Rhythm: Component-distinguishable Workload Deployment in Datacenters

Laiping Zhao[1], Yanan Yang[1], Kaixuan Zhang[1],

Xiaobo Zhou[1], Tie Qiu[1], Keqiu Li[1], Yungang Bao[2]

[1]Tianjin University, [2]Inst. Of Computing Technology, CAS

College of Intelligence and Computing

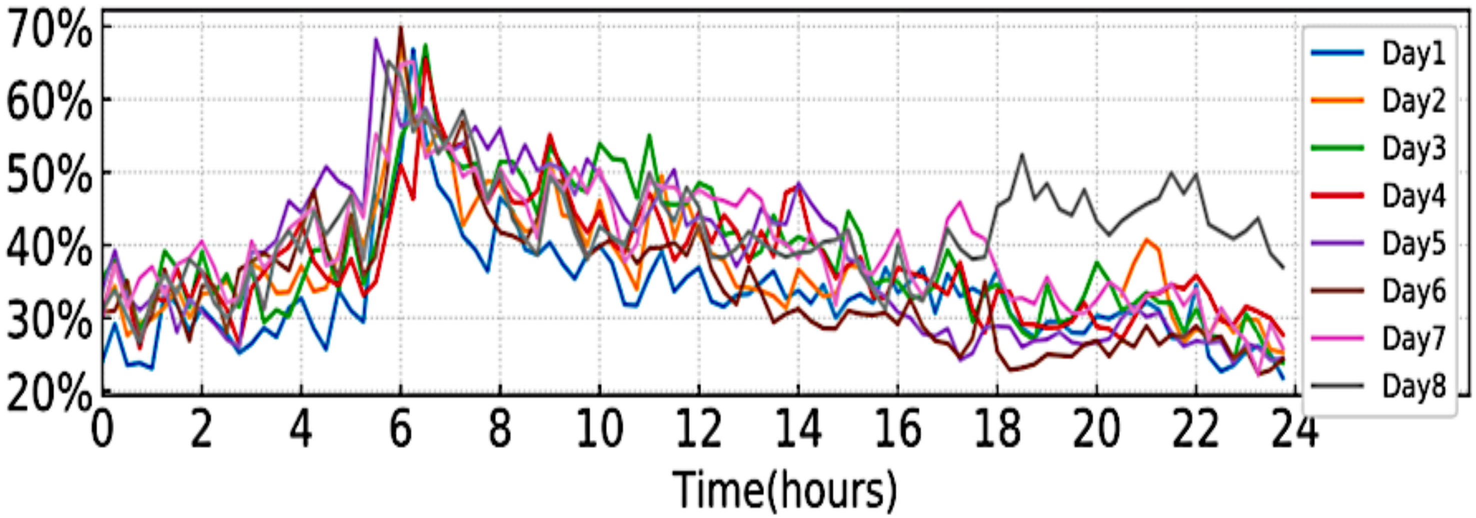
TANKLab Tianjin Key Laboratory of Advanced Networking

Outline

- Background
- Interference on LC components
- Rhythm Controller
- Experimental Evaluation
- Conclusion

Background

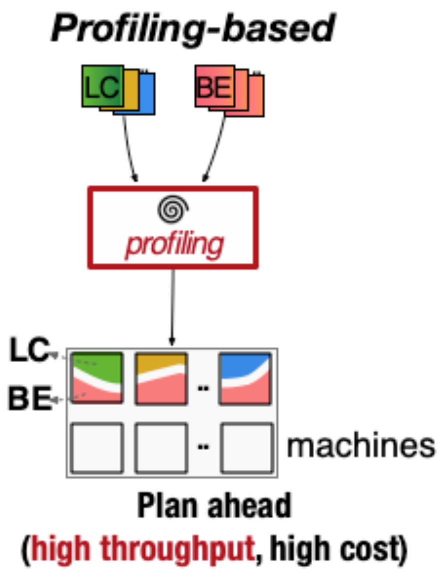
Low Resource Utilization of Datacenter

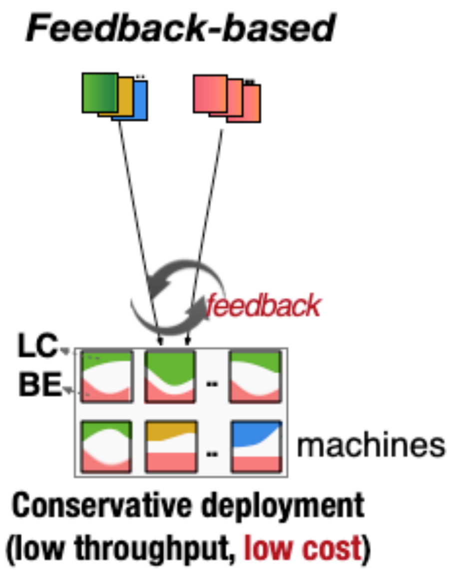


- Aliyun: The average CPU utilization of co-located cluster approaches to 40% [Guo, 2019].
- Improved, but still low utilization.

Background

- Co-location: Improving the resource utilization
- Interference causes unpredictable latency.



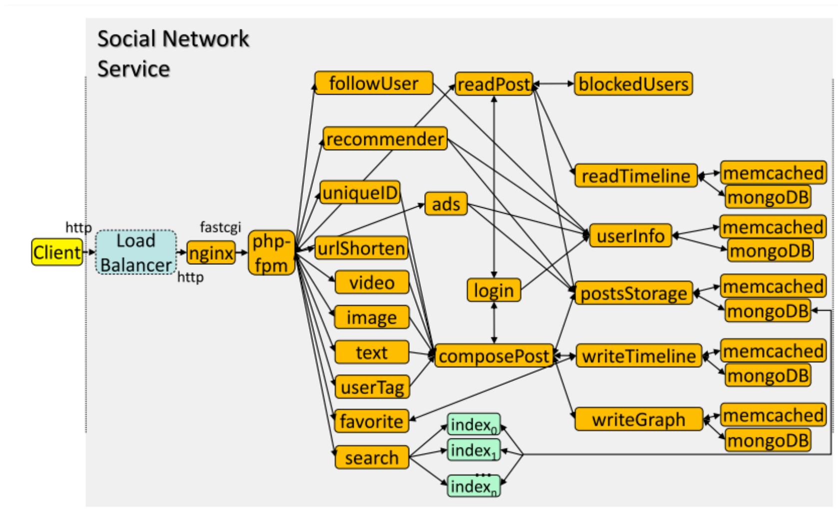


- Profiling of the workload.
- Schedule in a cross-complementing way.

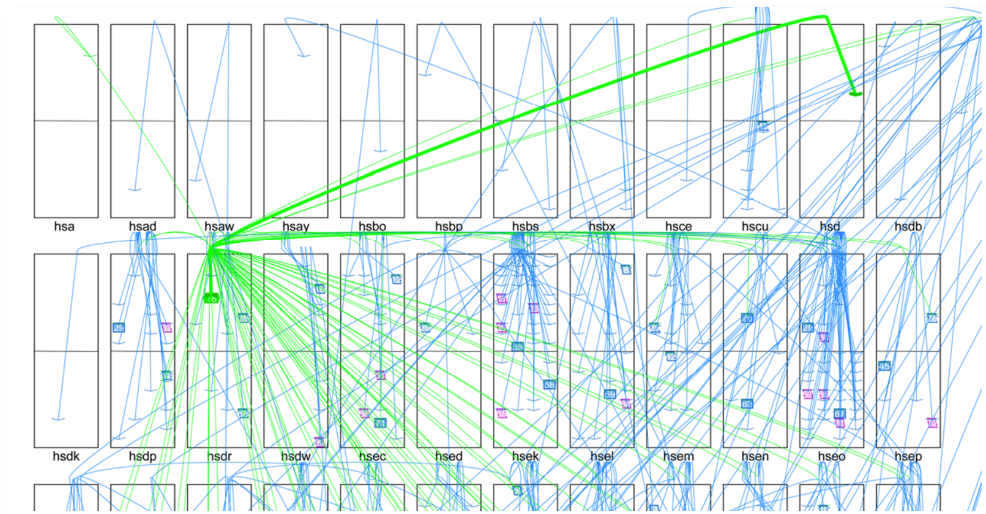
- Real-time monitoring;
- Passive adjustment on resource allocation.

Background

Many-component Services:



Source: [Deathstarbench, ASPLOS’19]



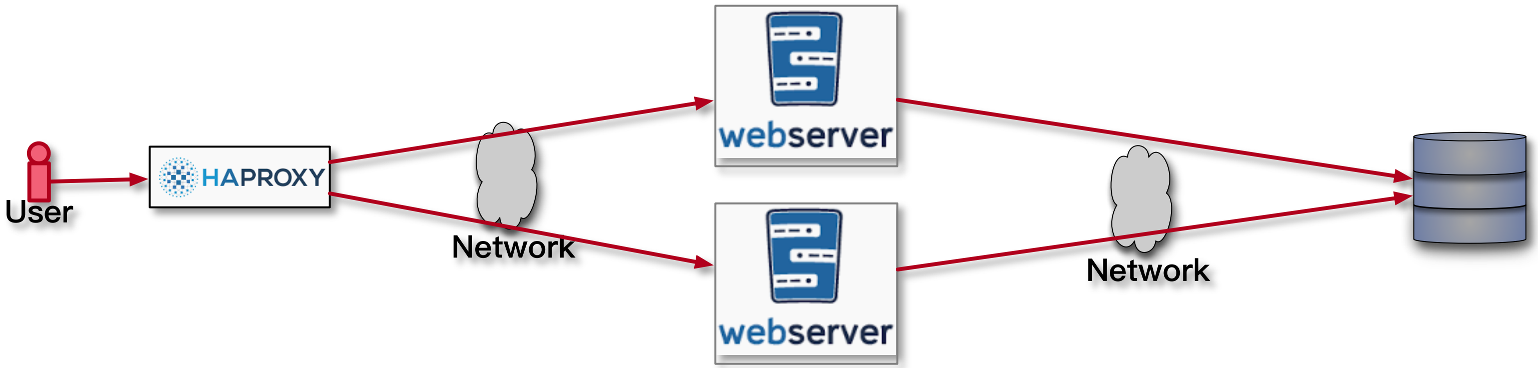
Source: [Google, Datacenter Computers modern challenges in CPU design, 2015]

- SocialNetwork service.
- 31 microservices.

- A Single Transaction Across ~40 Racks of ~60 Servers Each.
- Arc: client-server RPC.

Problem

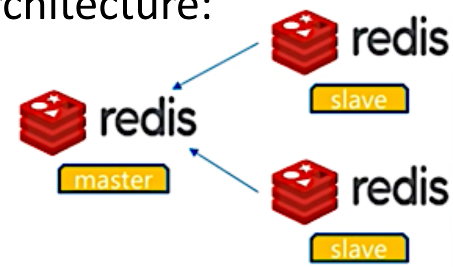
- How can we feedback-control when a request is served by multiple components collaboratively?
  - Latency: $\boxed{L_{overall}} = \boxed{L_{cpn1}} + \boxed{L_{cpn2}} + \cdots$
  - Tail Latency: $\boxed{TL_{overall}} = f\left(\boxed{TL_{cpn1}} + \boxed{TL_{cpn2}} + \cdots\right)$



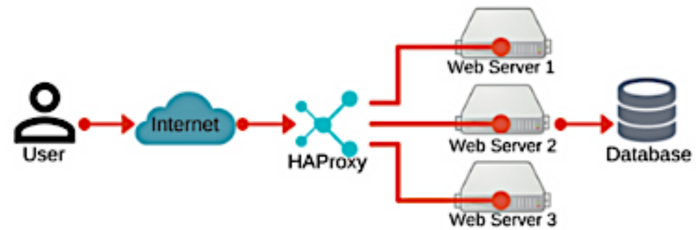
- Given an overall TL, how to derive a sub-TL for each component?
- ***OR:*** How the component-control affect the overall-TL?

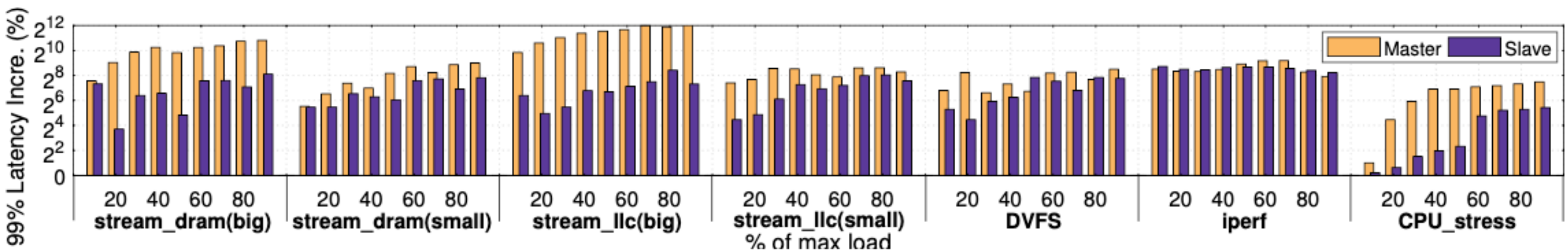
Inconsistent Interference Tolerance

Redis architecture:

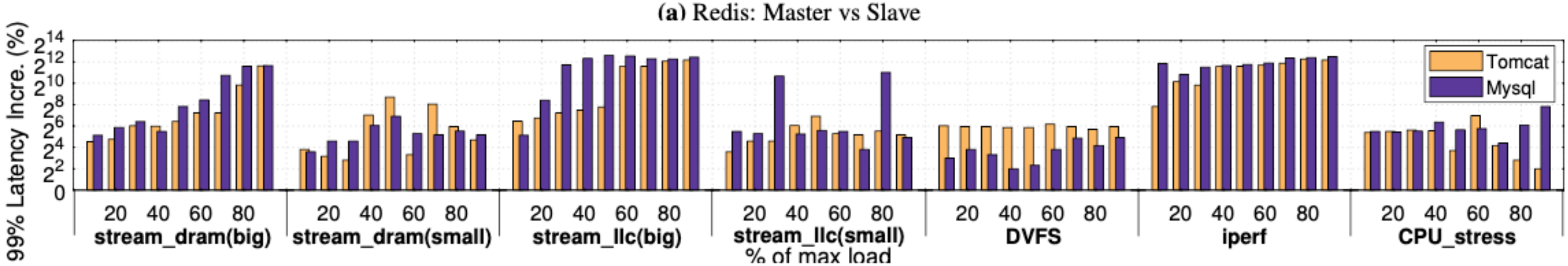


E-commerce architecture:





(a) Redis: Master vs Slave



(b) E-commerce website: Tomcat vs MySQL

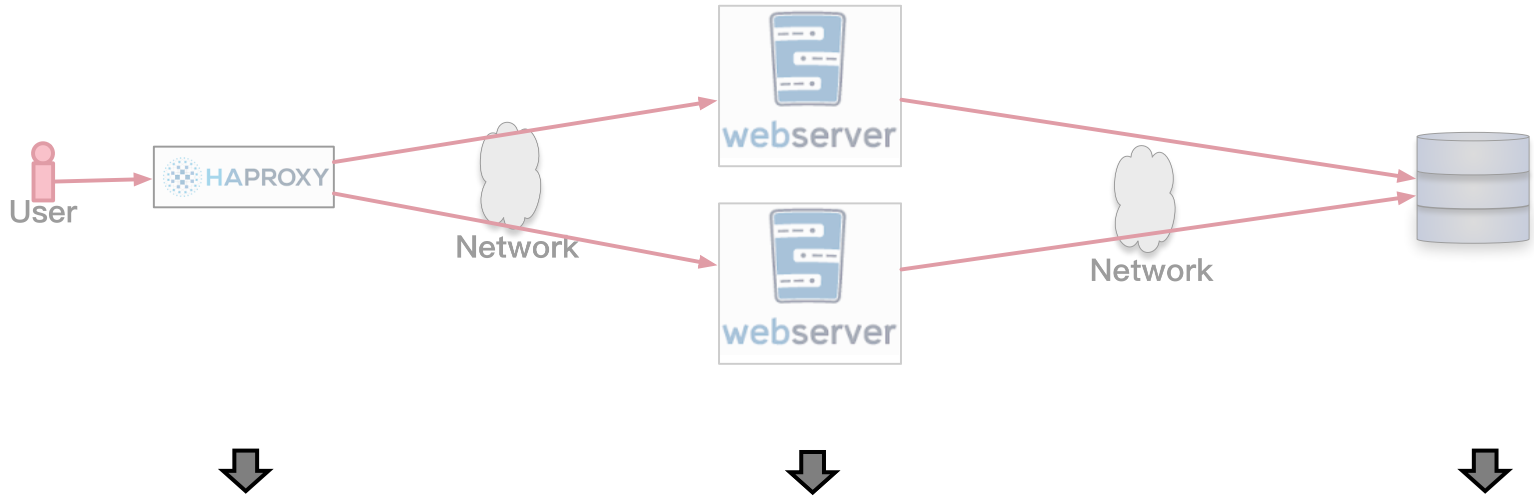
- Components perform significant difference (~435%) under the same source of interference.

Rhythm Design

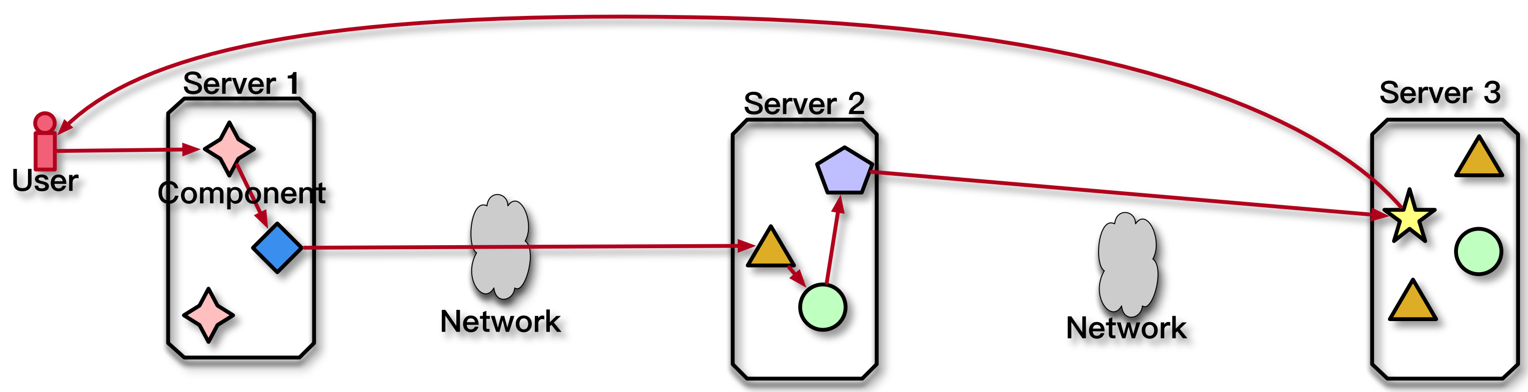
- Rhythm Insight:
  - ***Components with <u>smaller</u> <u>contributions</u> to the tail latency can be co-located with BE jobs <u>aggressively</u>.***

- Challenges:
  - ***How to quantify the contributions of a component?***
  - ***How to control the BE deployment aggressively?***
    - ***When to colocate?***
    - ***How many BEs can we co-locate with the LC?***

■Inconsistent interference tolerance ability;



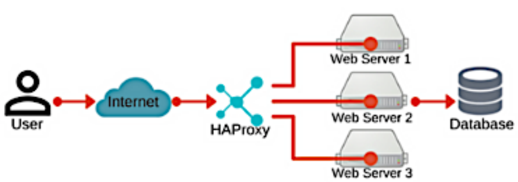
□Tracking user request:

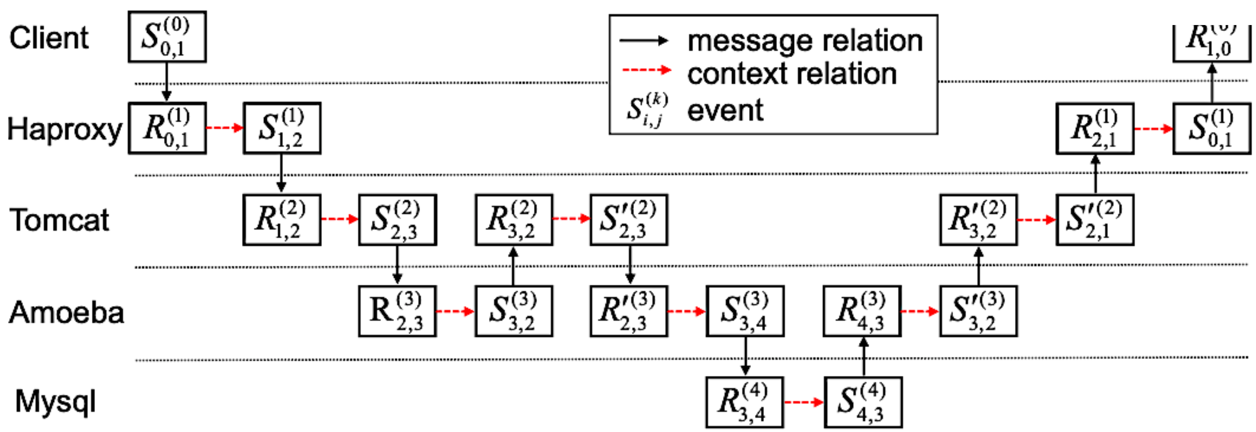


Request tracer

Causal path graph

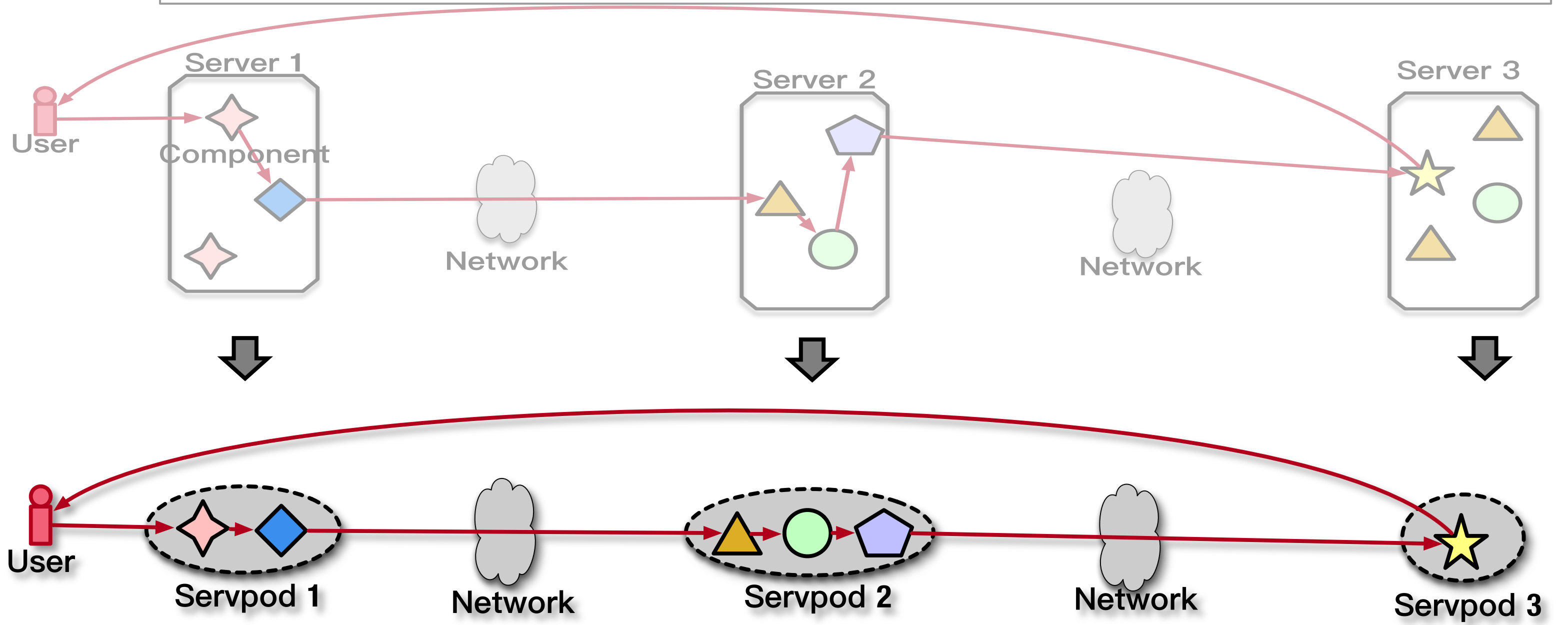
- Send/Receive events: ACCEPT, RECV, SEND, CLOSE
- Event: <*type*, *timestamp*, *context identifier*, *message identifier*>
- Context: <*hostIP*, *programName*, *processID*, *threadID*>
- Message: <*senderIP*, *senderPort*, *receiverIP*, *receivePort*, *messageSize*>





■ Inconsistent interference tolerance ability;

■ Tracking user request;

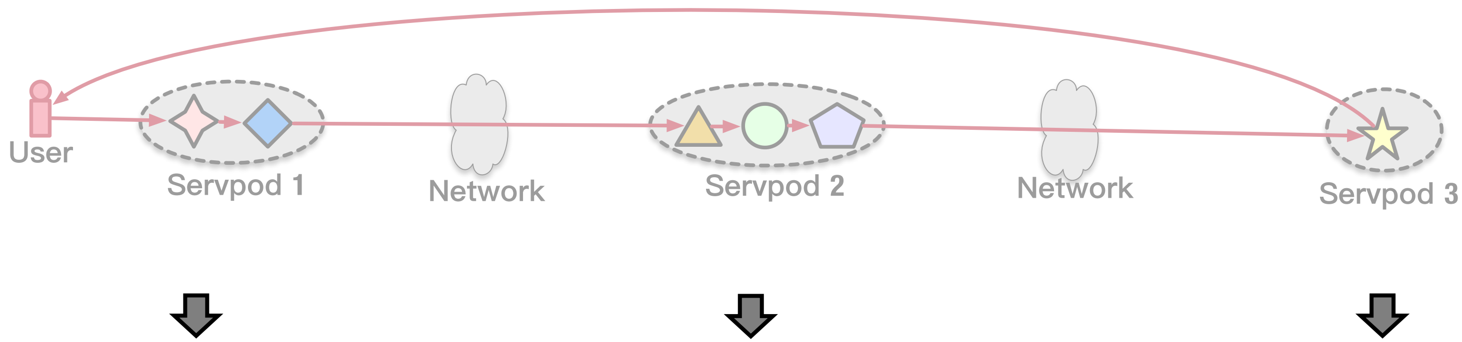


❑ Servpod abstraction:

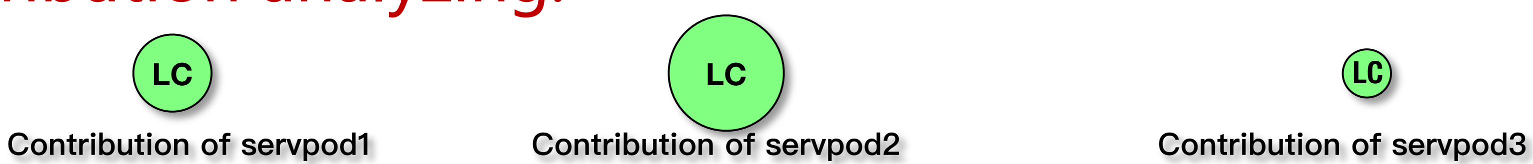
- A collection of service components from one LC service that are deployed together on the same physical machine.
- For deriving the sojourn time of each request in each server.

Rhythm

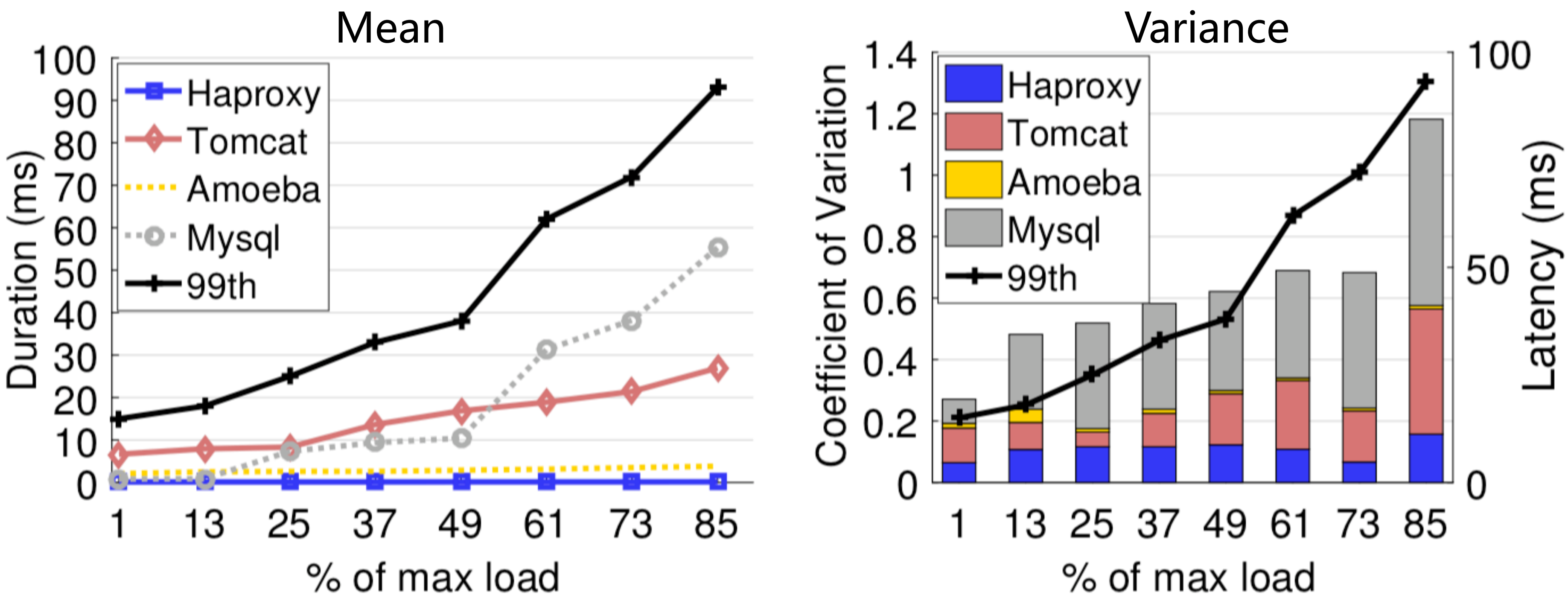
- Inconsistent interference tolerance ability;
- Tracking user request;
- Servpod abstraction;



- Contribution analyzing:



Contribution Analyzer

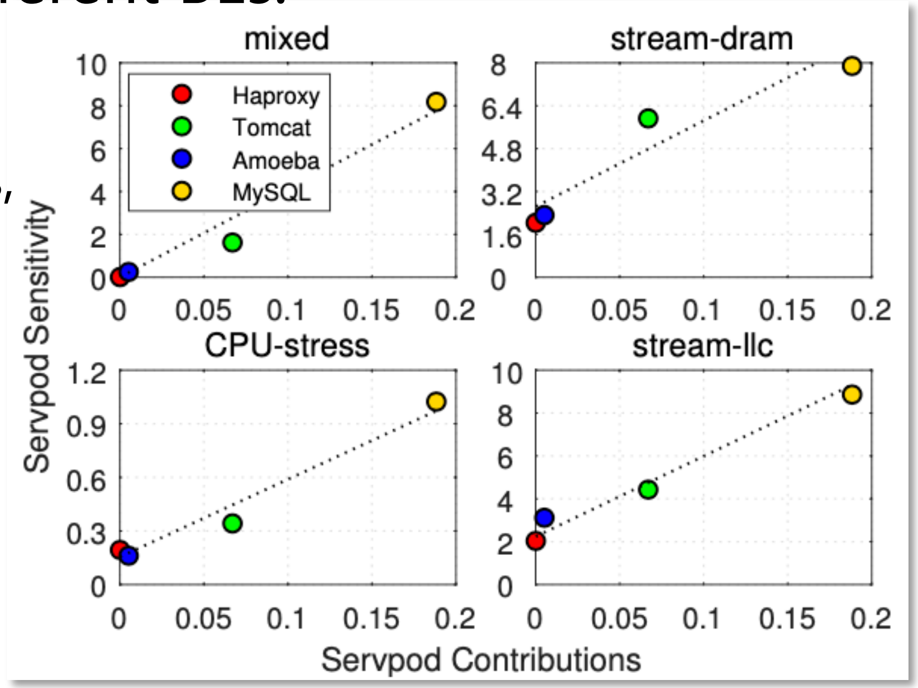


- Servpods with higher average sojourn time contribute more to TL.
- Servpods with higher sojourn time variance contribute more to TL.
- Servpods that highly correlated with the tail latency contribute more to tail latency.

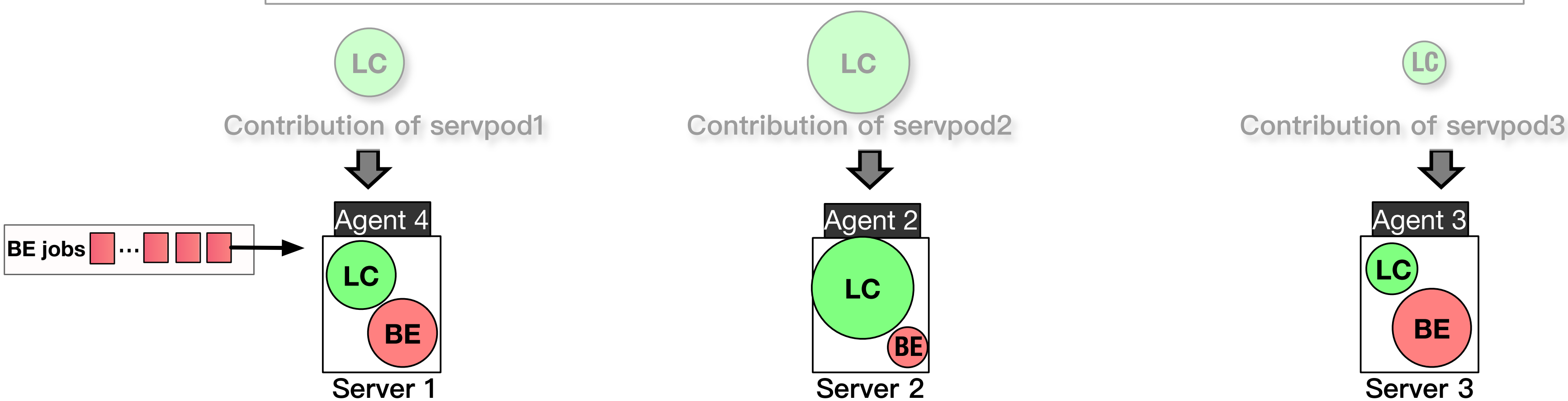
$$C_i = f(\rho_{T_i,T_{99th}}, P_i, V_i) = \rho_{T_i,T_{99th}} P_i V_i$$

Contribution Analyzer

- Is this definition effective?
  - Sensitivity vs contributions
  - The increase in the 99^{th}-tile latency when a single Servpod is interfered by different BEs:
    - Mixed BEs of wordcount, imageClassify, lstm, CPU-stress, stream-dram and stream-llc.
    - DRAM intensive: Stream-dram
    - CPU intensive: CPU-stress
    - LLC intensive: Stream-llc.



- Inconsistent interference tolerance ability;
- Tracking user request;
- Servpod abstraction;
- Contribution analyzing;

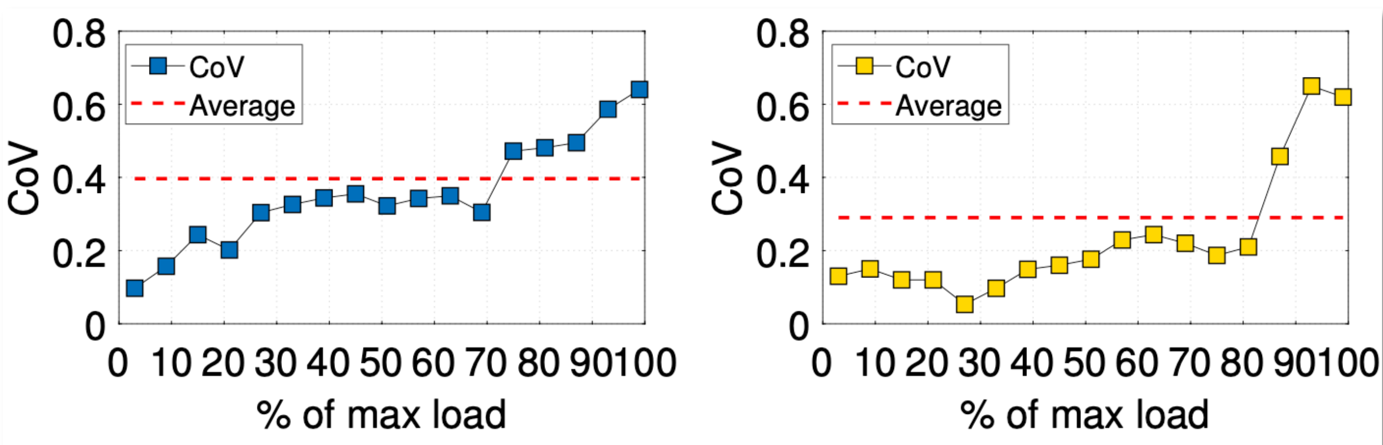


Controller:

- Loadlimit: allowing colocation when *load<loadlimit*;
  - The "Knee point" of performance-load curve.
- Slacklimit: the lower bound of *slack* for allowing the growth of BEs.
  - Slack = *SLA – currentTL;*
  - *Small contribution → larger slacklimit;*

Controller

- ***When can we co-locate workloads?***
  - Loadlimit.
- *Loadlimit* per servpod:
  - The upper bound of the request load for allowing the colocation with BE jobs;
  - *knee point*: 76% of max for MySQL; 87% of max for Tomcat.

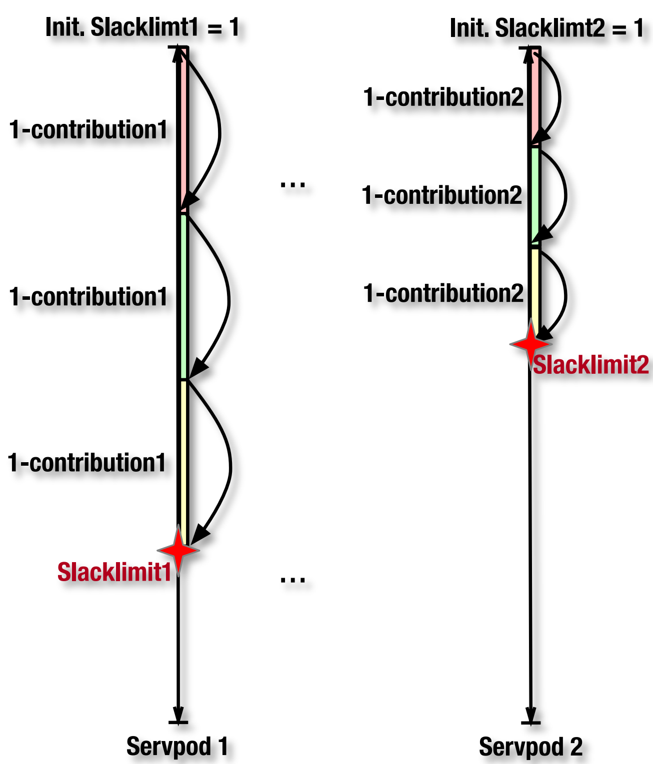


(a) MySQL **(b)** Tomcat

Controller

☐ *How many BEs can we co-locate?*

- Slacklimit: the lower bound of *slack* for allowing the growth of BE jobs.



Co-locating decisions:

Slack = SLA – currentTL;

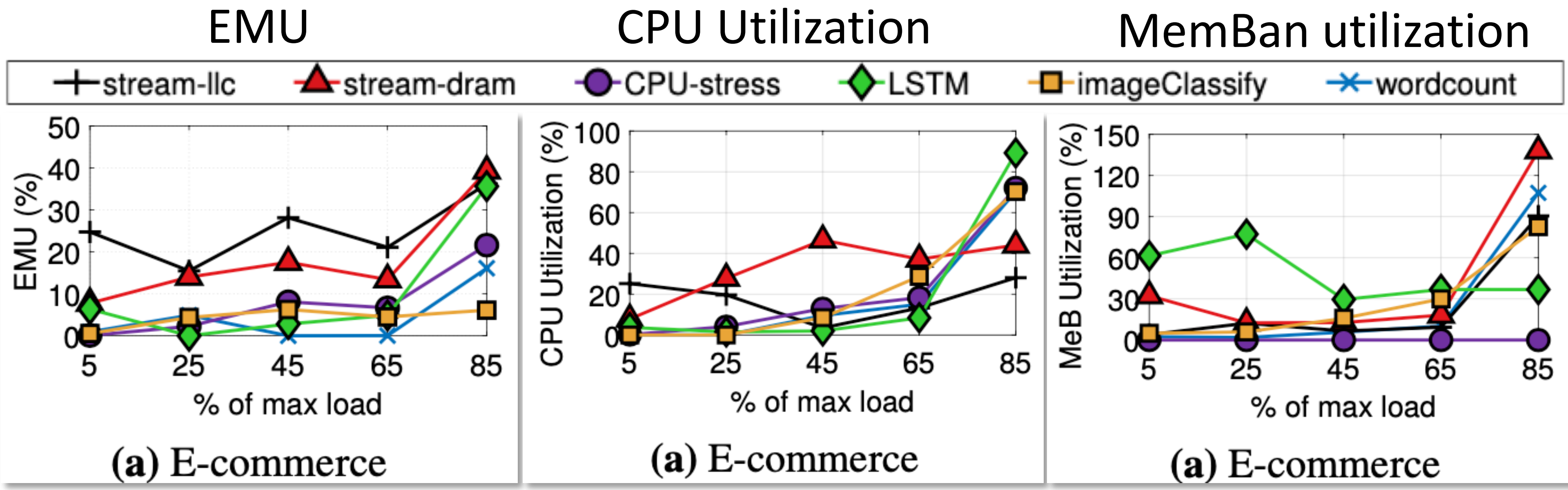
if $slack < 0$ **then**
  StopBE();
else if $0 < slack < slackLimit/2$ **then**
  CutBE();
else if $slackLimit/2 < slack < slackLimit$ **then**
  DisallowBEGrowth();
else
  AllowBEGrowth();

- contribution 1 < contribution 2
- slacklimit1 < slacklimit 2

Experimental Evaluation

- Benchmarks:
  - LC services：
    - Apache Solr：Solr engine+Zookeeper
    - Elasticsearch：Index+Kibana
    - Elgg：Webserver+Memcached+Mysql
    - Redis：Master + Slave
    - E-commerce: Haproxy+Tomcat+Amoeba+Mysql
  - BE Tasks：
    - CPU-Stress; Stream-LLC; Stream-DRAM
    - Iperf：Network
    - LSTM：Mixed
    - Wordcount
    - ImageClassify： deep learning
- Testbed
  - 16 Sockets, 64 GB of DRAM per socket. Each socket shares 20 MB of L3 cache.
  - Intel Xeon E7-4820 v4 @ 2.0 GHz: 32 KB L1-cache and 256 KB L2-cache per core.
  - The operating system is Ubuntu 14.04 with kernel version 4.4.0-31.

Overall Analysis



- *Overall analysis* (compared to Heracles [ISCA,2015])
  - Improve EMU (=LC throughput + BE throughput) by 11.6%~24.6%;
  - Improve CPU utilization by 19.1%~35.3%;
  - Improve memory bandwidth utilization by 16.8%~33.4%.

Timeline Analysis

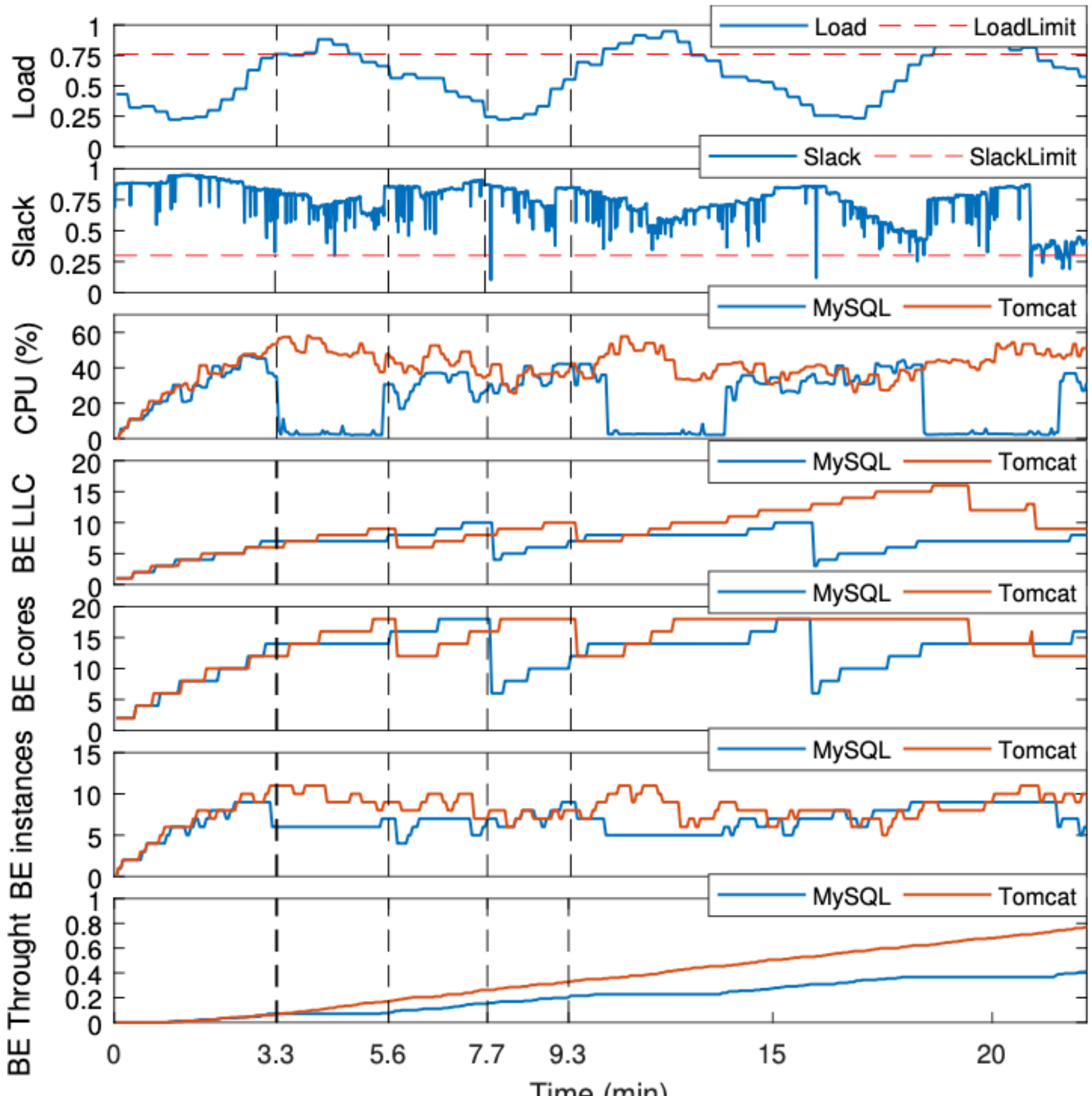


Figure 17. The timeline of Rhythm's running process.

- Timeline：
  - Time 3.3： suspendBE()；
  - Time 5.6： allowBEGrowth()；
  - Time 7.7： cutBE();
  - Time 9.3: suspendBE().

Conclusion

- Rhythm, a deployment controller that maximizes the resource utilization while guaranteeing LC service`s tail latency requirement.
  - Request tracer
  - Contribution analyzer
  - Controller
- Experiments demonstrate the improvement on system throughput and resource utilization.

Thank you!
Questions?