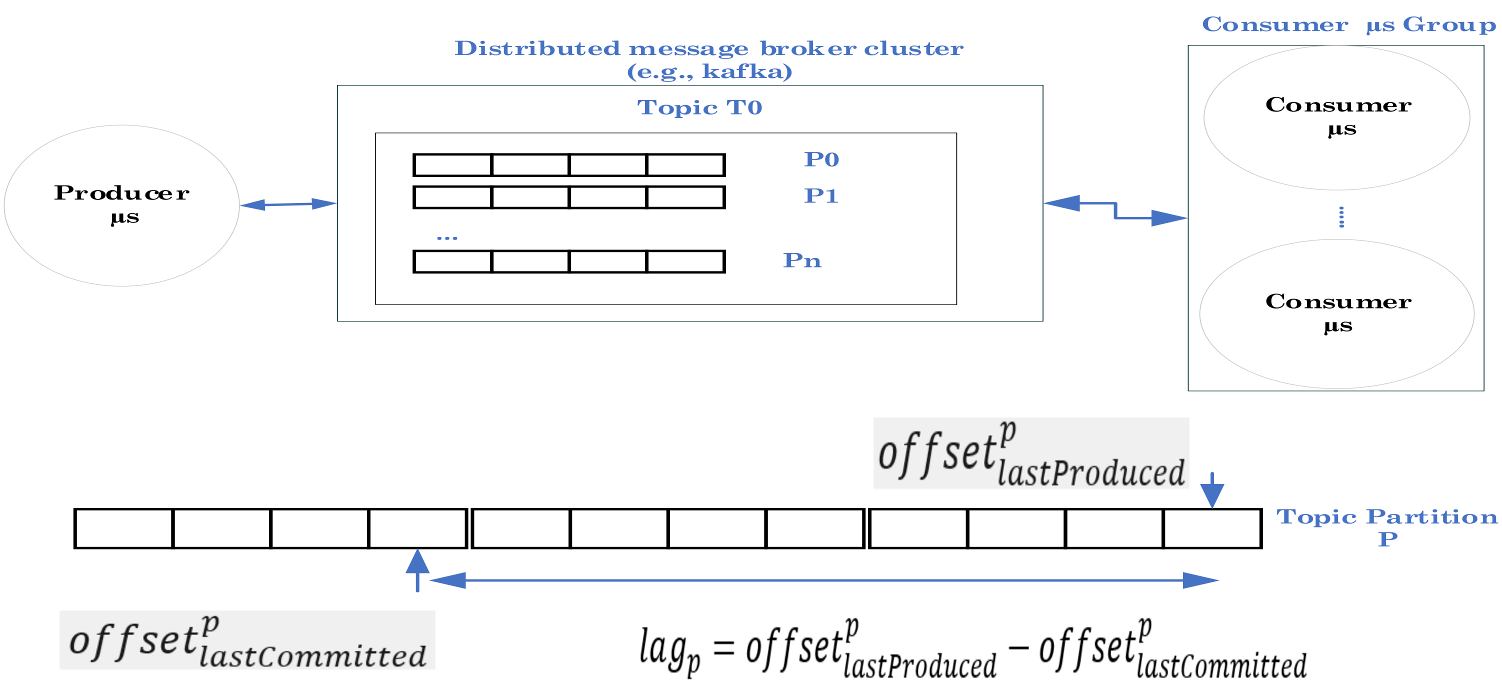
Background (Kafka distributed event broker)
Topic, partition, producer and consumer microservices



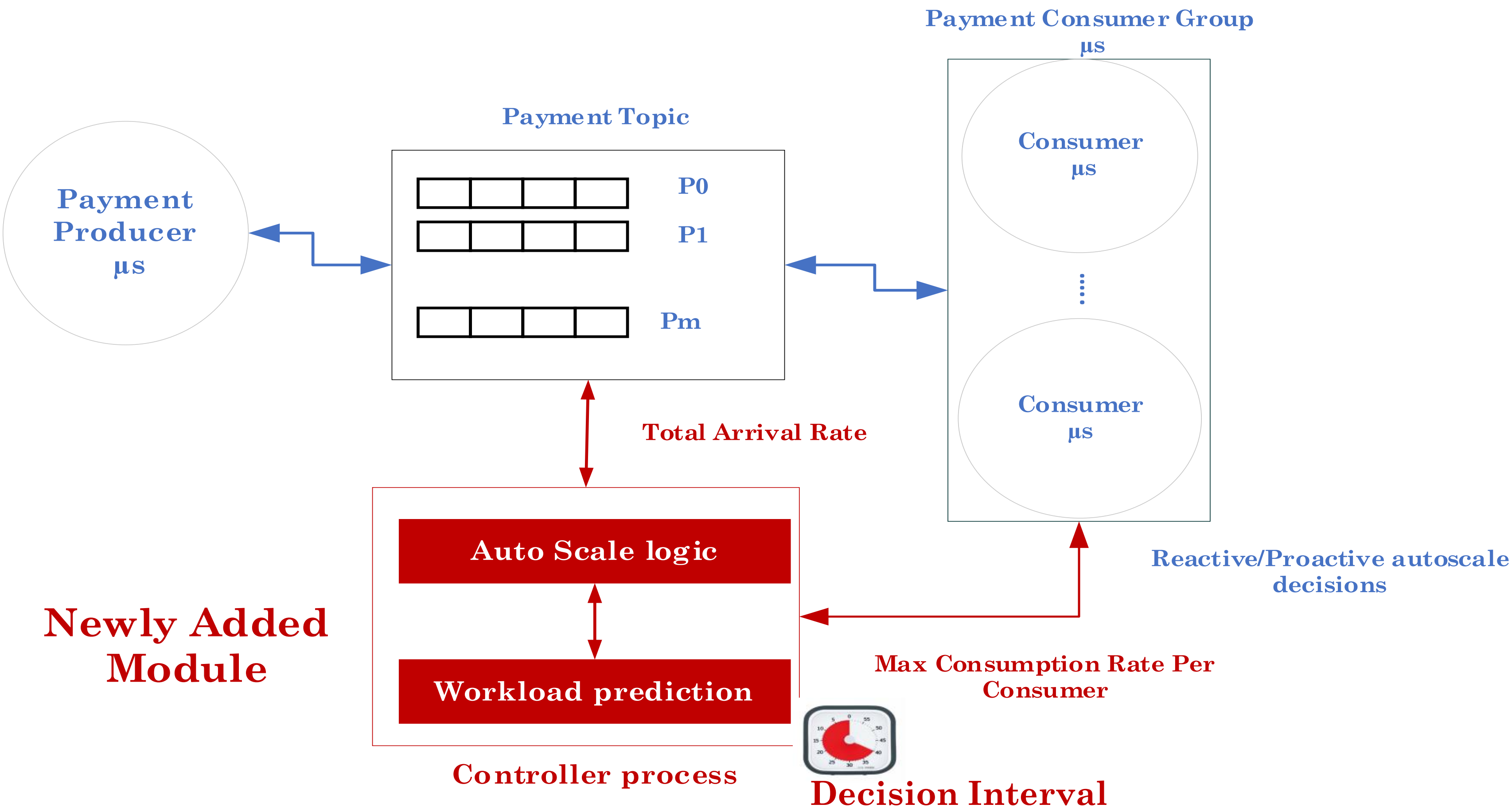


Event Arrival Rate, Event lag, maximum Consumption Rate Per Consumer

- $totalArrivalRate_t = \sum_{p \in partitions} \frac{(offset^p_{lastProduced})_t - (offset^p_{lastProduced})_{t-\delta}}{\delta}$
- $maxConsumptionRatePerConsumer = \frac{\#\ events\ polled\ per\ consumer}{ProcessingTime}$
- $lag_p = offset^p_{lastProduced} - offset^p_{lastCommitted}$
- $lag_{topic} = \sum_{p \in partitions} lag_p$



System Architecture and Kubernetes deployment



Reactive Autoscale

Algorithm 1. Reactive Autoscale

Input *n: current number of consumer microservices*

Input *decisionInterval: time between two successive scaling decisions*

REPEAT FOREVER

Query the broker to get the *totalArrivalRate* into topic and *maxConsumptionRatePerConsumer*

IF *totalArrivalRate >= n * maxConsumptionRatePerConsumer*

Scale up the consumer group by one

ELSE IF *totalArrivalRate < (n-1) * maxConsumptionRatePerConsumer*

Scale down the consumer group by one

SLEEP *decisionInterval*

- When current total arrival rate > total consumption rate : scale up by 1
- If a replica is removed, and current total arrival rate **REMAIN** < total consumption rate : scale down by 1.

UNIVERSITÉ CÔTE D'AZUR

Proactive Autoscale

Algorithm 2. Proactive Autoscale

Input *n: current number of consumer microservices*

Input *decisionInterval: time between two successive scaling decisions*

REPEAT FOREVER

Query the broker to get the *totalArrivalRate* into topic and *maxConsumptionRatePerConsumer*

IF *totalArrivalRate >= n × maxConsumptionRatePerConsumer* **OR** ***predicted****(totalArrivalRate) >= n × maxConsumptionRatePerConsumer*

Scale up the consumer group by one

ELSE IF *totalArrivalRate < (n-1) × maxConsumptionRatePerConsumer* **AND** ***predicted****(totalArrivalRate) < (n-1) × maxConsumptionRatePerConsumer*

Scale down the consumer group microservices by one

SLEEP *decisionInterval*

- When current **OR predicted** total arrival rate > total consumption rate : scale up by 1
- If a replica is removed, and current **AND predicted** total arrival rate **REMAIN** < total consumption rate : scale down by 1.

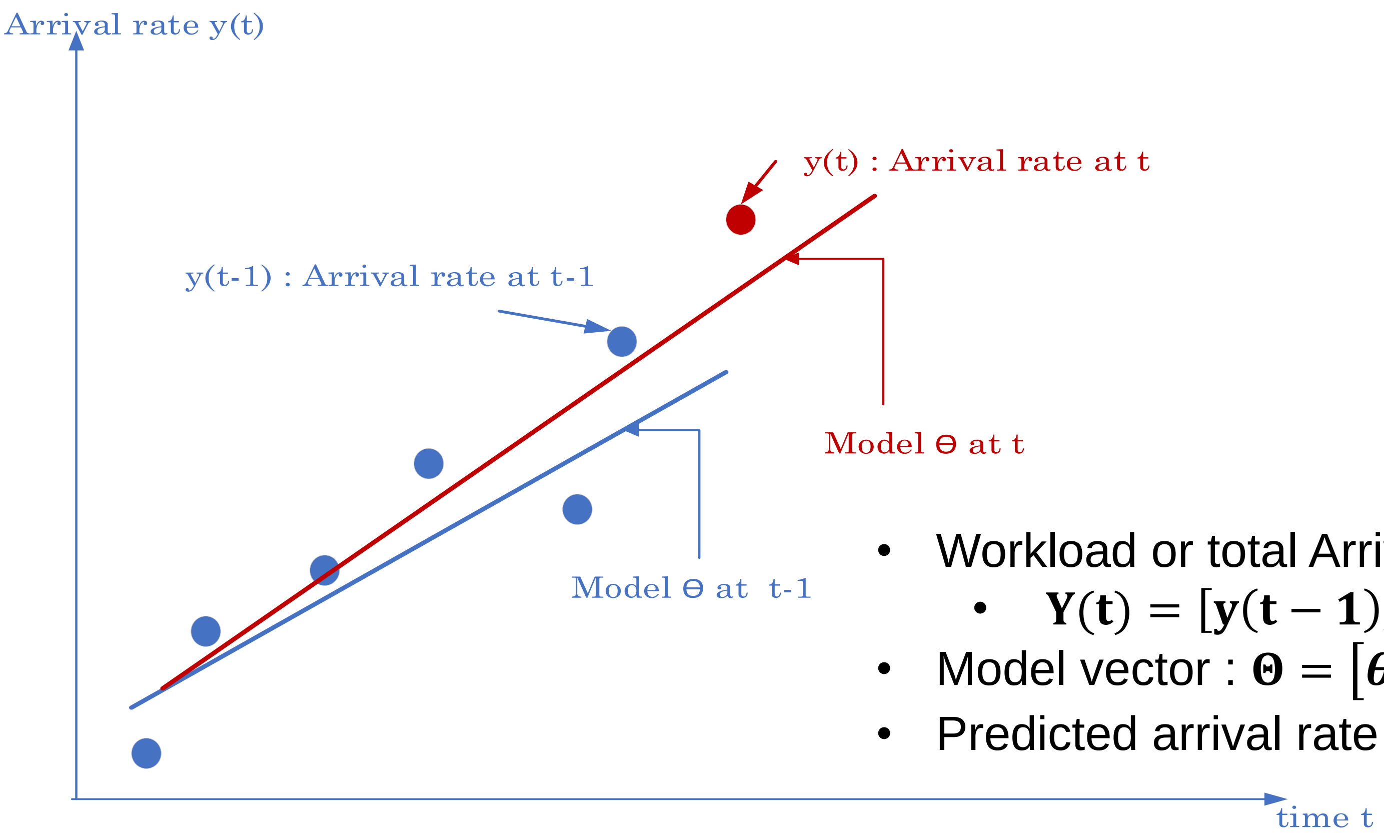
Arrival rate prediction using Recursive Least Squares RLS

- **Autoregression (AR)**, **autoregression moving average (ARMA)** and **autoregression integrated moving average (ARIMA)** are widely used in cloud computing for time series (workload) prediction [29]

- **AR, ARMA and ARIMA** can be **trained online** using the exponentially weighted **recursive least squares RLS** algorithm
  - RLS is an extension to the classical least-square (LS) targeted towards real time applications where data arrives sequentially to the system
  - Exponentially weighted RLS *ewRLS* incorporates a **forgetting factor** that discounts older data to make the model representative for the most recent state of the workload

[29] C. Qu, . R. N. Calheiros and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey.,"

Exponentially weighted Recursive Least Squares Algorithm (ewRLS)



- Workload or total Arrival rate vector :
  - $\mathbf{Y(t)} = [\mathbf{y(t-1)}, \ \dots, \ \mathbf{y}(\mathbf{t}-\boldsymbol{p})]$
- Model vector : $\boldsymbol{\Theta} = [\boldsymbol{\theta_1} \dots, \boldsymbol{\theta_p}]$
- Predicted arrival rate $\hat{\boldsymbol{y}}(\boldsymbol{t}) = \ \boldsymbol{\Theta}(\mathbf{t-1})^{\boldsymbol{T}}\mathbf{Y(t)}$

- At time t, when y(t) [the arrival rate at time t] is observed update the model $\Theta(t)$ using the equations shown in Algorithm 3 (see backup slides).

Seamless switching from reactive to proactive provisioning

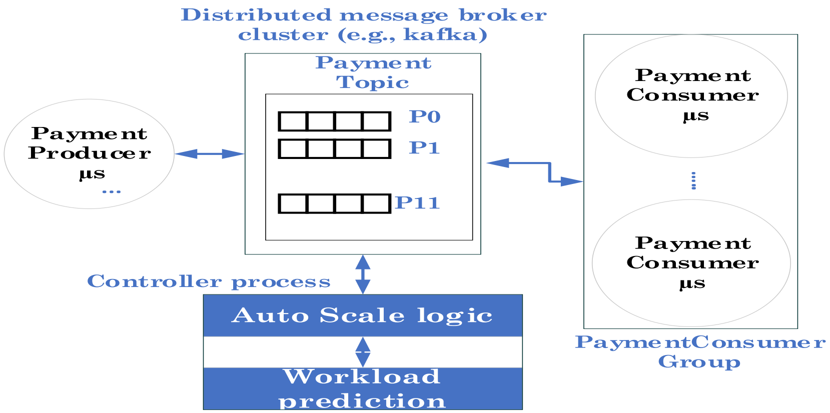
- At each decision interval t (a new observation of arrival rate, y(t) is available) calculate the mean relative prediction accuracy ***Acc***
  - The average relative prediction accuracy over all the past predictions
  - *Acc* metric also used in [30] mean elasticity index MEI

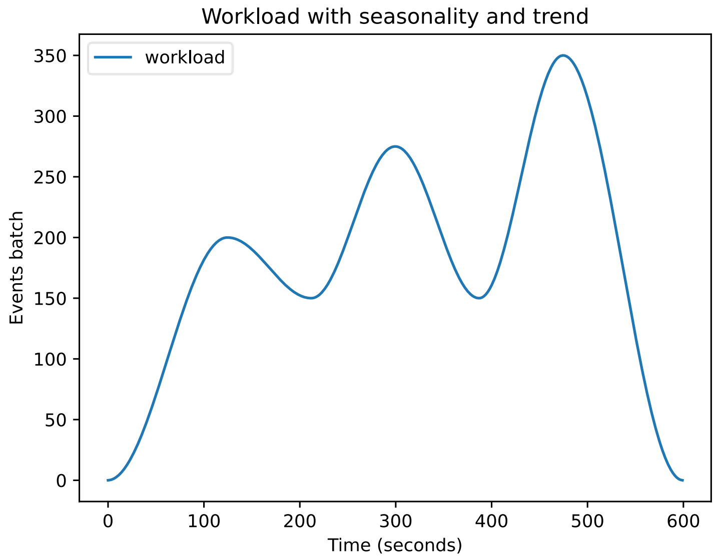
- When the model achieves a configurable value for *ACC* (default 0.85) the model switches to proactive provisioning.
  - Currently, once proactive mode is activated, the reactive mode is not re-activated again (e.g., if *ACC* drops below its configured value)

[30] V. R. Messias, J. C. Estrella, , R. Ehlers, Santana, M. J. Santana, R.C. Santana, and S. Reiff-Marganiec, S., "Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure".

Methodology and Results

- Google Cloud using a GKE Kubernetes cluster of 4 VMs.
- 10 minutes workload: the batch of events sent to the broker each second. (Adapted from [11])
- Payment processing consumer microservice:
  - Maximum event processing latency of 5 seconds
- Peek provisioning: 4 consumer microservices operating at 100 events/second
  - zero events violated the latency SLA.
  - 40 replica.minutes



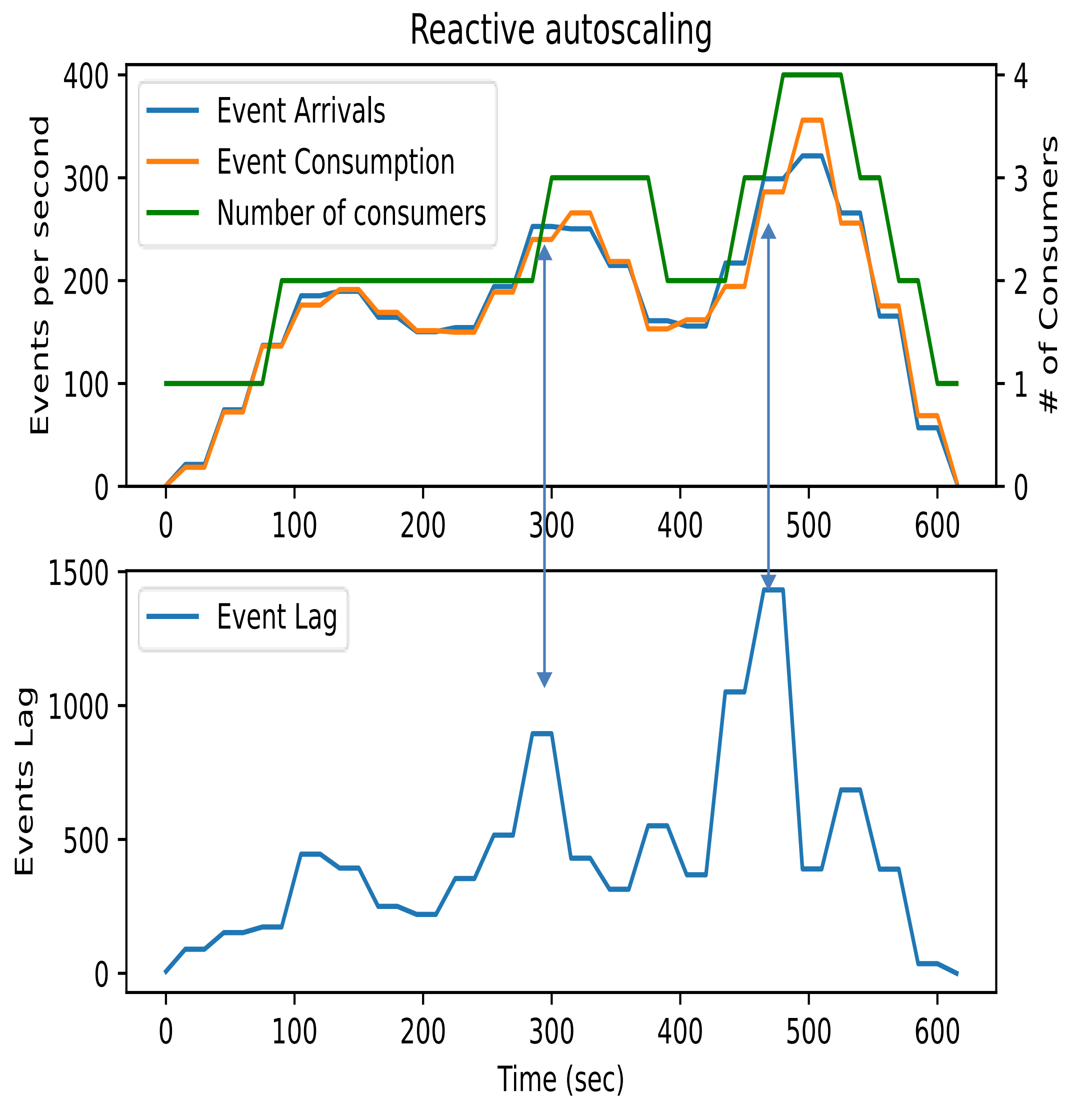


[11] P. Chindanonda, V. Podolskiy and M. Gerndt, "Self-Adaptive Data Processing to Improve SLOs for Dynamic IoT Workloads,"

Reactive Autoscaling

- Payment microservices operating at 100 events/sec, maximum event processing latency of 5 seconds

- 97.2% SLA guarantee (2.8% violation) at the cost of 23.8 replica.minutes
  - 59.5% reduction in the cost of consumer microservices compared to peek load provisioning.

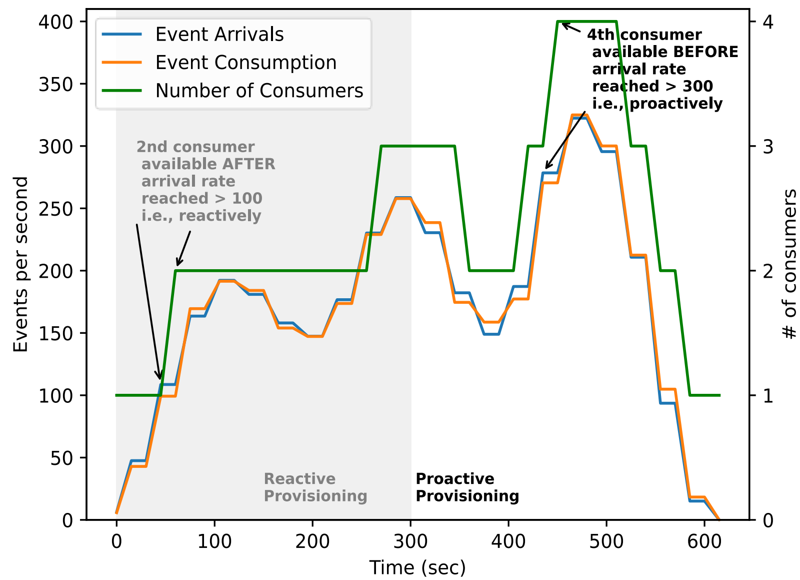


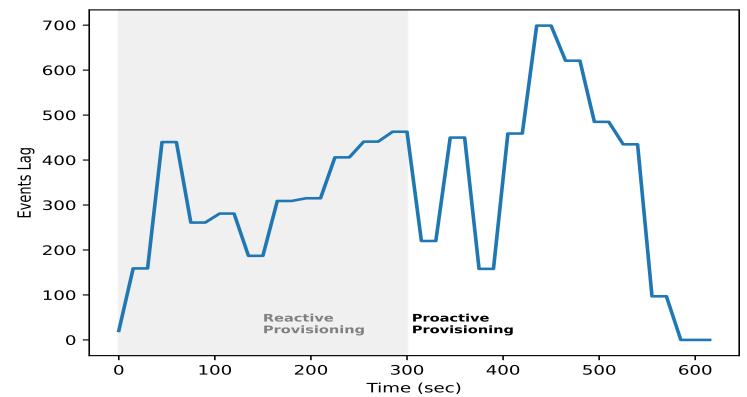
Why only 97.2% SLA guarantee? Can we do better?

- Delay until the payment microservice is ready and can start serving payment events
  - $\boldsymbol{total\ delay\ = provisioning\ time + joining\ time + rebalancing\ time}$
- Provisioning time :
  - Time required so that Kubernetes launches a consumer instance and the instance probed ready
    - [1, 3] seconds
- Joining time :
  - Time so that all consumer microservices in the group are aware that a rebalancing shall take place.
  - Up to 3 seconds (default heartbeat interval)
- Rebalancing time :
  - Actual assignment of topic partitions to the consumer microservices in the group
  - A consumption blocking operation that takes [1, 3] seconds
- **Maximum** total delay from SLA detection till the newly added microservice starts consumption = 9 seconds.

From Reactive to Proactive Autoscaling

- Autoscaling started in reactive mode and switched to proactive at time 300 seconds.
- Event processing latency SLA of 5 seconds, 100 events/sec consumers
  - 2.3% events violated the SLA (97.7% SLA guarantee)
- Observations:
  - The experiment ran reactively for 50% of the time.
  - Proactive autoscaling :
    - Can help in **Provisioning time**
    - Can not help in the blockage of events consumption flow resulting out of consumer group rebalancing. **(rebalancing time)**





Incremental non-blocking rebalancing

- Given the negative impact of blocking rebalancing on latency SLAs, Kafka recently introduced incremental (nonstop of the world) rebalancing
  - A non-blocking continual flow version of rebalancing.
  - Consumption of events **will block only** for those partitions that will be **reassigned** to other consumer microservices.
  - Experiments omitted from the paper due to space limitations.

Conclusion and ongoing work

- A framework for cost-efficient tail latency SLA guarantee of event driven microservices.

- Further **design space exploration** under larger scale deployments and more realistic workload traces.
  - Consumer microservices running on heterogenous servers, that is, having different consumption rate.

- Tackling a **pipeline** of event driven microservices instead of single producer consumer microservices.

- Investigation of the case of **stateful** consumer microservices.

Thank You

Questions?

Backup slides

Exponentially weighted Recursive Least Squares Algorithm (ewRLS)

Algorithm 3. Exponentially weighted Recursive Least Squares Algorithm (ewRLS) [16][17]

Input $\boldsymbol{p}$: history length of arrival rate vector used to forecast

Input $\boldsymbol{\lambda}$: forgetting factor ($\lambda = 0.98$)

Input P(0): positive definite matrix e.g., *P(0)= I* identity matrix

$\boldsymbol{Input\ Y(1)} = [y(0), y(-1), \dots, y(-p+1)]$ initial random values of arrival rate vector

$\mathbf{Input\ \Theta(0)} = \left[\theta_0, \theta_1 \dots, \theta_{p-1}\right]$ initial random values of model vector

For t= 1 to INFINITY do (loop every time a new sample of arrival rate is available)

$$Y(t) = [y(t-1), y(t-2), \dots, y(t-p)]$$

$$K_t = \frac{P(t-1)\,Y(t)}{\lambda \ + \ Y^T(t)P(t-1)\,Y(t)}$$

$$P_t = \frac{1}{\lambda}[P(t-1) - \frac{P(t-1)\,Y(t)Y^T(t)P(t-1)}{\lambda + Y^T(t)P(t-1)\,Y(t)}]$$

$$\Theta(t) = \Theta(t-1) + K_t[y(t) - \hat{y}(t|\Theta(t-1))]$$

End For

References

[1] R. Laigner, K. Marcos , D. Pedro, B. Leonardo, C. Carlos, M. Lemos, A. Darlan, L. Sérgio and Y. Z. Yongluan, "From a monolithic big data system to a microservices event-driven architecture," *IEEE 46th Euromicro Conference on Software Engineering and Advanced Applications,* pp. 213-220., 2020.

[2] P. Das, L. Rodrigo and Z. Yongluan, "HawkEDA: a tool for quantifying data integrity violations in event-driven microservices," *Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems,* pp. 176-179, 2021.

[3] Q. Xiang, P. Xin, H. Chuan, W. Hanzhang, X. Tao, L. Dewei, Z. Gang and C. Yuanfang, "No Free Lunch: Microservice Practices Reconsidered in Industry," *arXiv preprint arXiv:2106.07321,* 2021.

[4] C. Richardson, "Building microservices: Inter-process communication in a microservices architecture," 24 July 2015. [Online]. Available: https://www.nginx.com/blog/building-microservices-inter-process-communication/. [Accessed 11 September 2021].

[5] A. Bellemare, Building Event-Driven Microservices, O'Reilly Media, Inc., 2020.

[6] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce.," *EEE International Conference on Software Architecture Workshops (ICSAW),* pp. 243-246, 2017.

[7] N. Dragoni, G. Saverio, A. L. Lafuente, M. Manuel, M. Fabrizio, R. Mustafin and L. Safina, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering,* pp. 195-216, 2017.

[8] G. Yu, P. Chen and Z. Zheng, "Microscaler: Cost-effective scaling for microservice applications in the cloud with an online learning approach," *IEEE Transactions on Cloud Computing,* 2020.

[9] B. Choi,, C. Byungkwon, J. Park, C. Lee and D. Han, "pHPA: A Proactive Autoscaling Framework For Microservice Chain," in *5th Asia-Pacific Workshop on Networking (APNet 2021). Association for Computing Machinery, Inc*, 2021.

[10] G. Yu, P. Chen, H. Chen, G. Zijie , Z. Huang, L. Jing, T. Weng, X. Sun and X. Li, "MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments.," *Proceedings of the Web Conference 2021,* pp. 3087-3098, 2021.

[11] P. Chindanonda, V. Podolskiy and M. Gerndt, "Self-Adaptive Data Processing to Improve SLOs for Dynamic IoT Workloads," *Computers,* vol. 9, no. 1, p. 12, 2020.

[12] N. Narkhede, G. Shapira and T. Palino, Kafka: the definitive guide: real-time data and stream processing at scale, O'Reilly Media, Inc., 2017.

[13] G. Shapira , T. Palino, R. Sivaram and K. Petty, Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale, second edition, O'Reilly Media, Inc., 2021.

[14] S. BLEE-GOLDMAN, "From Eager to Smarter in Apache Kafka Consumer Rebalances," Confluent, 11 5 2020. [Online]. Available: https://www.confluent.io/blog/cooperative-rebalancing-in-kafka-streams-consumer-ksqldb/. [Accessed 11 9 2021].

[15] M. S. Aslanpour, S. S. Gill and A. . N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things,* vol. 12, 2020.

[16] A. H. Sayed, Adaptive filters, John Wiley & Sons, 2011.

[17] L. Weifeng, J. C. Principe and S. S. Haykin, Kernel adaptive filtering: a comprehensive introduction, Wiley, 2010.

[18] S. A. Baset, "Cloud SLAs: present and future," *ACM SIGOPS Operating Systems Review,* vol. 46, no. 2, pp. 57-66, 2012.

[19] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," *ACM SIGPLAN Notices,* vol. 49, no. 4, pp. 127-144, 2014.

[20] J. Bar, "New AWS Auto Scaling – Unified Scaling For Your Cloud Applications," Amazon, 16 January 2018. [Online]. Available: https://aws.amazon.com/autoscaling/. [Accessed 3 May 2021].

[21] "Kubernetes Horizontal Pod Autoscaler," 9 September 2021. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. [Accessed 2021 September 2021].

[22] C. Qu, R. N. Calheiros and R. Buyya, "A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances," *Journal of Network and Computer Applications, Elsevier,* vol. 65, pp. 167-180, 2016.

[23] M. Ghobaei-Arani, S. Jabbehdari and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," *Future Generation Computer Systems,* vol. 78, pp. 191-210, 2018.

[24] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu and J. Chen, "A cost-aware auto-scaling approach using the workload prediction in service clouds.," *Information Systems Frontiers,* vol. 16, no. 1, pp. 7-18, 2014.

[25] N. Roy, A. Dubey and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," *IEEE 4th International Conference on Cloud Computing,* pp. 500-507, 2011.

[26] S. Islam, J. Keung, K. Lee and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems,* vol. 28, no. 1, pp. 155-162, 2012.

[27] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu and R. Buyya, "Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions," *arXiv preprint arXiv:2106.12739,* 2021.

[28] S. C. Hoi, D. Sahoo, L. Jing and P. Zhao, "Online learning: A comprehensive survey," *Neurocomputing,,* pp. 249-289, 2021.

[29] C. Qu, . R. N. Calheiros and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey.," *ACM Computing Surveys (CSUR),* vol. 51, no. 4, 2018.

[30] V. R. Messias, J. C. Estrella, , R. Ehlers, Santana, M. J. Santana, R.C. Santana, and S. Reiff-Marganiec, S., "Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure". *Neural Computing and Applications*, 27(8), 2383-2406. 2016