What Autoscaling should react to?

Energy efficiency & computing

Submitted by,
Alpit Gupta (110451714)
Aditya Prakash (110350983)
Kumar Sasmit (110308698)
Vishal Sahu (110310091)

Supervised by,
Anshul Gandhi

Problem statement

- Determine suitable metric for horizontal autoscaling policy which results in:
  - Maximum energy/power saving
  - Minimum wear-and-tear of VMs
  - Minimum turnaround time
  - Minimal SLA violations

Motivation

- Modern web servers are configured to **handle worst case load requirements**.
- Most of the backend servers remain **underutilized** for most of the time.
- There is need of informed auto-scaling policy which can address above problems.

Prior Work

- Load Balancer Behavior Identifier (LoBBI) for Dynamic Threshold Based Auto-scaling in Cloud
  - **Rule engine** based framework to add/remove VMs[link]
  - Reactive and proactive scaling by **analyzing load behavior**
- Cloud Auto-scaling with Deadline and Budget Constraints
  - Reduces cost by choosing **appropriate instance types** (High CPU, High IO, Mixed)[link].
  - Scaling based on **performance desire** and workload information.
- Dynamic Selection of VMs for Application Servers in Cloud Environments[link]
  - This paper dynamically selects type of VM as per requirement.
  - **Observes change in workload** patterns and configuration.

Prior Work contd...

- BATS: Budget-Constrained Autoscaling for Cloud Performance Optimization[link]
  - Considers budgeting period(monthly/ yearly) in making scaling decisions.
  - Autoscaling based on past and instantaneous load data.
- Adaptive, Model-driven Autoscaling for Cloud Applications
  - Used Kalman filtering + **queueing theory** for modeling to proactively determine the right scaling actions[link].
- Evaluating Auto-scaling Strategies for Cloud Computing Environments[link]
  - Evaluation using **log traces from** real workload servers such as **Google**.
  - Studies auto scaling performance based of unique metric “Autoscaling demand index”.

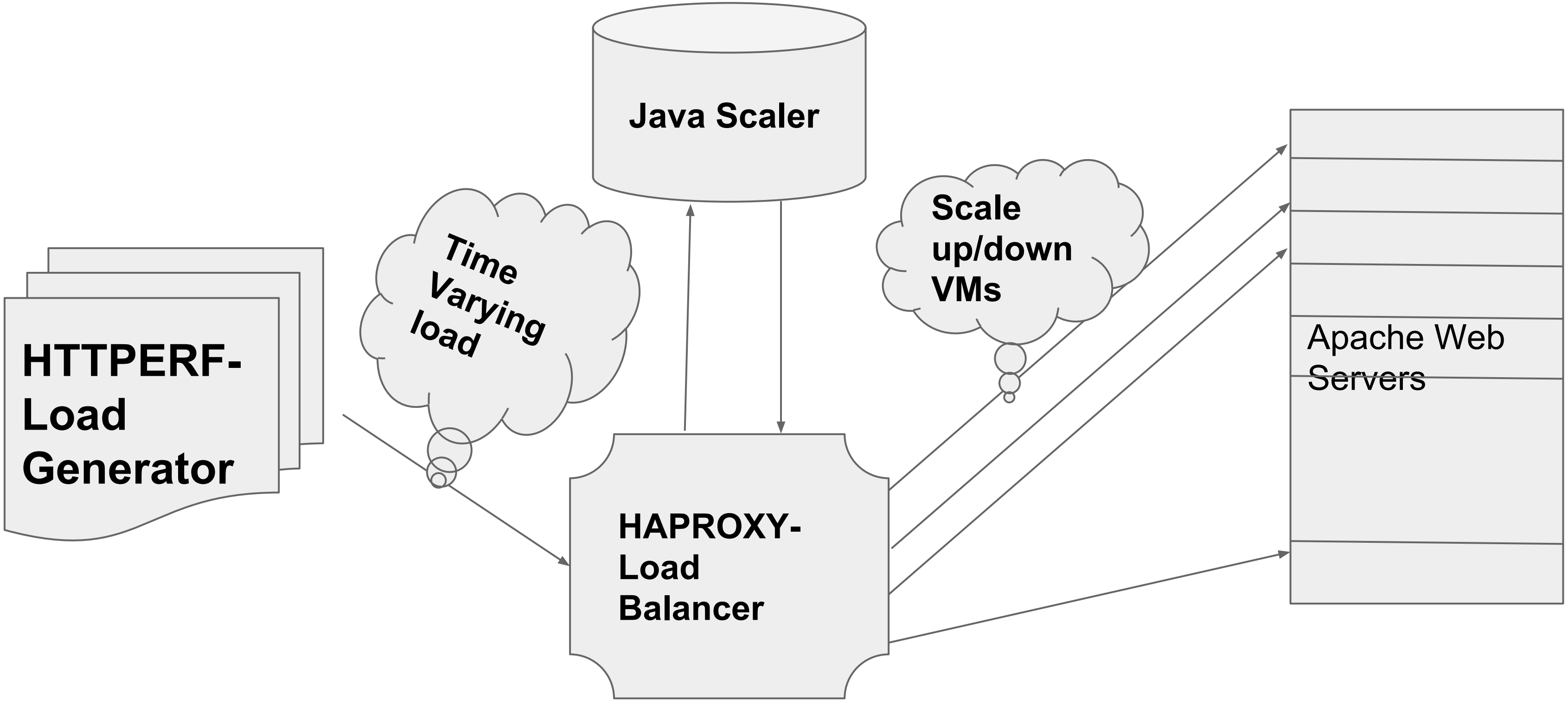
Our early Attempts

- Google cloud auto-scaling techniques
  - In-built Custom metric based autoscaling
  - Complete customization is not available to user.
- Network packet sniffing
  - Sniff IP packets to measure request rate, queue length and response time metric values.
  - Tedious and inaccurate method.
- Apache based Load balancing
  - Observe apache load balancing logs to estimate the task distribution among servers.

Metric under action

- Request Rate
  - Number of sessions per second over last elapsed second.
  - Used HTTPERF as a time varying load generator.
- Queue Length
  - Number of requests queued, waiting for a server.
  - Used apache web server to buffer requests.
- Response Time
  - Average response time(ms) over the last 1024 requests.
  - Haproxy keeps track of response time.

System architecture



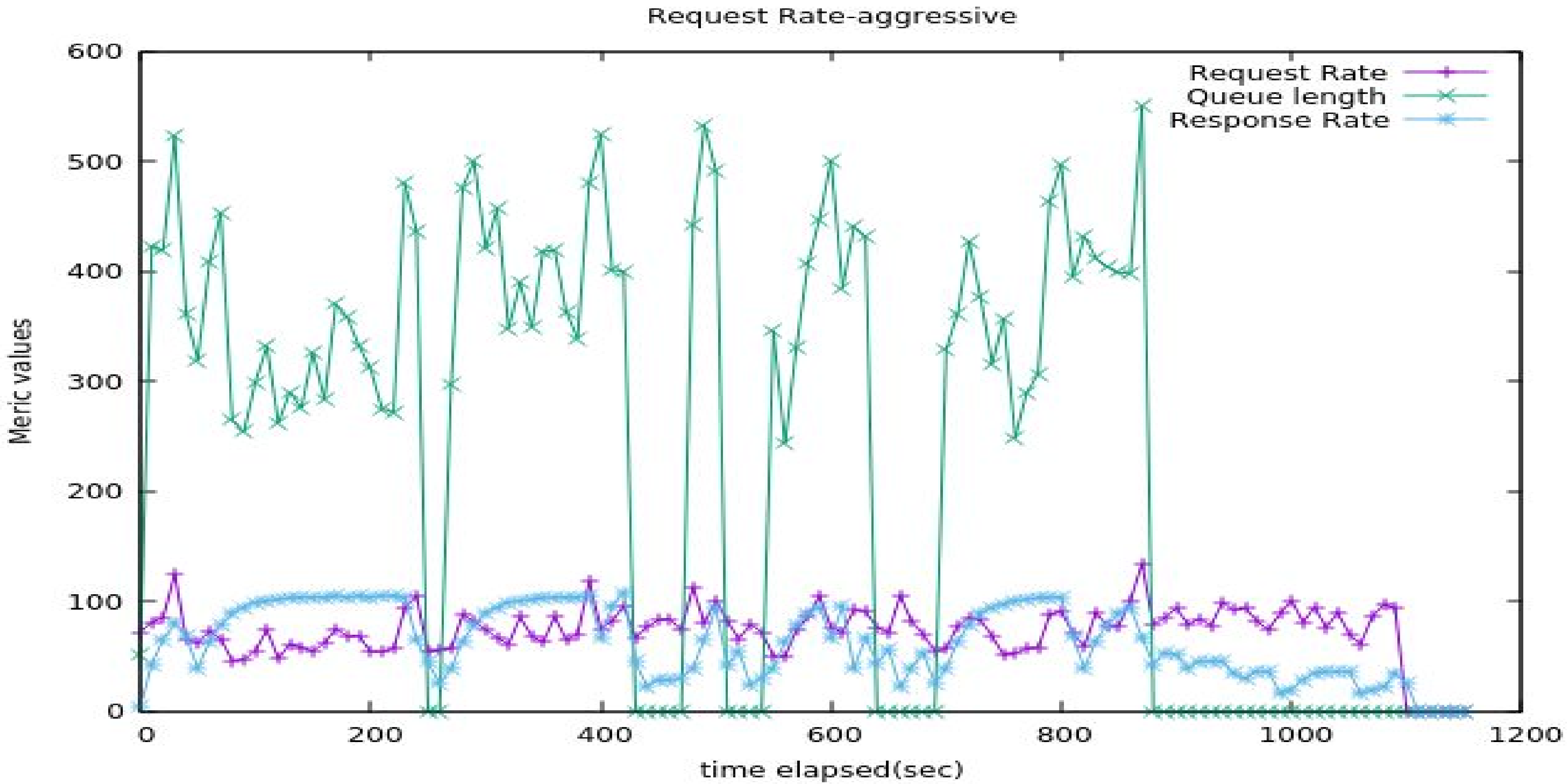
Results and Evaluation

Variety of Modes & Traces

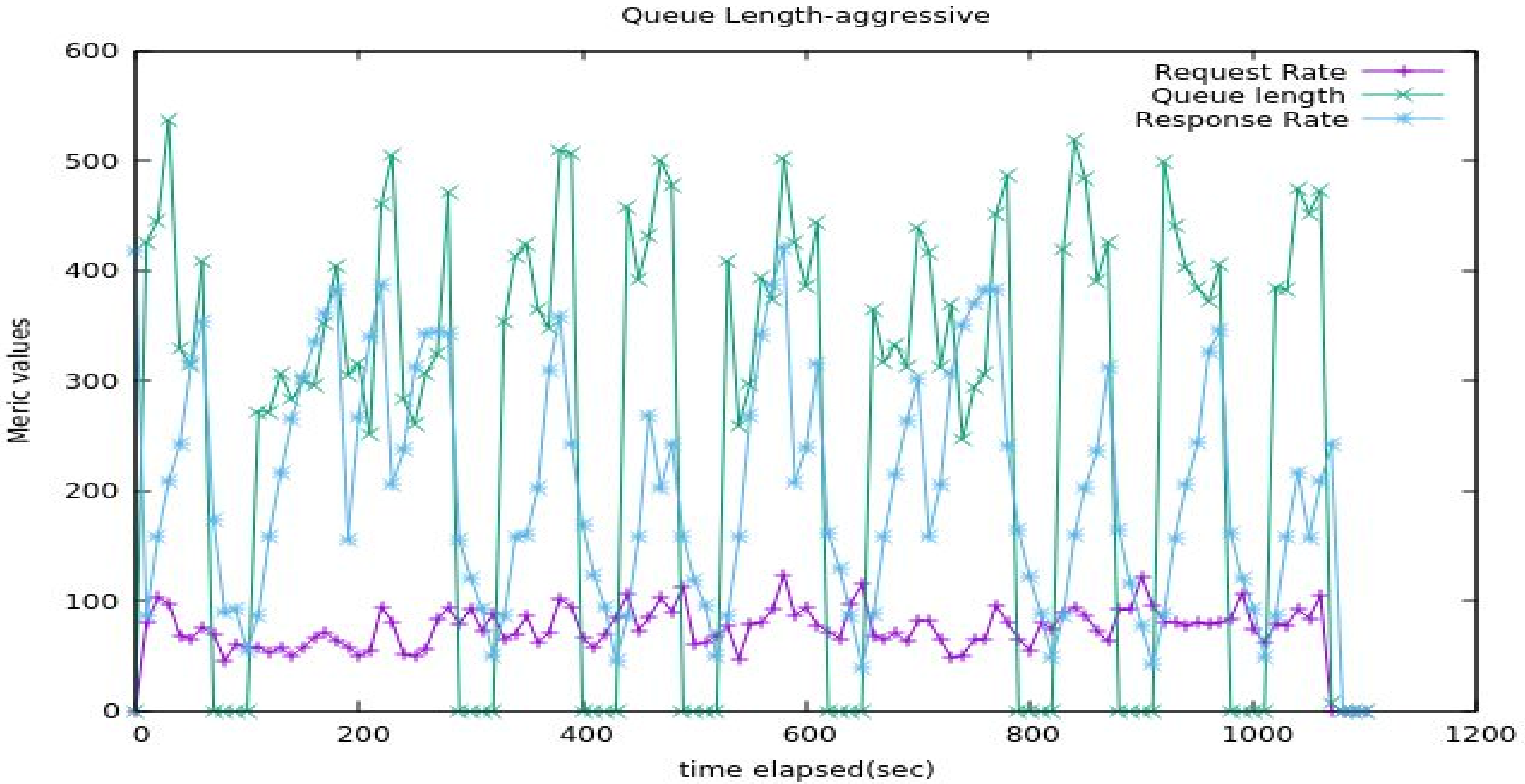
- Workload traces:
  - Static : Smaller variation between min and max request rate.
  - Dynamic: Larger variation between min and max request rate.

- Autoscaling threshold Modes:
  - Conservative: Wider range between upscale and downscale bounds.
  - Aggressive : Narrow range between upscale and downscale bounds.

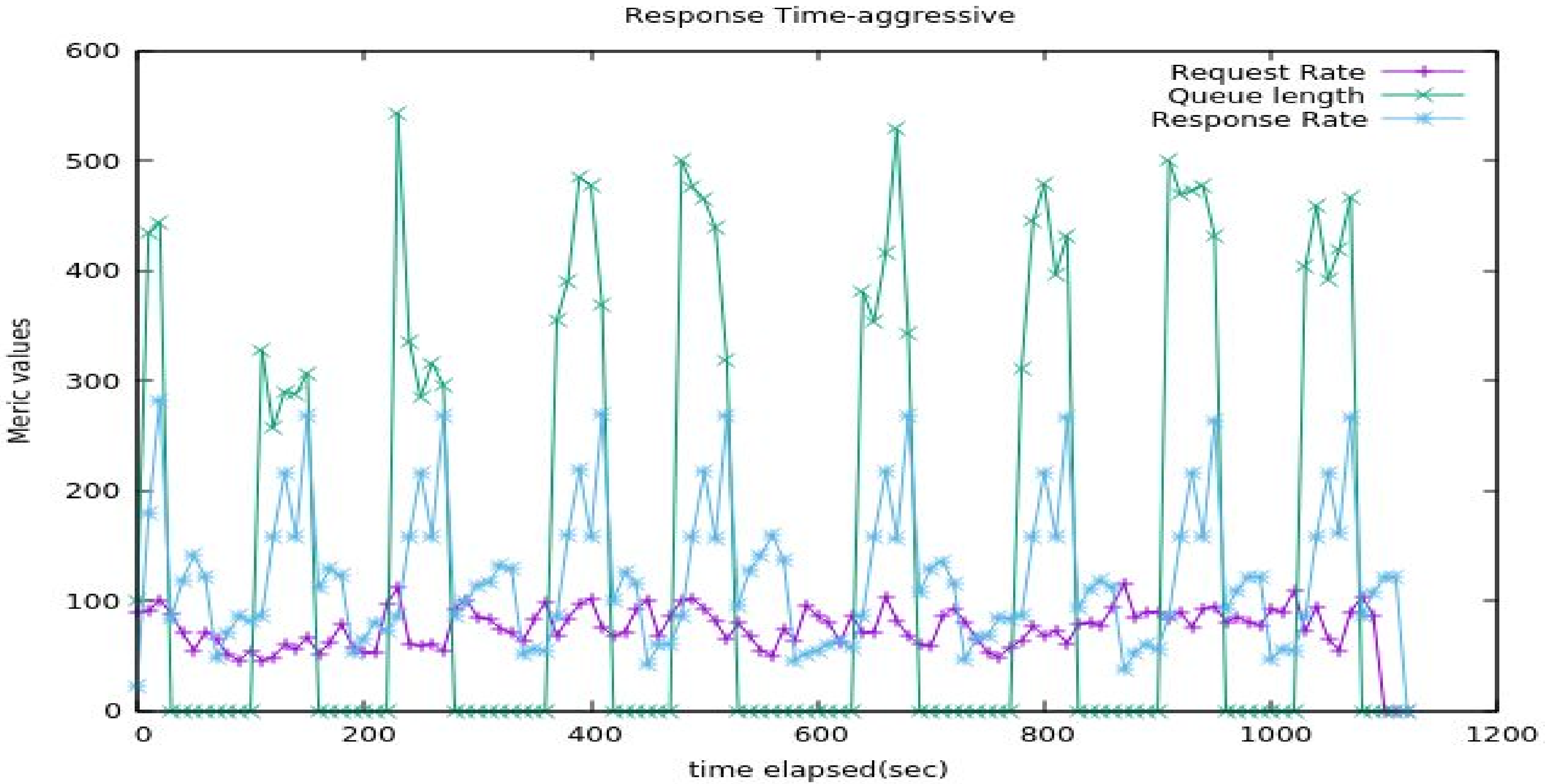
Request Rate, Static trace



Queue Length, static trace

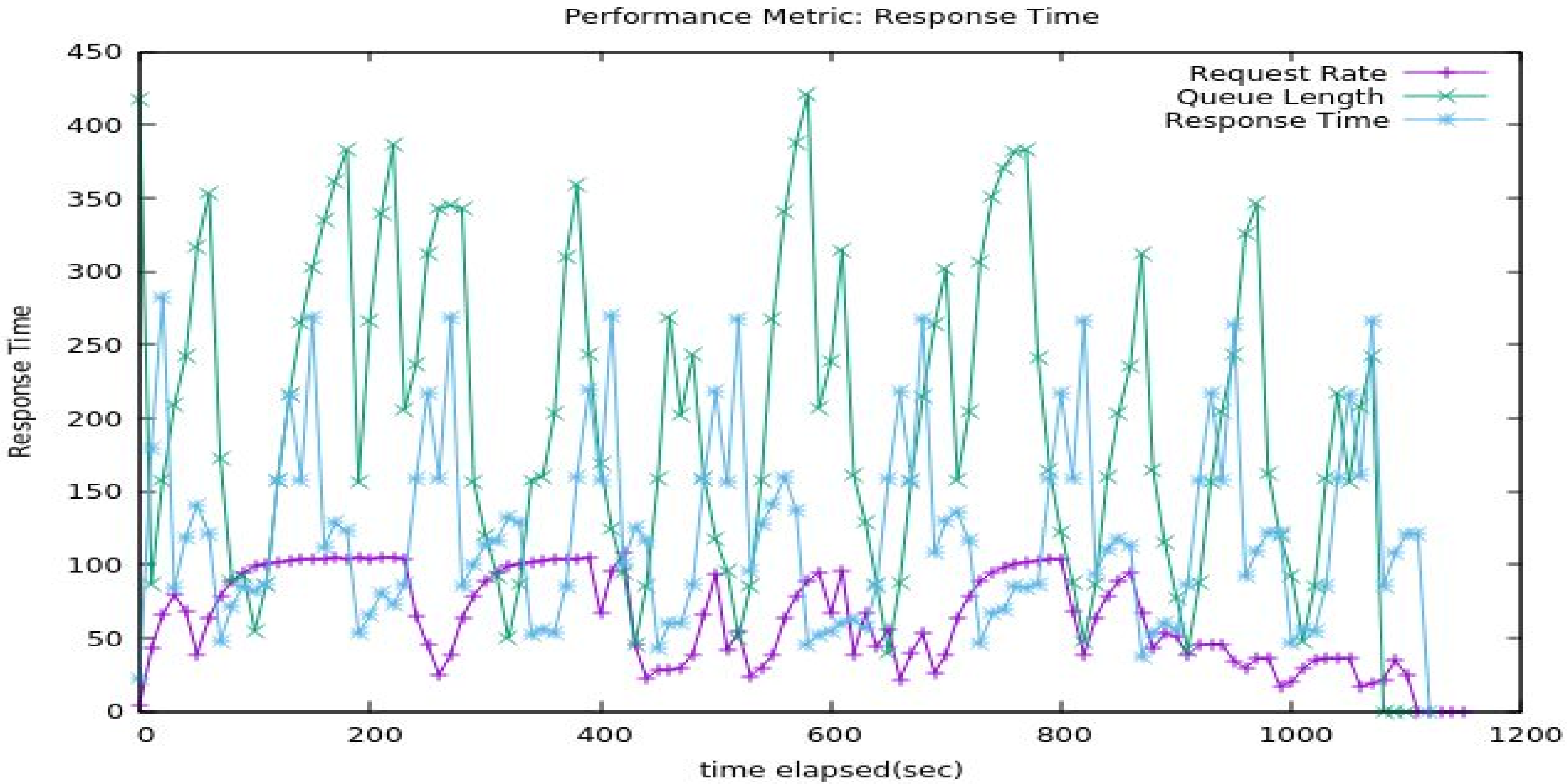


Response Time, static trace



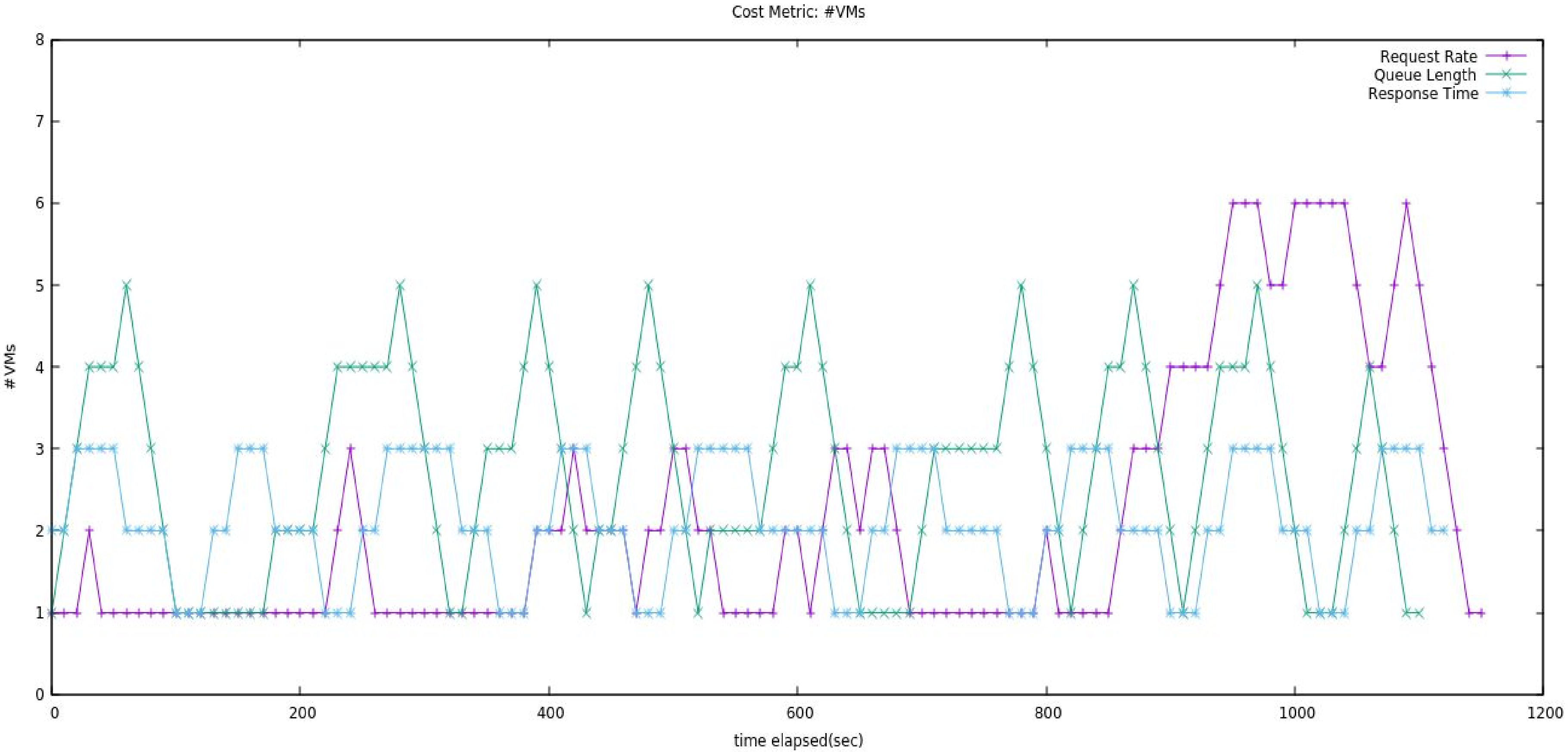
Performance Metric, static trace

Request Rate : Avg. Response time = 63.06034
Queue Length : Avg. Response time = 197.6036
Response Time : Avg. Response time = 122.6018

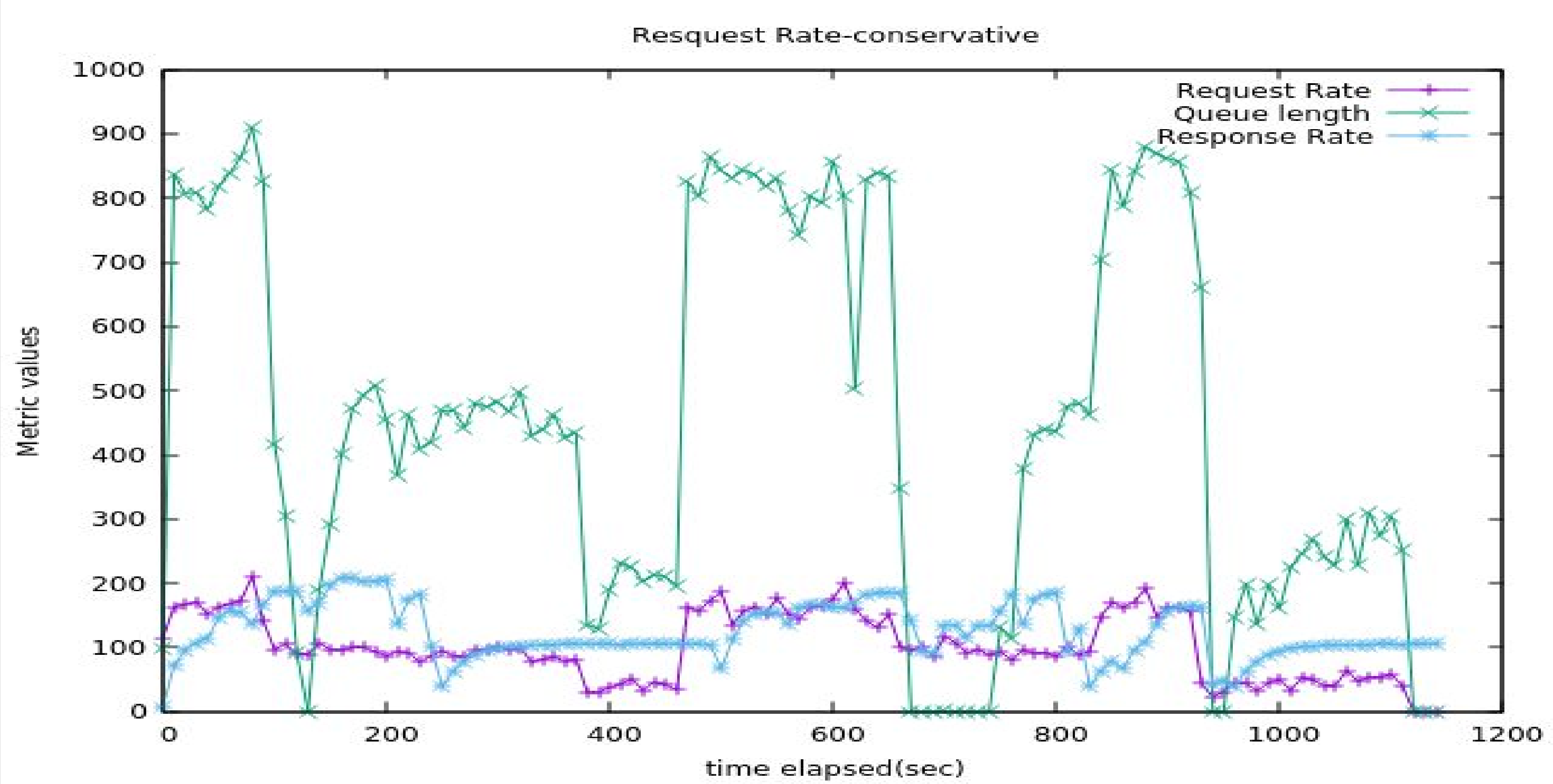


Cost Metric, static trace

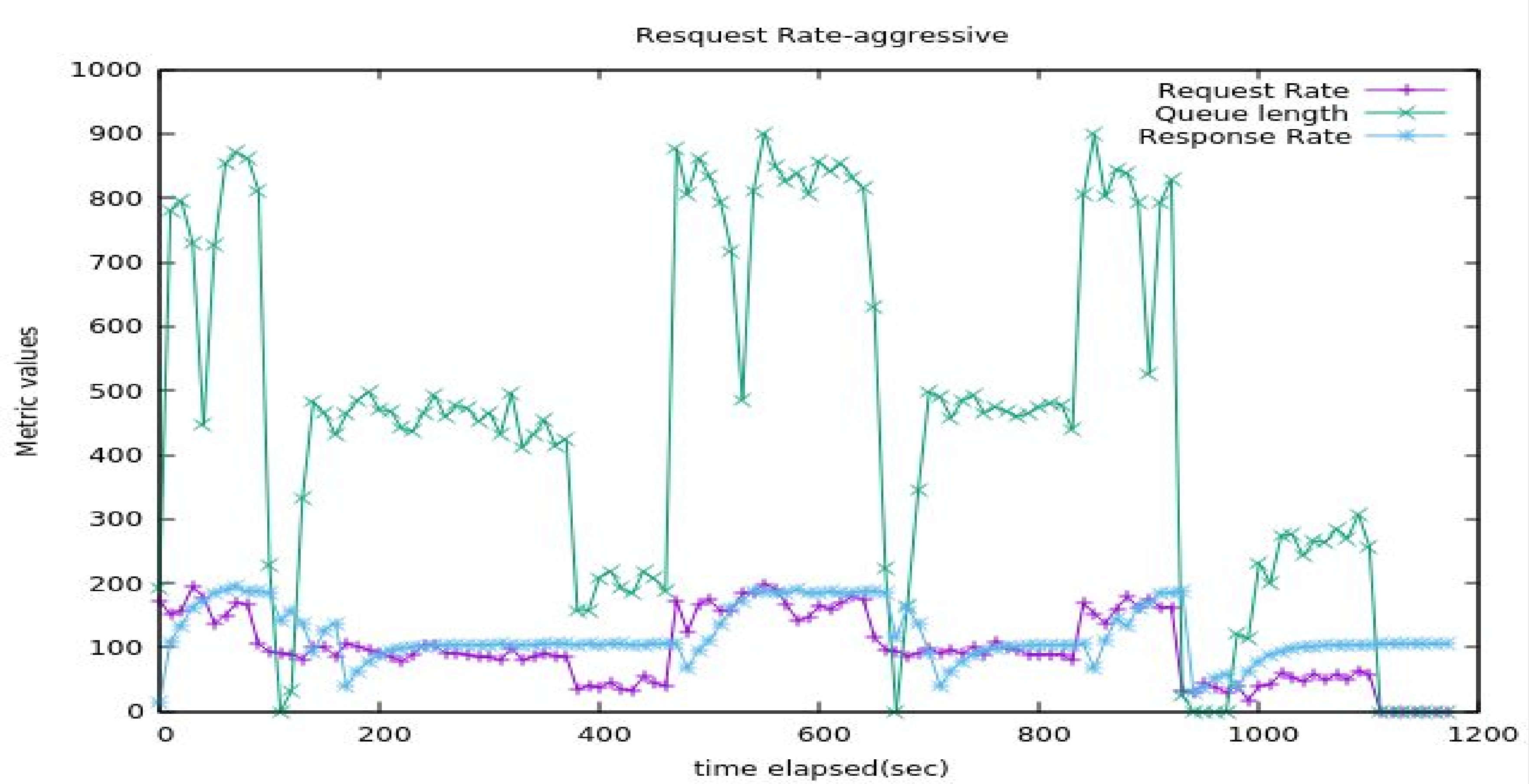
Request Rate : Average #VM = 2.172414
Queue Length : Average #VM = 2.702703
Response Time: Average #VM = 2.115044



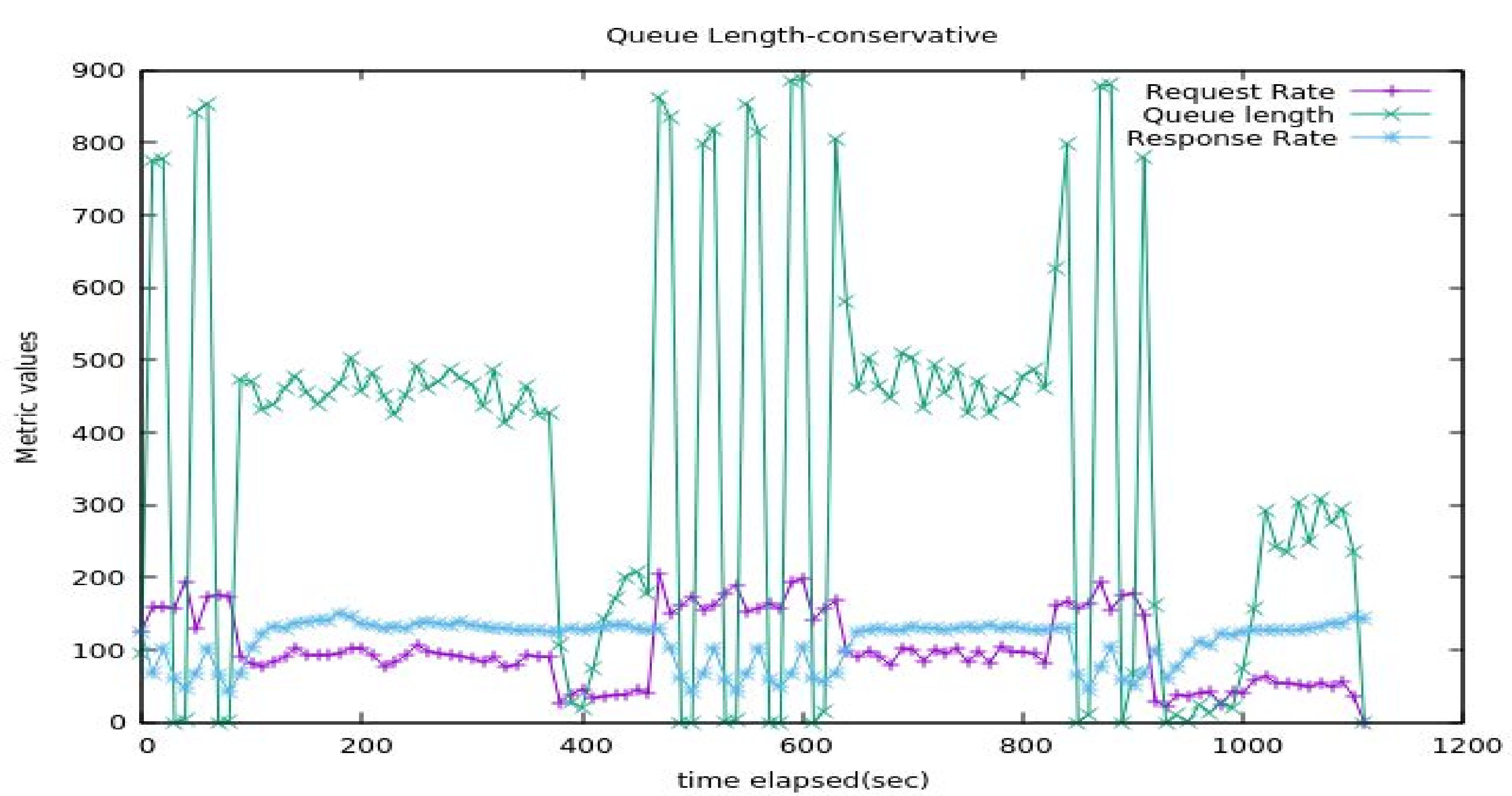
Request rate-Conservative, dynamic trace



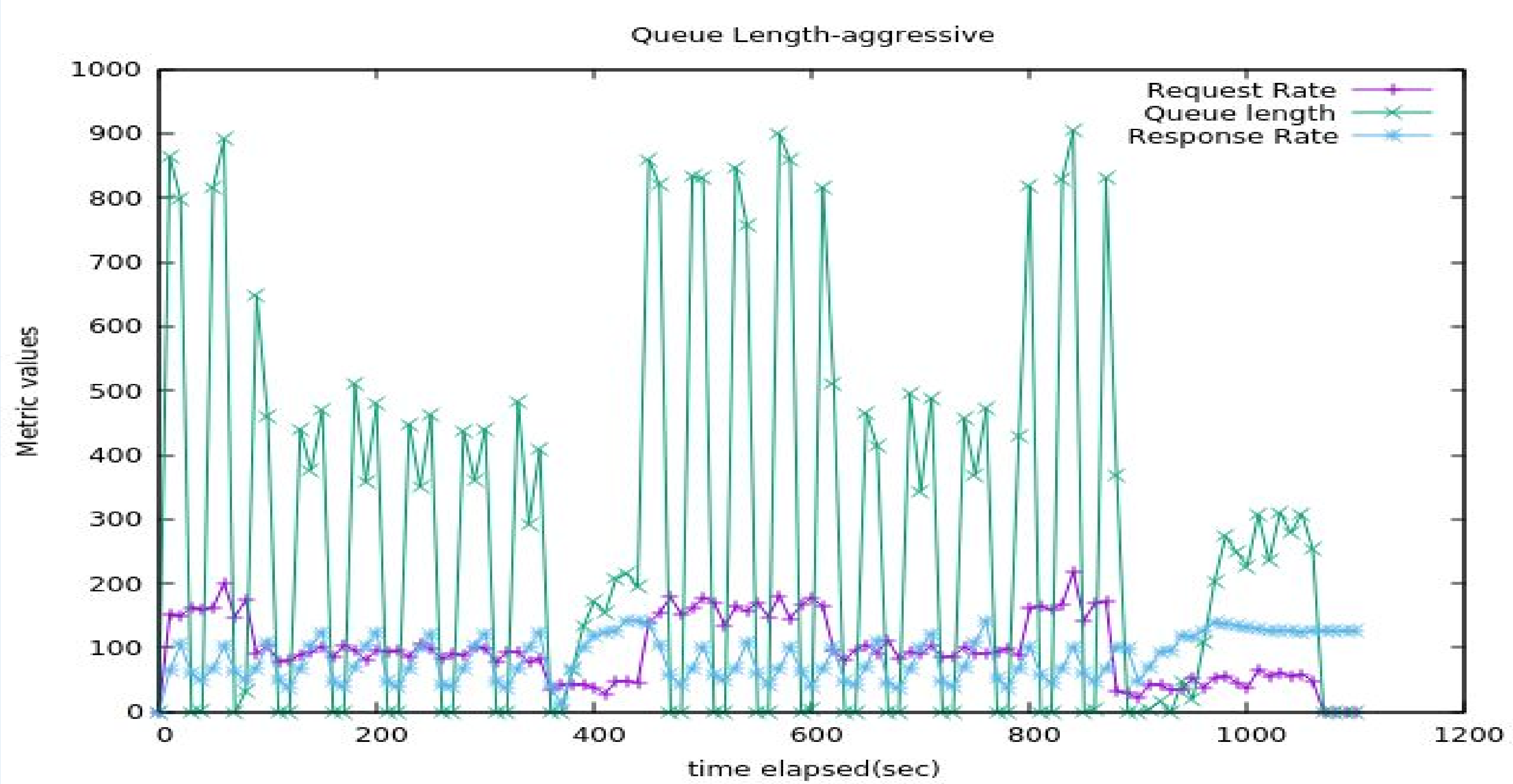
Request rate-aggressive, dynamic trace



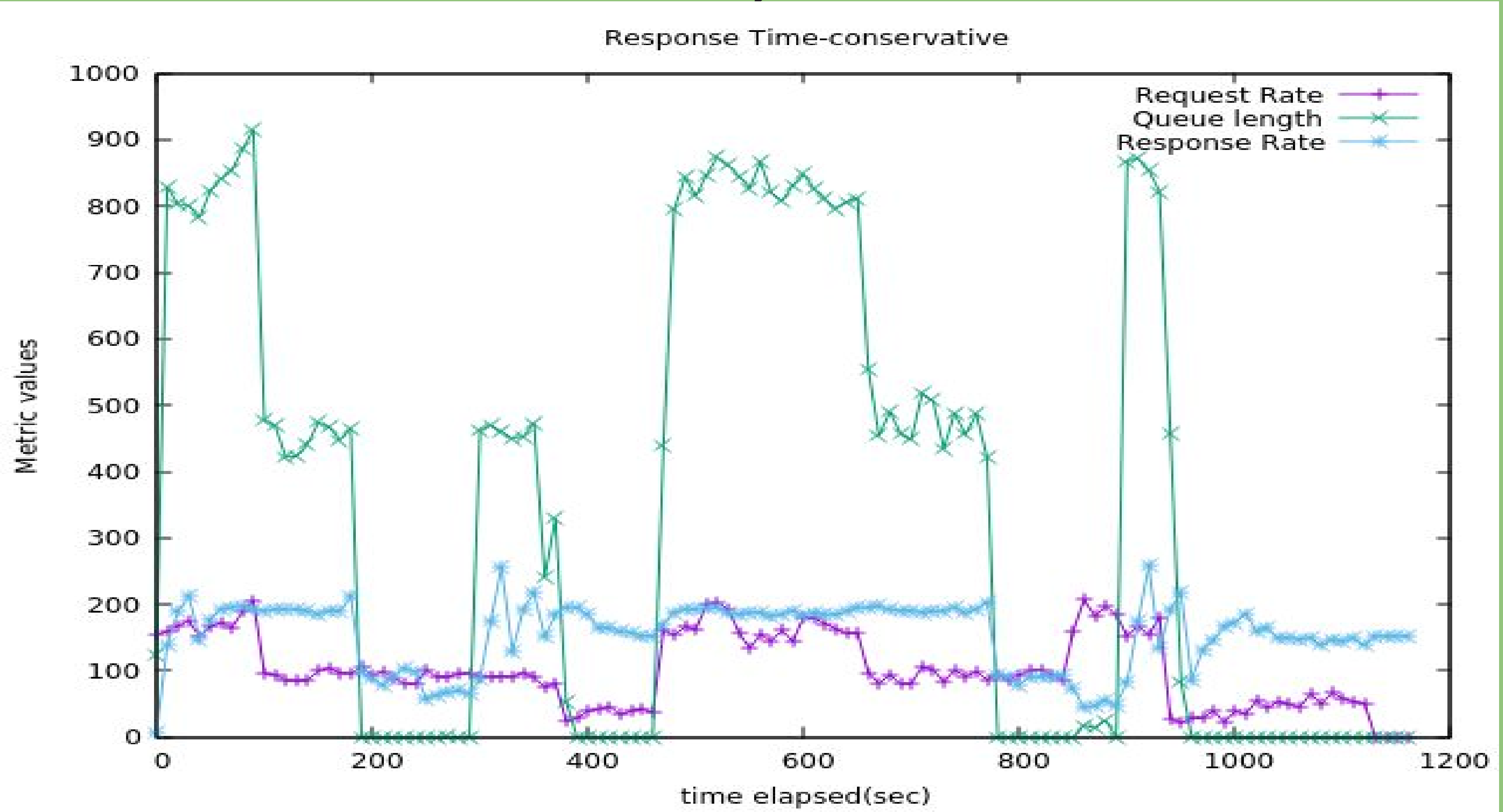
Queue Length-conservative, dynamic trace



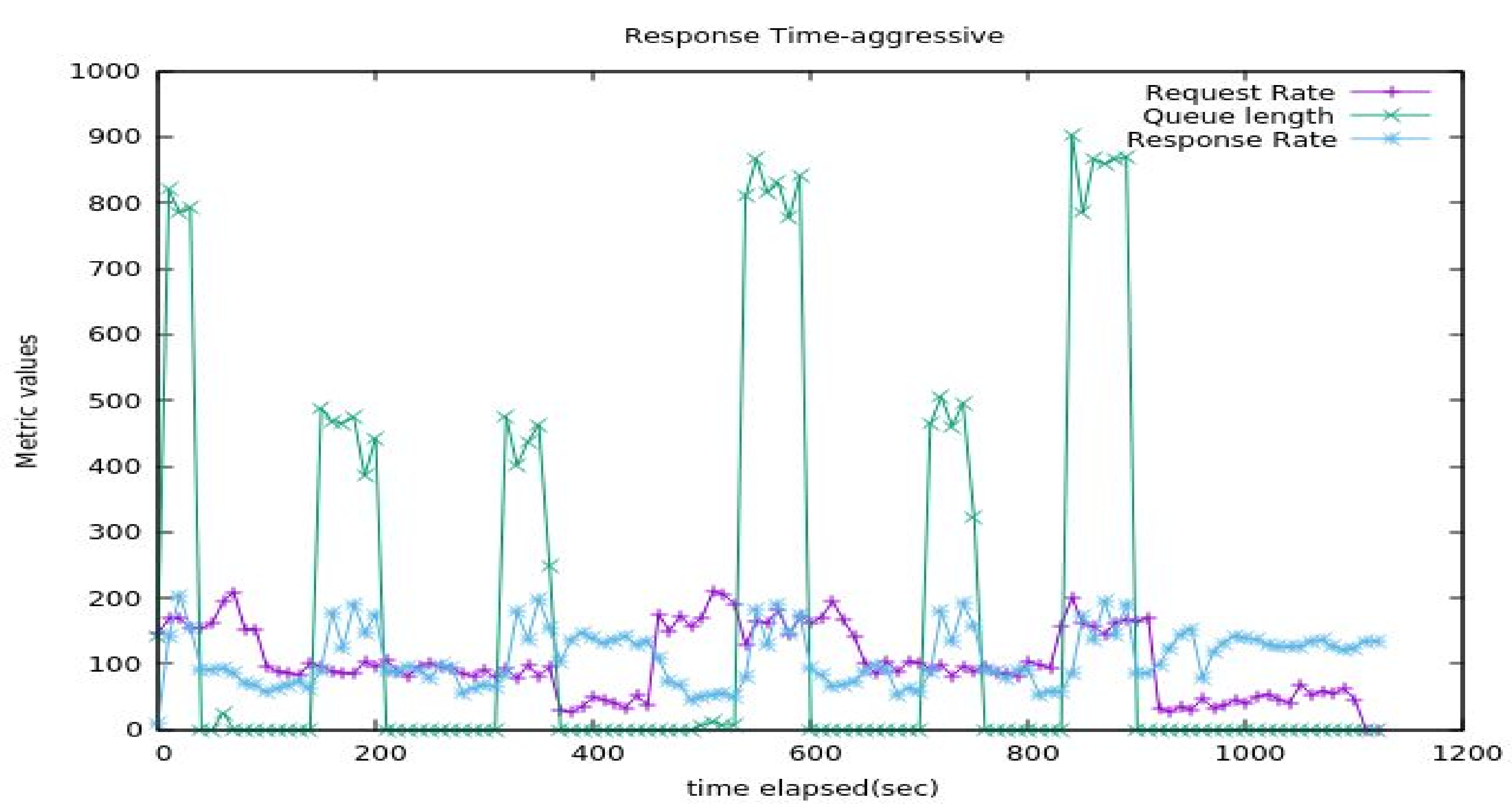
Queue Length-aggressive, dynamic trace



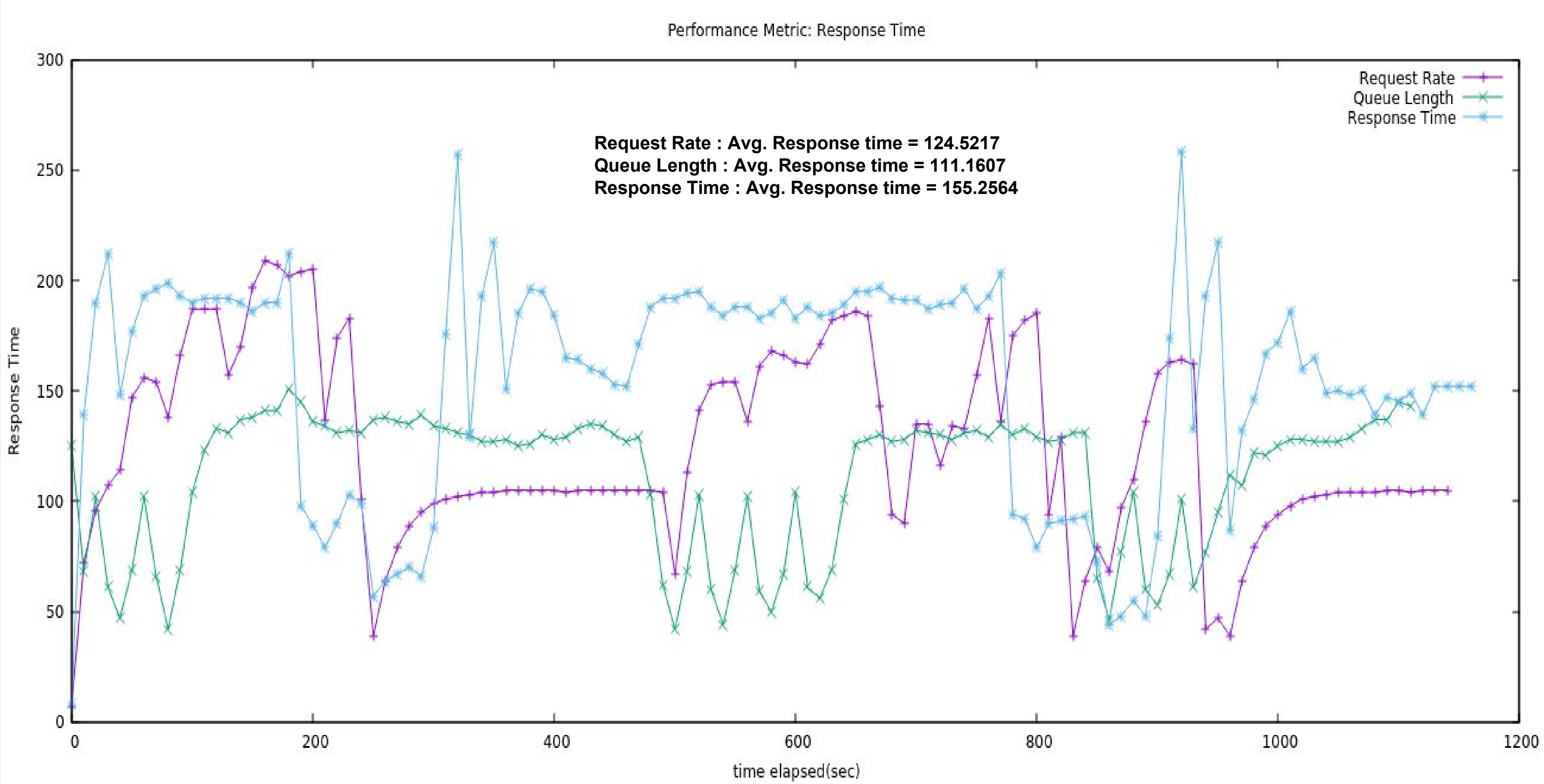
Response Time-conservative, dynamic trace



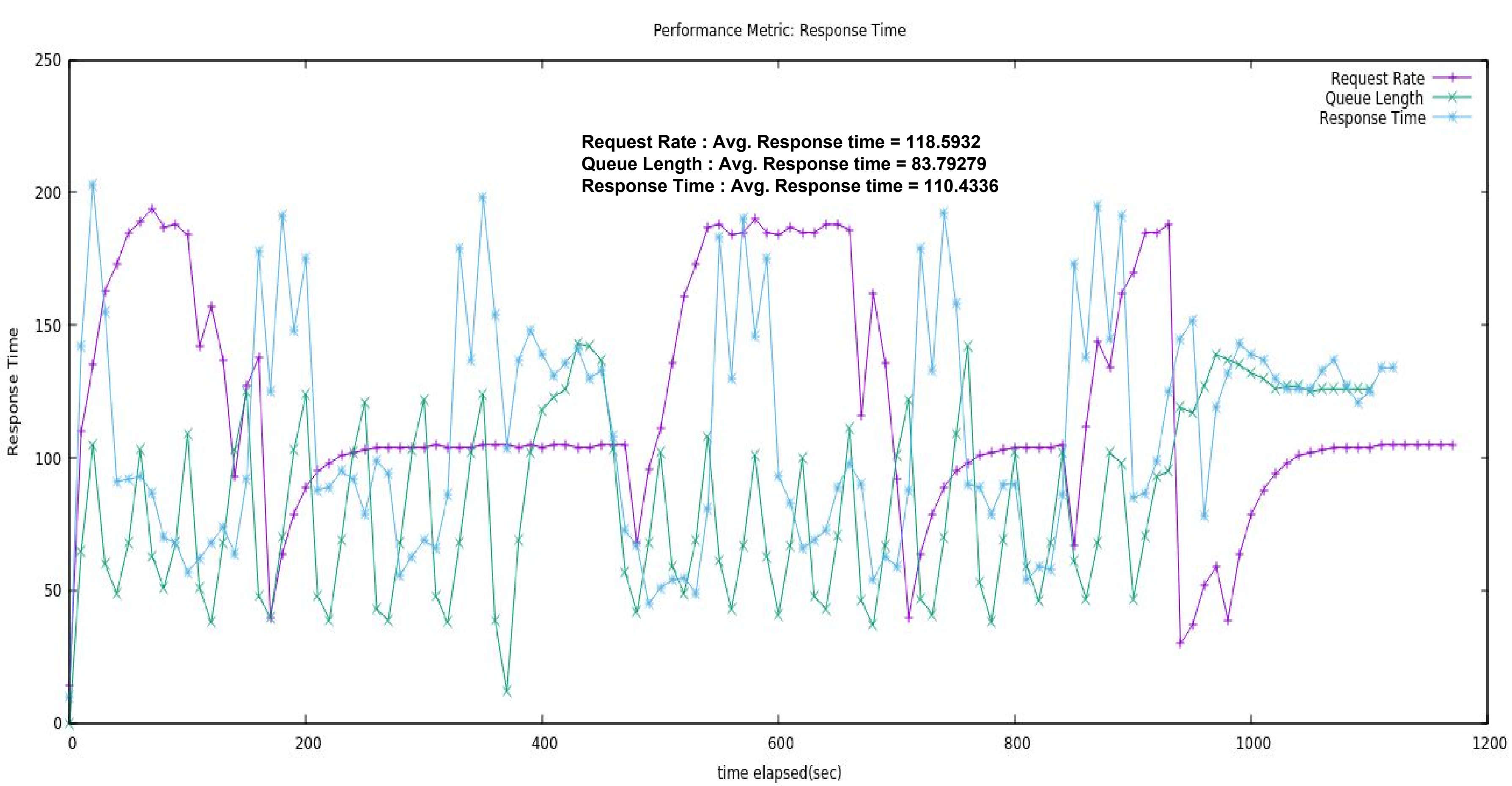
Response Time-aggressive, dynamic trace



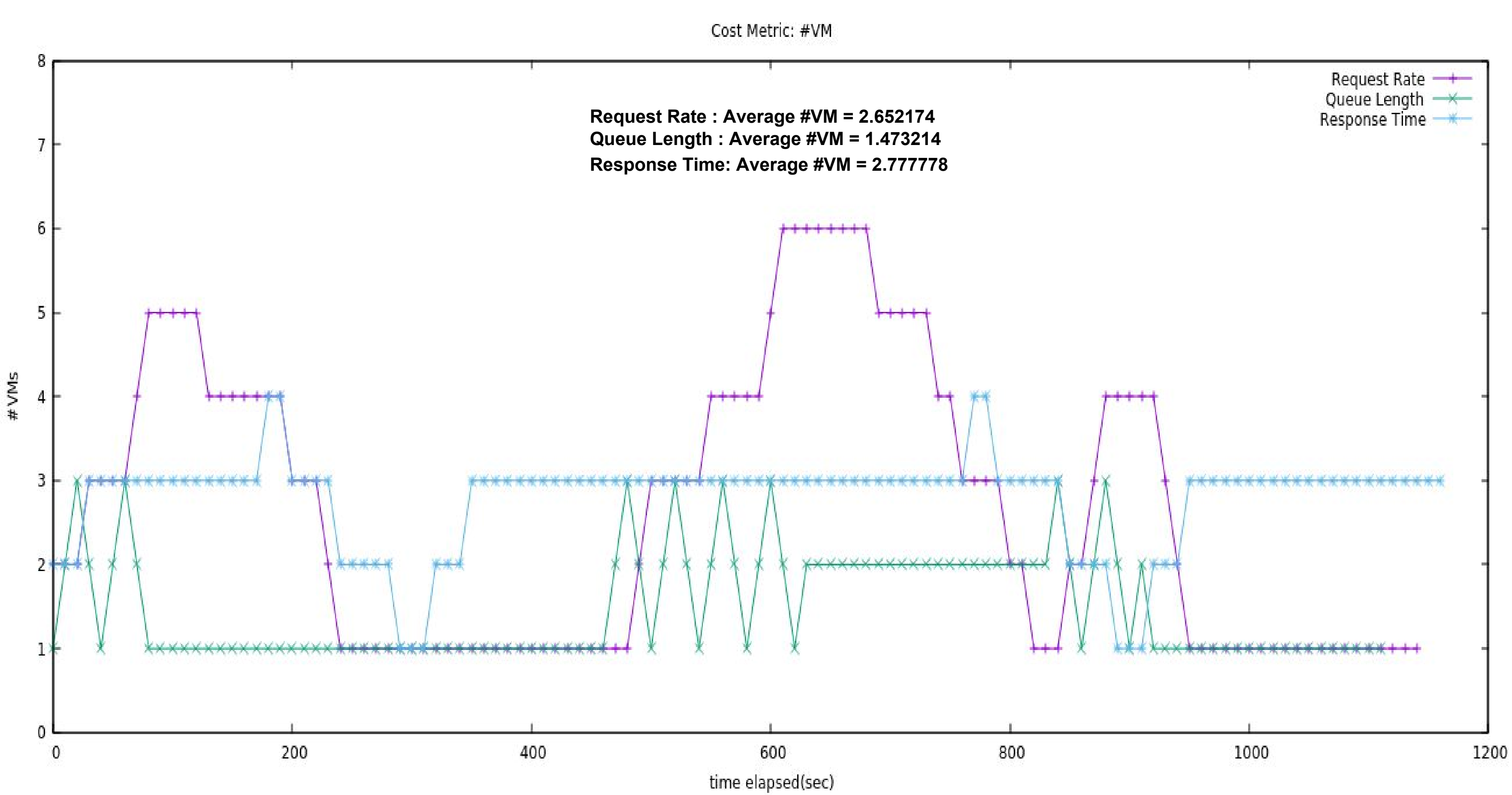
Performance Metrics: conservative, dynamic



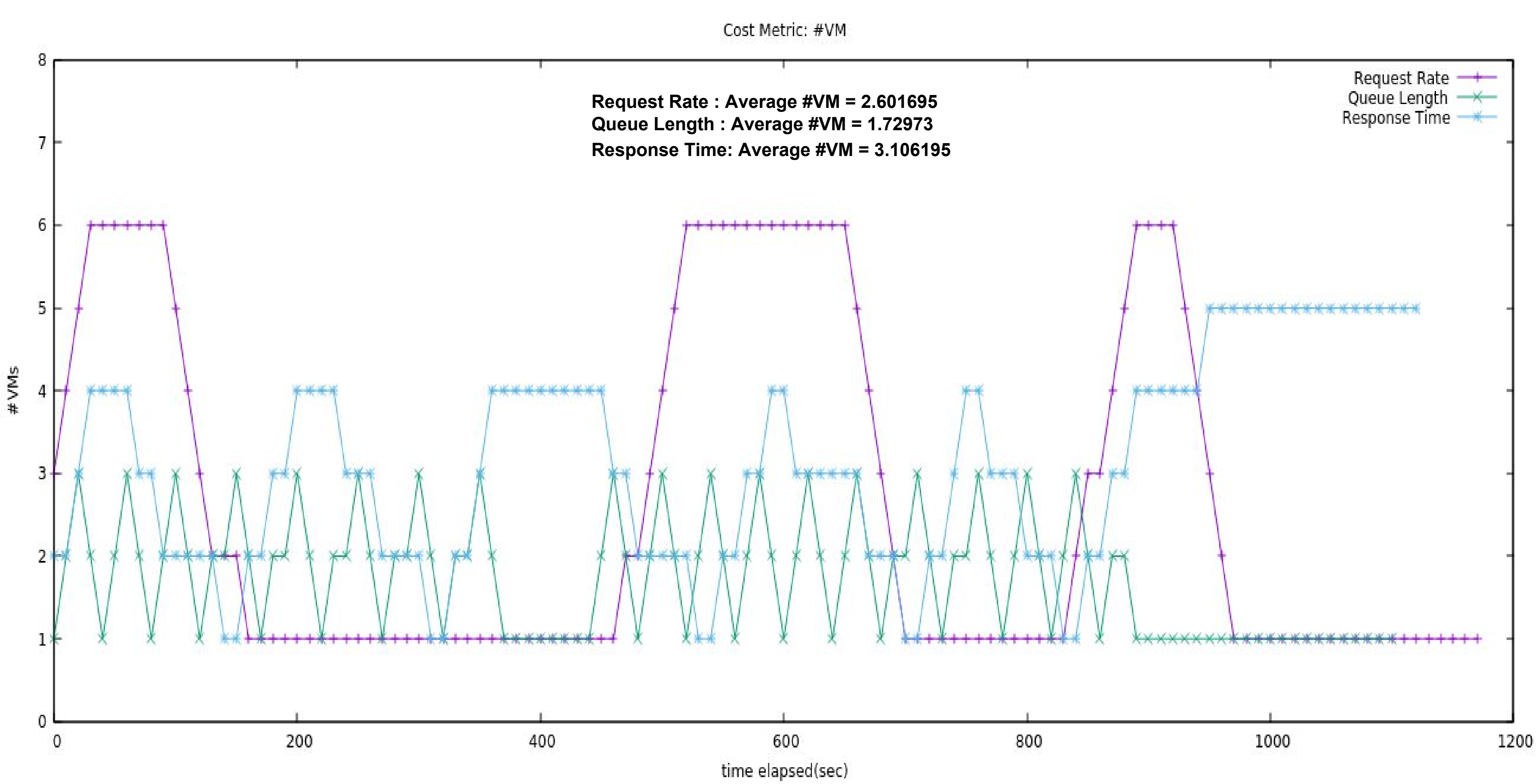
Performance Metrics: Aggressive, dynamic



Cost Metrics: conservative, dynamic trace



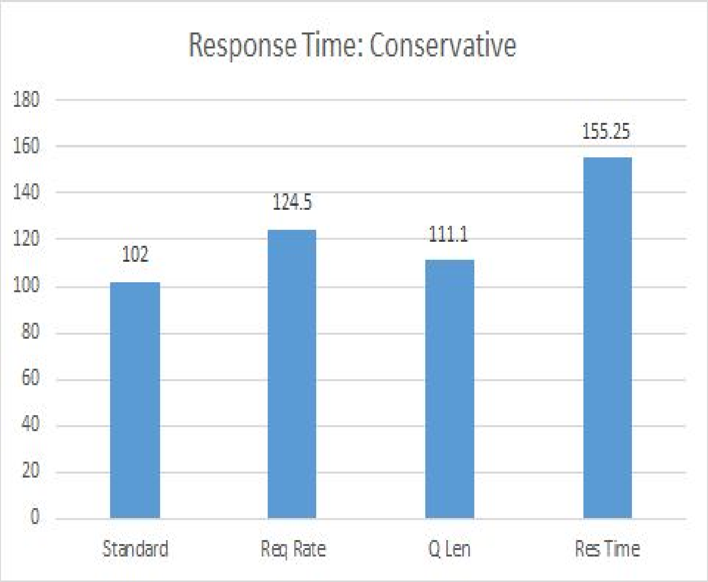
Cost Metrics: Aggressive, dynamic trace

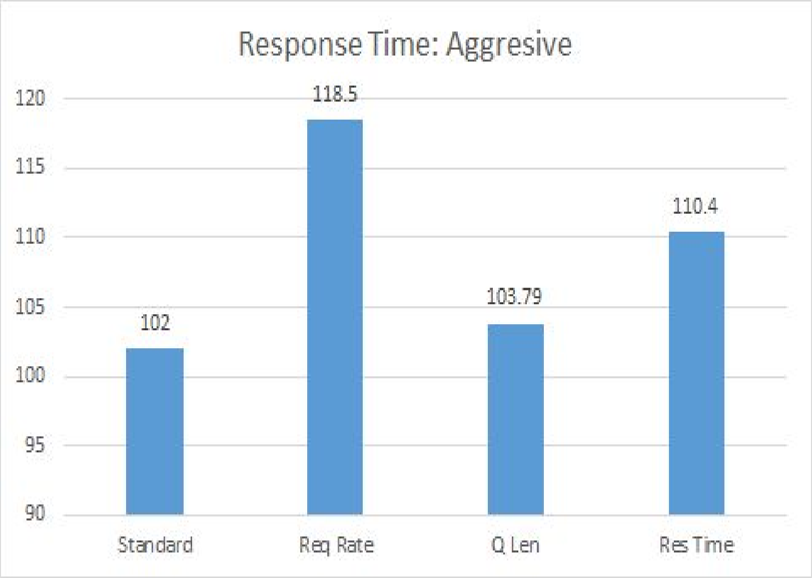


Summary from graphs

	Static Trace	Dynamic trace	
		Conservative	Aggressive
Cost(#of VMs)	Response time	Queue length	Queue length
Performance	Request rate	Queue length	Queue length

Benchmarking





Conclusion

- Response time and queue length are found complementary to each other
  - Increase in queue length leads to rise in response time
- Wear and tear is:
  - More for all metrics in aggressive approach
  - More for autoscaling based on request rate
- For bursty loads (dynamic traces), autoscaling based on queue length is favorable.

References

- https://cbonte.github.io/haproxy-dconv/configuration-1.5.html
- http://engineeringblog.yelp.com/2015/04/true-zero-downtime-haproxy-reloads.html
- http://www.mervine.net/performance-testing-with-httperf
- https://httpd.apache.org/docs/2.4/
- https://cloud.google.com/compute/docs/autoscaler/

Code & experimental data Repo:

- https://github.com/waytoalpit/AutoScale_Haproxy

Thank you!