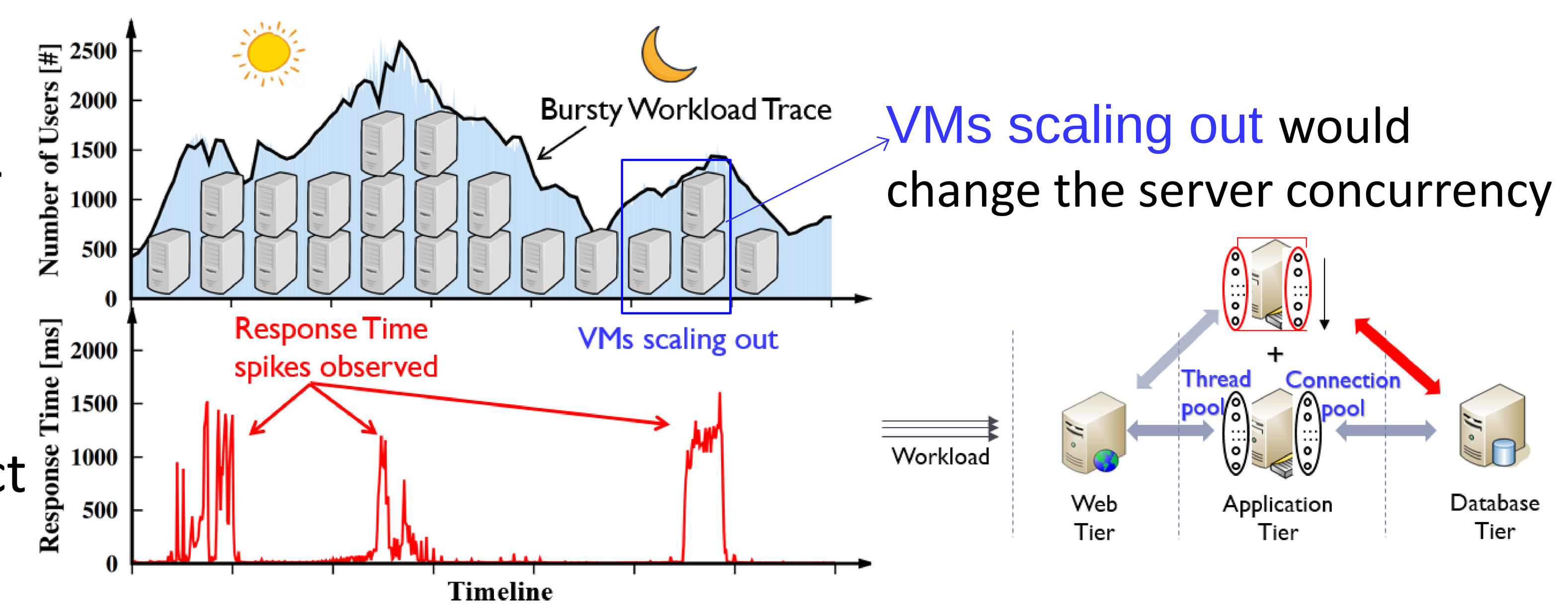
Coordinating Fast Concurrency Adapting with AutoScaling for SLO-Oriented Web Applications

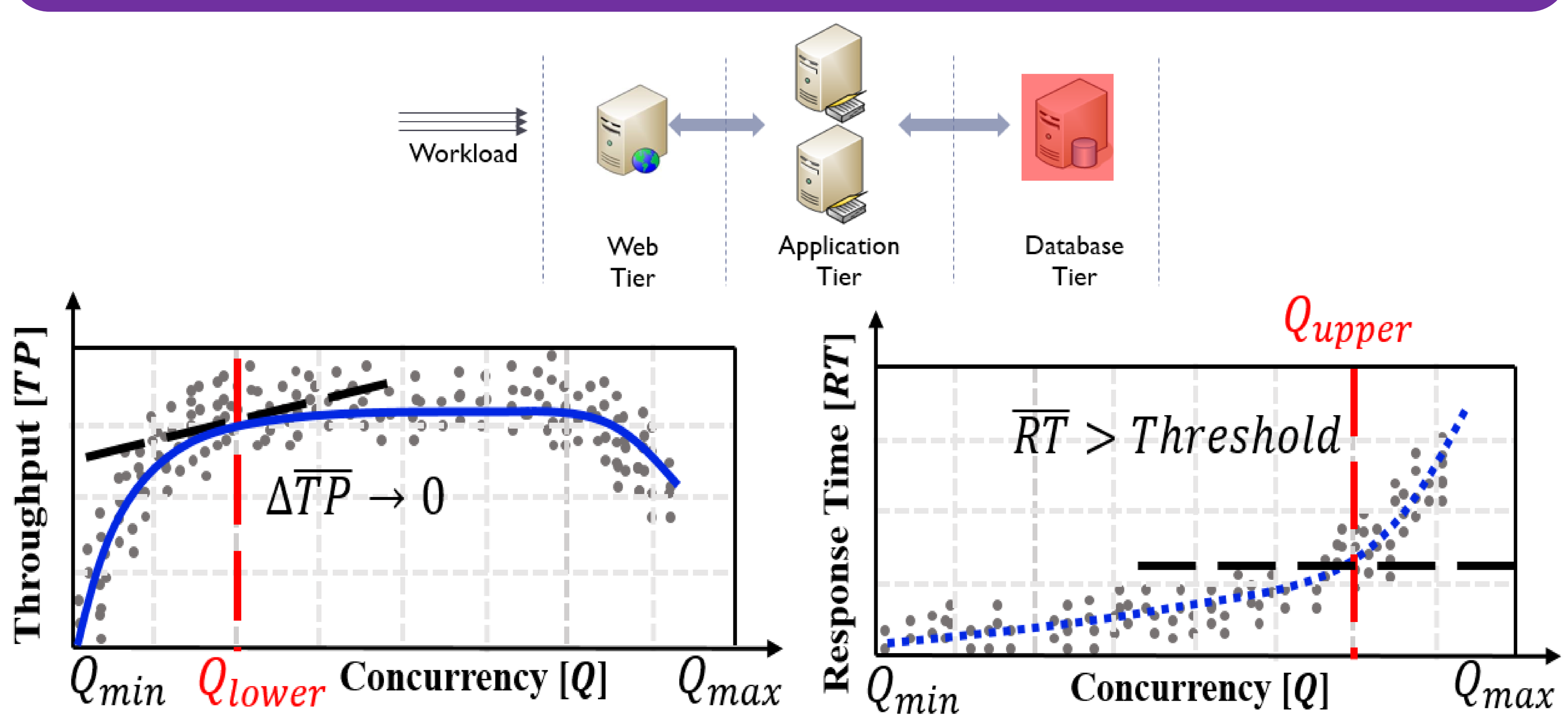
Jianshu Liu, Louisiana State University

AutoScaling: Achieve Good Performance and High Resource Efficiency

- **Cloud computing platforms support Automatically Scaling (AutoScaling) a web application to match the naturally bursty workload.**
  - For example, Amazon prepares more servers to handle over 10X larger customers over Black Friday than in regular periods.
- **Effectively scaling a web application is challenging:**
  - Strict Service Level Objectives (SLO), e.g., response time < 300ms.
  - Soft resources (e.g., server threads/connections) allocation also impact system performance besides adding new servers.

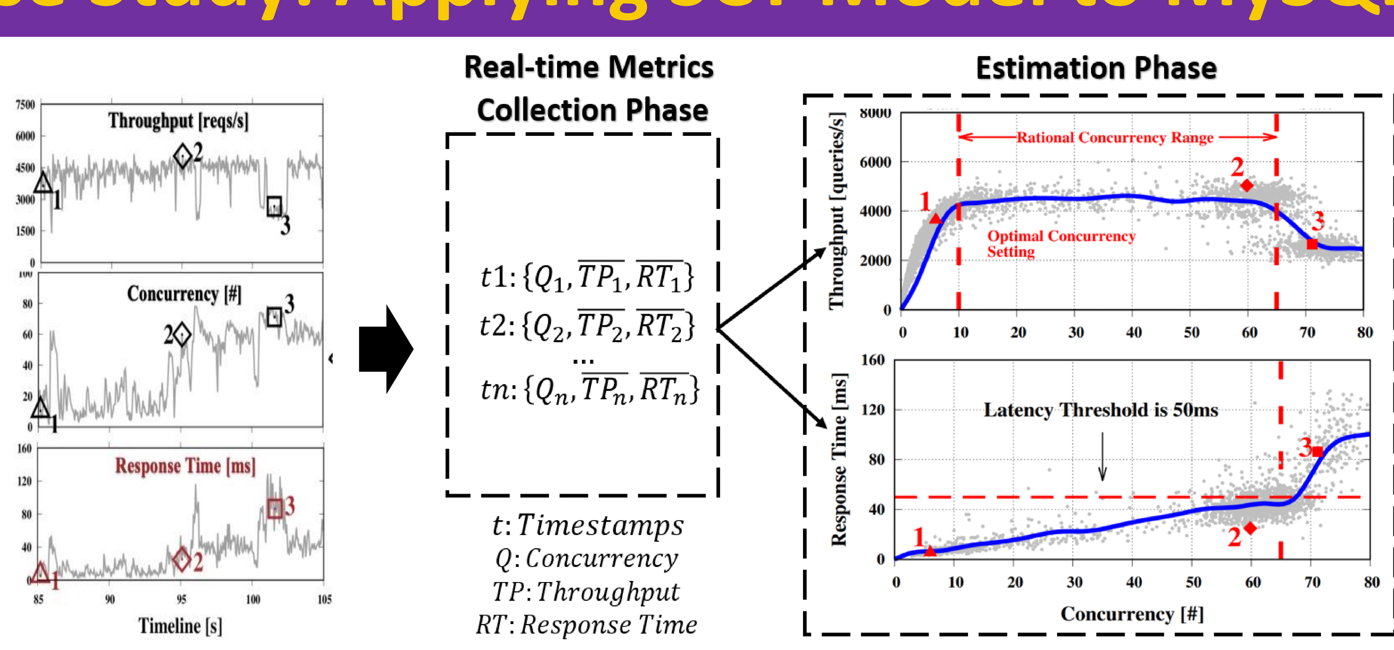


Real-time Online Scatter-Concurrency-Throughput (SCT) Model



- **Real-time Metrics Collection Phase**
  - Collect a server's real-time concurrency, throughput and response time as a tuple measured at a fine granularity (e.g., 50ms) during a short time period (e.g., 3 minutes).
  - Extract the main sequence curve from the scatter graph.
- **Rational Concurrency Range Estimation Phase**
  - Estimate rational concurrency range $[Q_{lower}, Q_{upper}]$ based on statistical intervention analysis and latency threshold.
  - We select the Q_{lower} as the optimal concurrency setting since we make a tradeoff to guarantee a low response time.

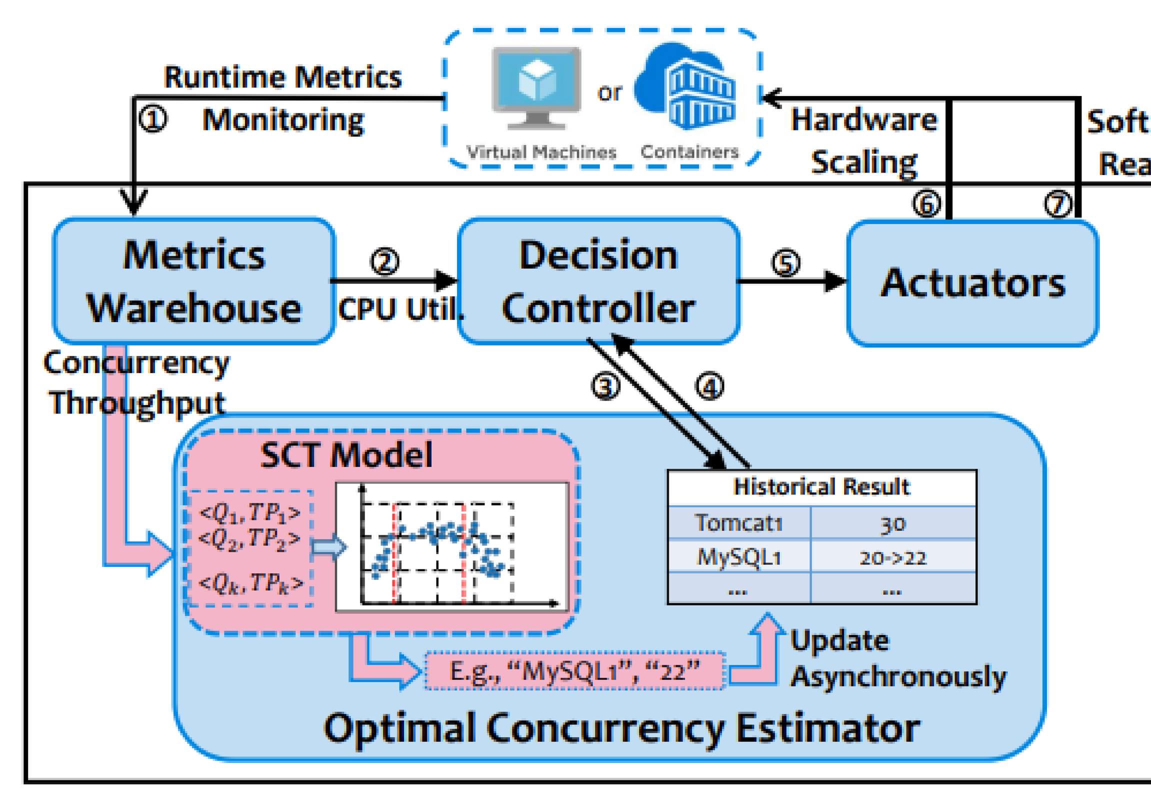
Case Study: Applying SCT Model to MySQL



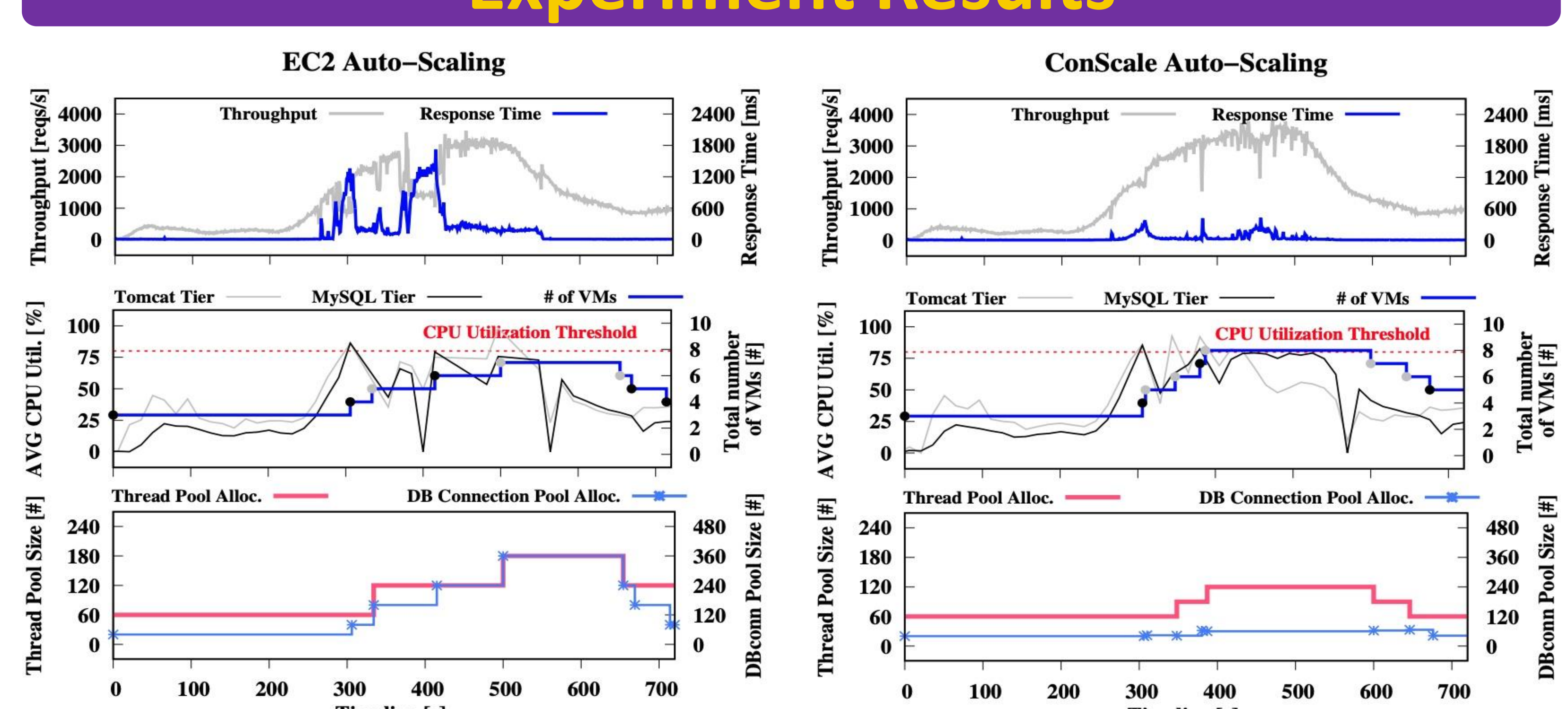
- Our SCT model indicates the rational MySQL concurrency range is [10, 65], which can achieve the highest throughput and satisfy SLO requirement (i.e., RT < 50ms).
- We choose the lower bound of such rational range (i.e., 10) as the optimal MySQL concurrency setting.

Our Solution: Integrating SCT Model to System Scaling Design (ConScale)

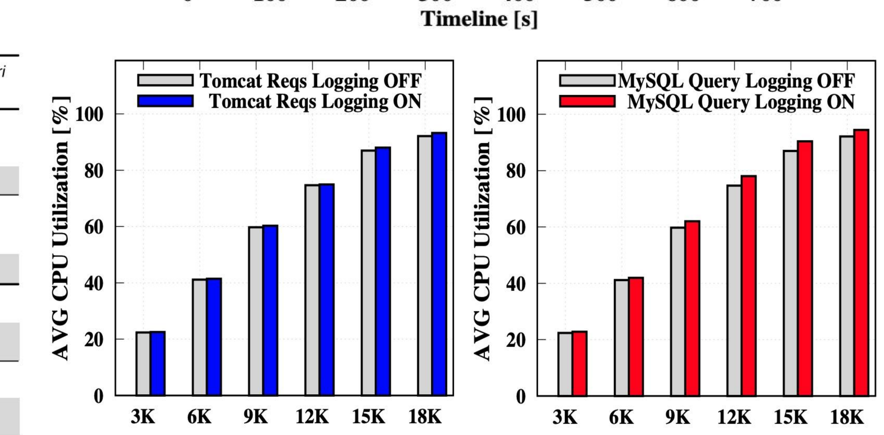
- **ConScale Framework Design**



Experiment Results



		Percentile Response time [ms]	*Large Variation*	*Quick Varying*	*Slowly Varying*	*Big Spike*	*Dual Phase*	*Steep Tri Phase*
VM-based AutoScaling	RT_{95th}	EC2-AutoScaling	462	157	1135	687	225	101
		DCM	274	92	157	367	192	75
		ConScale	**157**	**48**	**85**	**179**	**81**	**56**
	RT_{99th}	EC2-AutoScaling	2345	684	3252	3981	1153	1259
		DCM	1080	443	1499	1376	606	537
		ConScale	**465**	**229**	**218**	**479**	**328**	**171**
Container-based AutoScaling	RT_{95th}	Kubernetes-HPA	213	120	276	252	121	164
		ConScale	**79**	**36**	**56**	**60**	**85**	**49**
	RT_{99th}	Kubernetes-HPA	1004	707	1534	1103	962	876
		ConScale	**266**	**121**	**276**	**246**	**187**	**159**



- ConScale helps EC2-AutoScaling mitigate the large response time fluctuations. (Kubernetes-HPA and DCM also compared)
- ConScale can restrict the 95th and 99th response time below 500ms under six categories of workload traces.
- ConScale only causes a maximum 4.82% CPU overhead at peak workload.

Conclusion

- Effectively autoscaling is difficult due to strict SLO requirements of e-commercial web applications and complex soft resources tuning.
- Implement the ConScale framework to realize fast and intelligent soft resources adaption based on our online SCT model to handle temporary overloading in system scaling scenarios in clouds.
- Our ConScale can help various large-scale systems effectively maintain a stable response time and satisfy SLO requirements.

Contact

Jianshu Liu
Department of Computer Science and Engineering
Email: jliu96@lsu.edu

This work was accepted by IEEE Transactions on Parallel and Distributed Systems (TPDS) on February 14, 2022