Detecting *Latency Degradation Patterns* in *Service-based Systems*

Vittorio Cortellessa **Luca Traini**
University of L'Aquila, Italy

11th ACM/SPEC International Conference on Performance Engineering

Challenges in Modern Distributed Systems

Move fast (Rubin and Rinard, 2016) vs *Performance assurance*

Several performance issues come out only with real live user traffic (Veeraraghavan et al., 2016).

Julia Rubin and Martin Rinard. 2016. The challenges of staying together while moving fast: an exploratory study. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16). Association for Computing Machinery, New York, NY, USA, 982–993. DOI:https://doi.org/10.1145/2884781.2884871

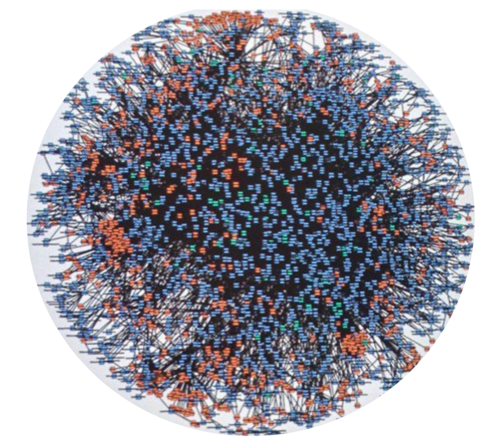
Kaushik Veeraraghavan, Justin Meza, David Chou, Wonho Kim, Sonia Margulis, Scott Michelson, Rajesh Nishtala, Daniel Obenshain, Dmitri Perelman, and Yee Jiun Song. 2016. Kraken: leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services. In Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation (OSDI'16). USENIX Association, USA, 635–650.

Performance debugging in production

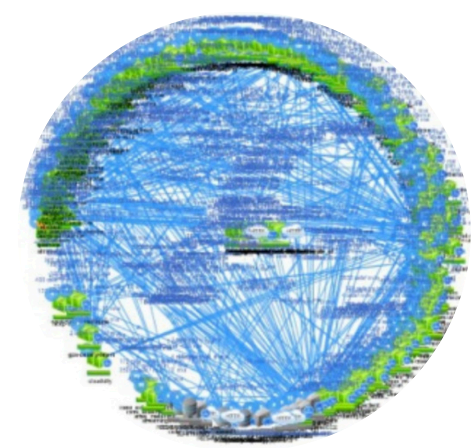
Fundamental activity during software evolution

Challenge: A request triggers several Remote Procedure Calls (RPC)

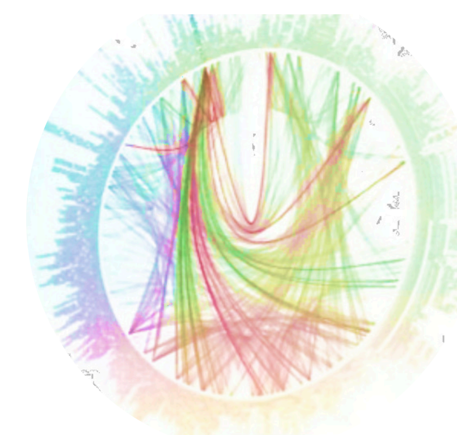
Availability of workflow-centric solutions
(e.g. Zipkin[1], Jaeger[2])



Amazon



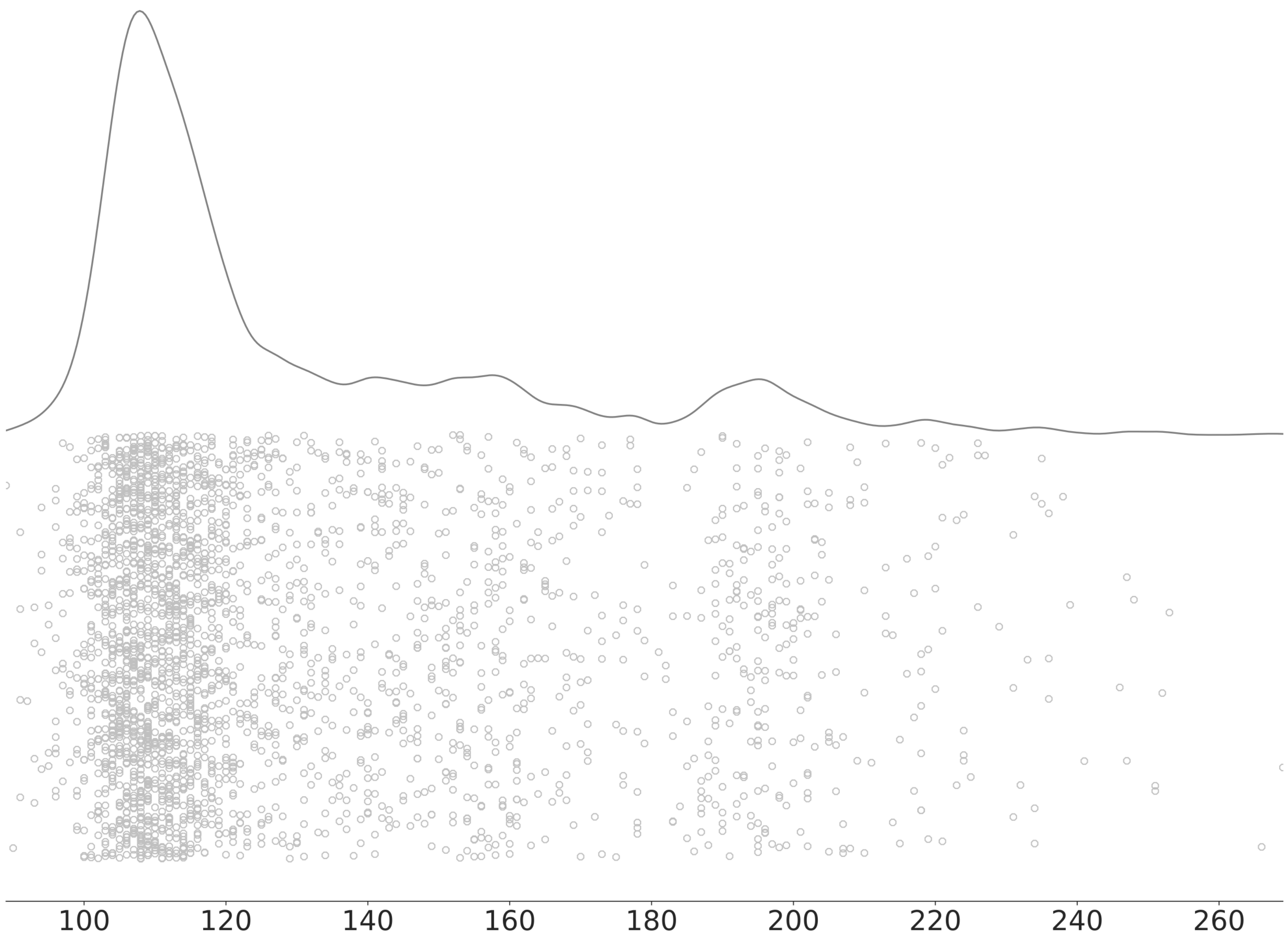
Netflix



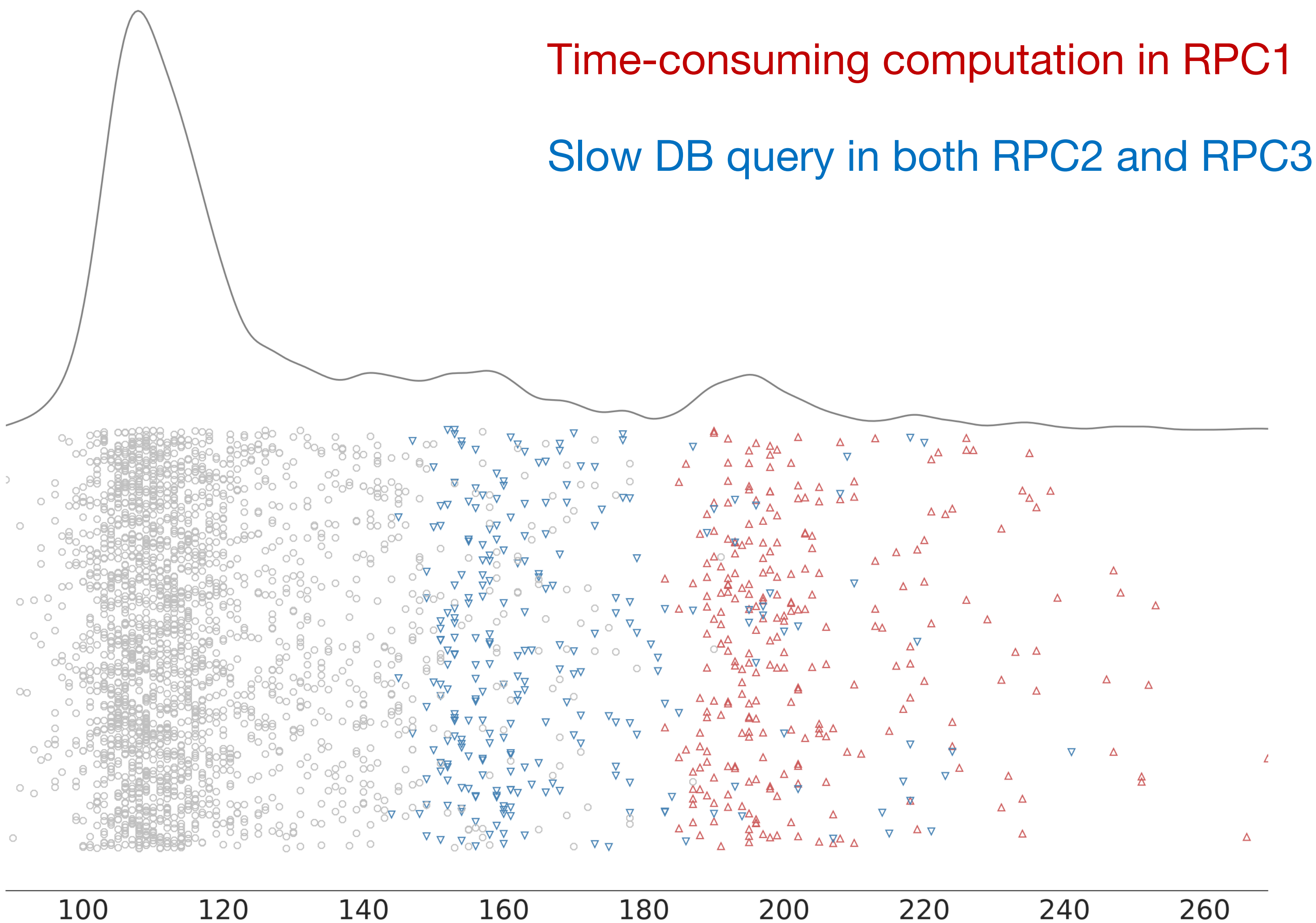
Twitter

[1] https://zipkin.io/
[2] https://www.jaegertracing.io/

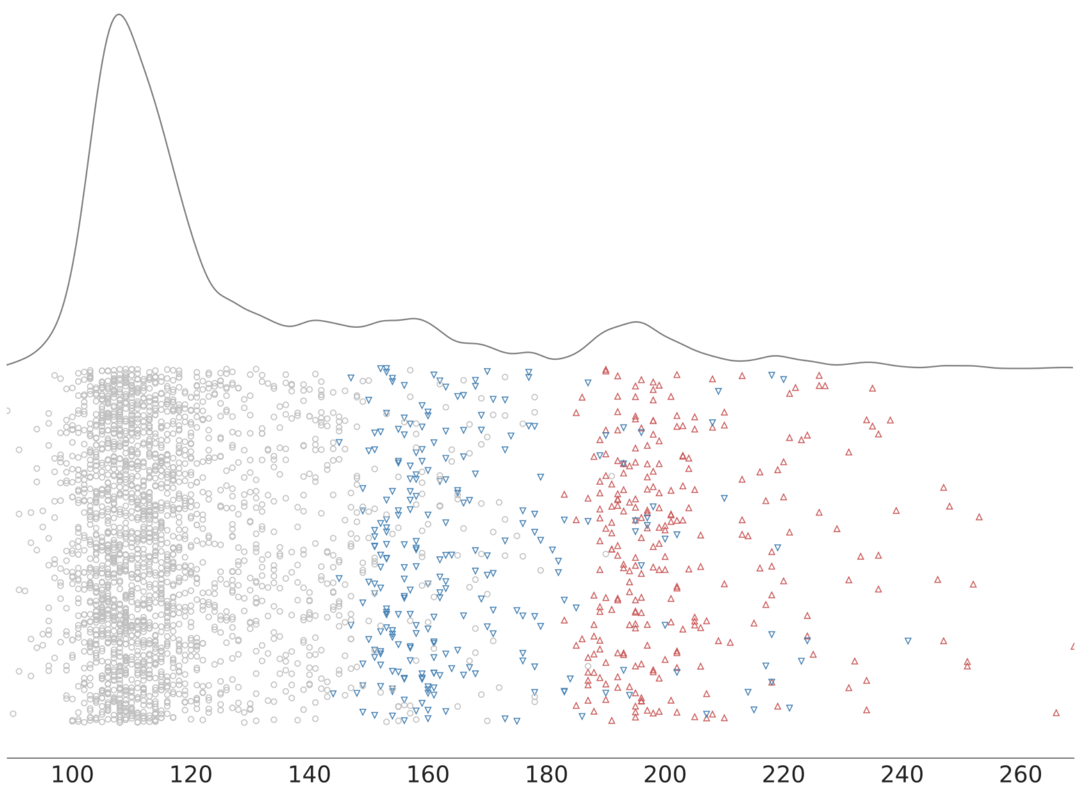
Triaging requests



Triaging requests



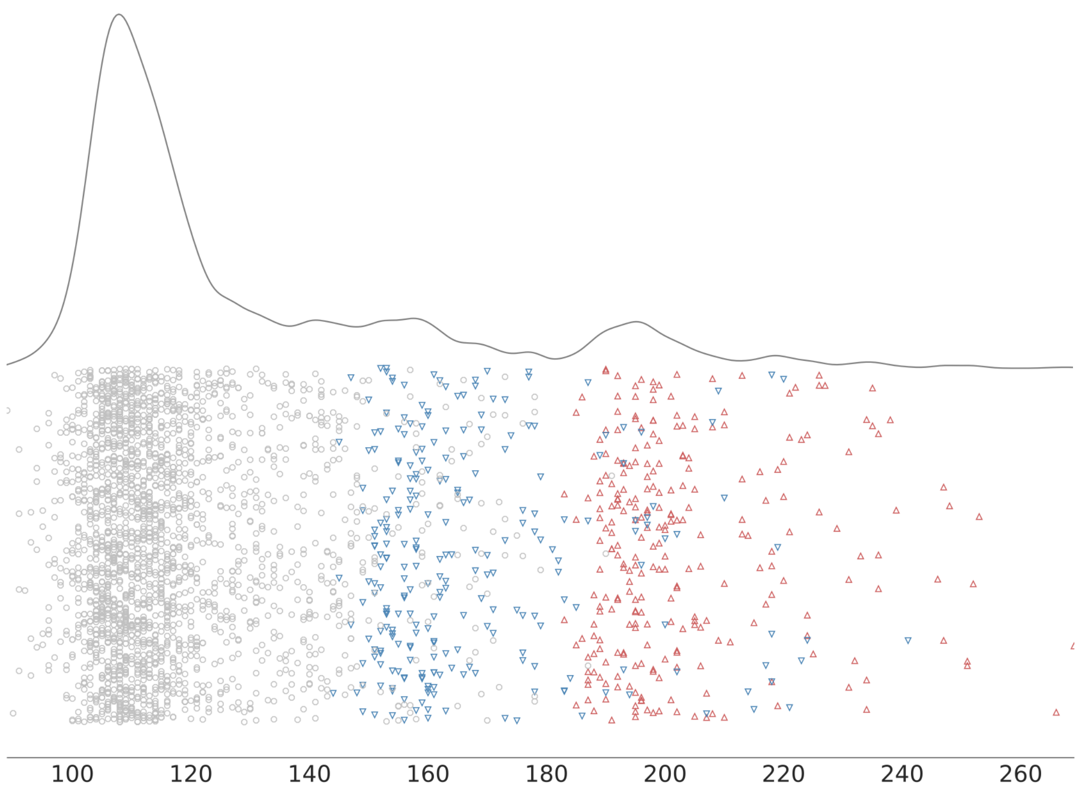
Latency Degradation Patterns



getProfile execution time is > 30ms

getRecommended execution time is > 20ms AND getCart execution time is > 15ms

Latency Degradation Patterns



condition

pattern

getProfile execution time is > 30ms

getRecommended execution time is > 20ms AND getCart execution time is > 15ms

Formal Notation

A *request trace* is denoted as :

$$r = (e_0, e_1, ..., e_m, L)$$

where e_i denotes the execution time of the RPC i and L the entire request latency

A *condition* is denoted as:

$$c = \langle j, e_{min}, e_{max} \rangle$$

where j refers to the RPC j

A *request trace* r satisfies c denoted as

$r \lhd c$ if $r = (..., e_j, ...)$ and $e_{min} \leq e_j < e_{max}$

Formal Notation

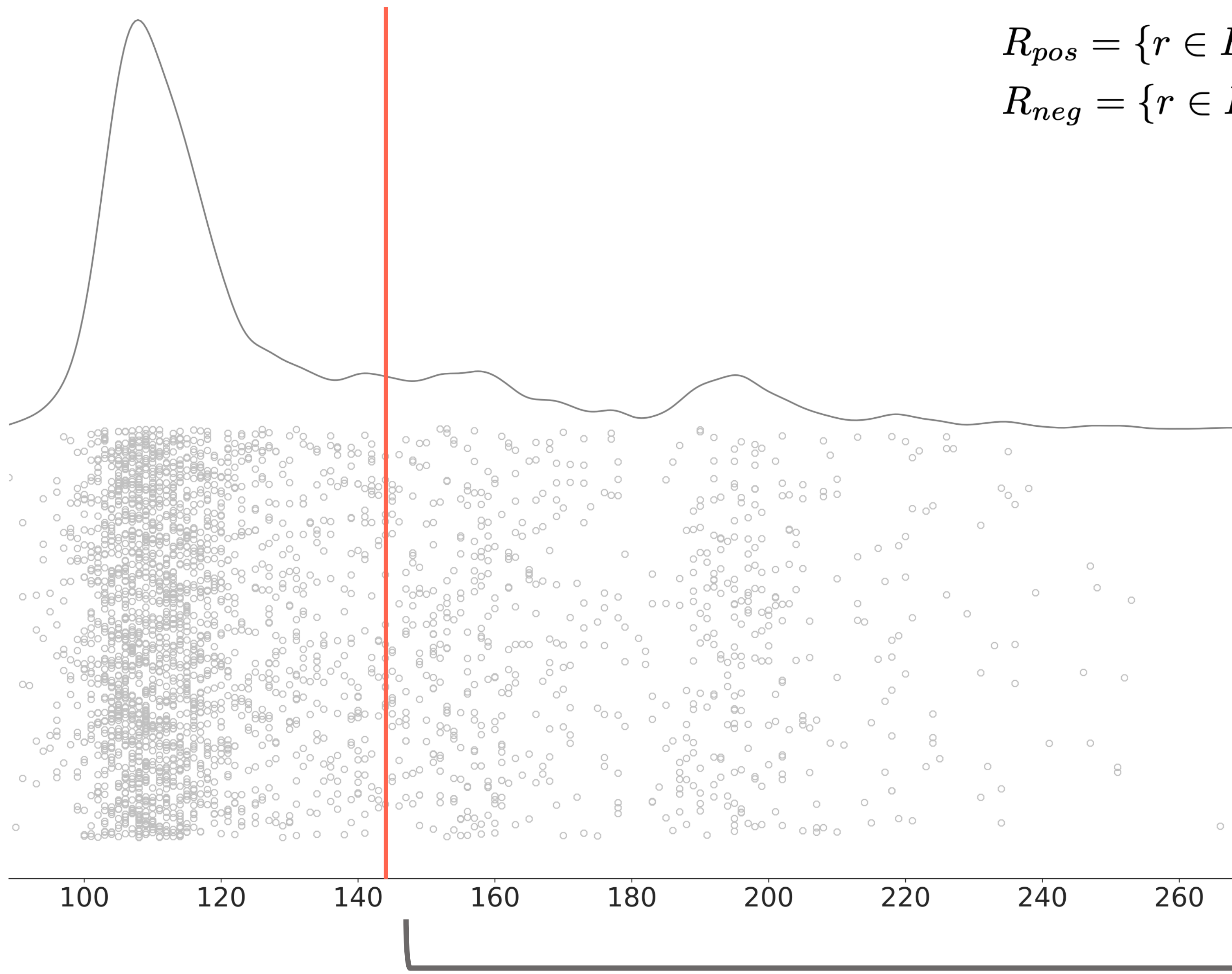
A *pattern* is denoted :

$P = \{c_0, c_1, ..., c_k\}$ where c_i is a *condition* and $k > 0$

A *request trace* ***r*** satisfies a *pattern* ***P*** denoted as

$r \lhd P$ if $\forall c \in P,\ r \lhd c$

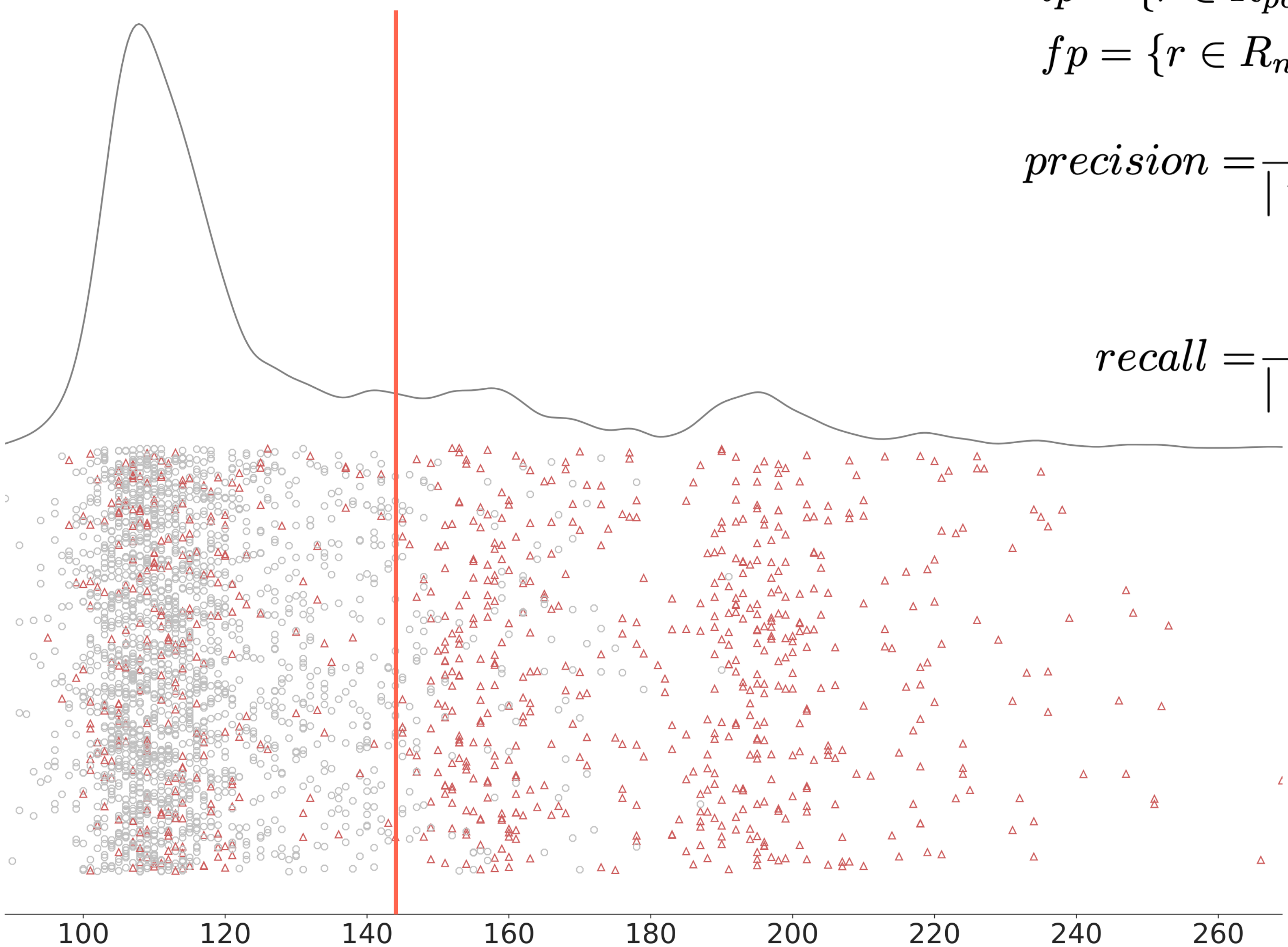
Formal notation



$$R_{pos} = \{r \in R \mid L \in I\}$$
$$R_{neg} = \{r \in R \mid L \notin I\}$$

latency interval considered as *degraded*
denoted as I

Precision and recall



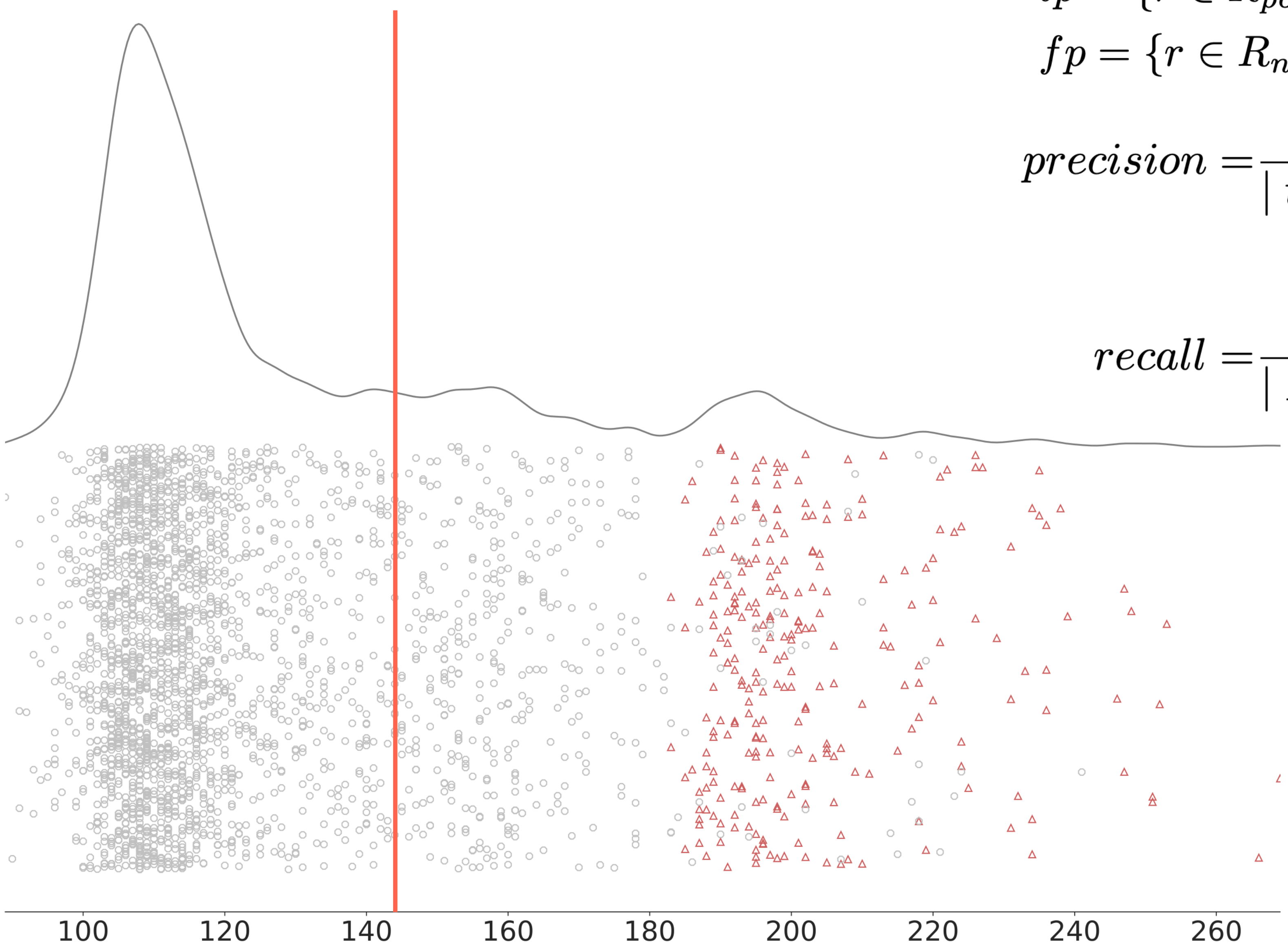
$$tp = \{r \in R_{pos} \mid r \lhd \mathbf{P}\}$$

$$fp = \{r \in R_{neg} \mid r \lhd \mathbf{P}\}$$

$$precision = \frac{\mid tp \mid}{\mid tp \mid + \mid fp \mid}$$

$$recall = \frac{\mid tp \mid}{\mid R_{pos} \mid}$$

Precision and recall



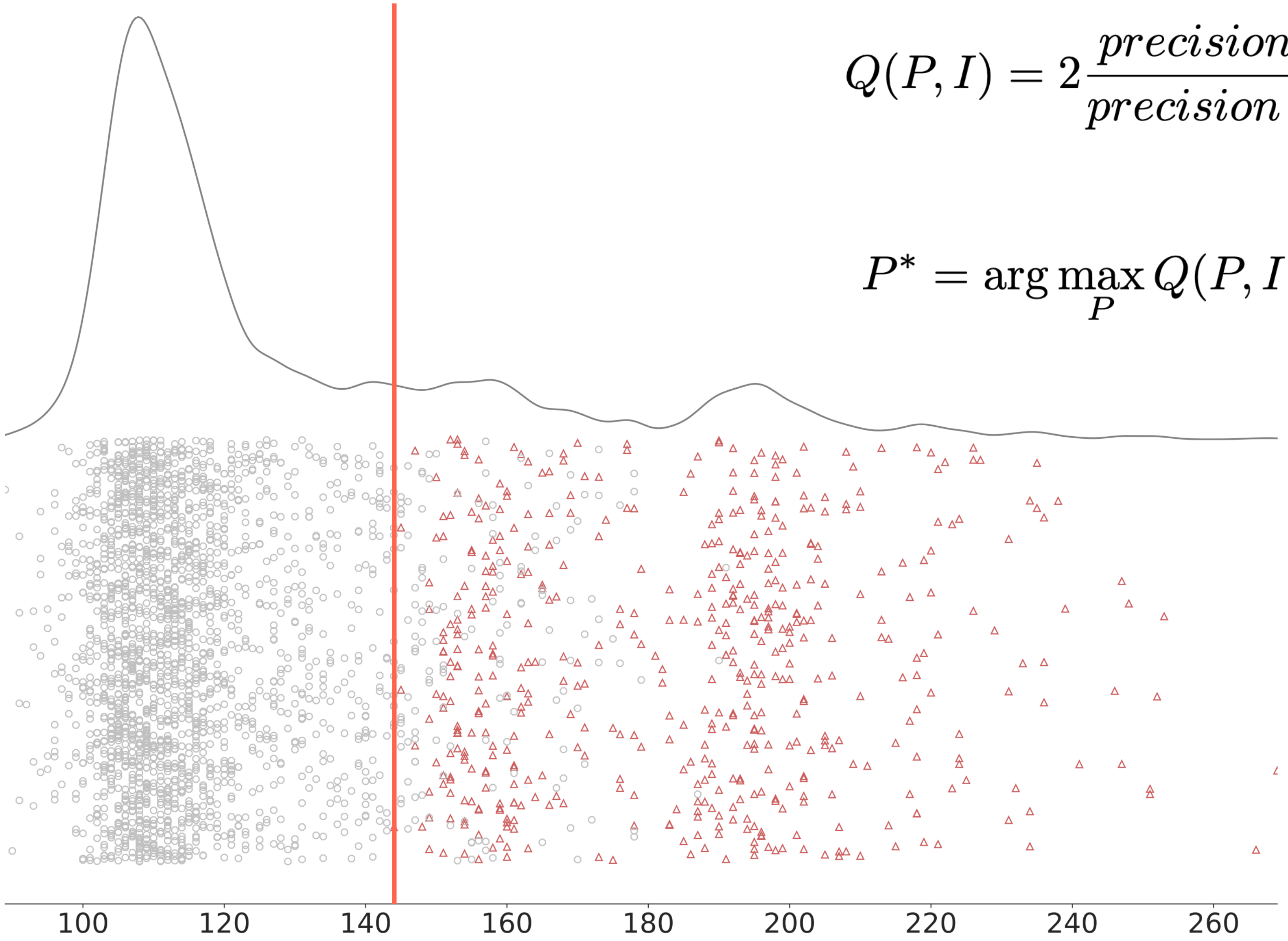
$$tp = \{r \in R_{pos} \mid r \lhd \mathbf{P}\}$$

$$fp = \{r \in R_{neg} \mid r \lhd \mathbf{P}\}$$

$$precision = \frac{\mid tp \mid}{\mid tp \mid + \mid fp \mid}$$

$$recall = \frac{\mid tp \mid}{\mid R_{pos} \mid}$$

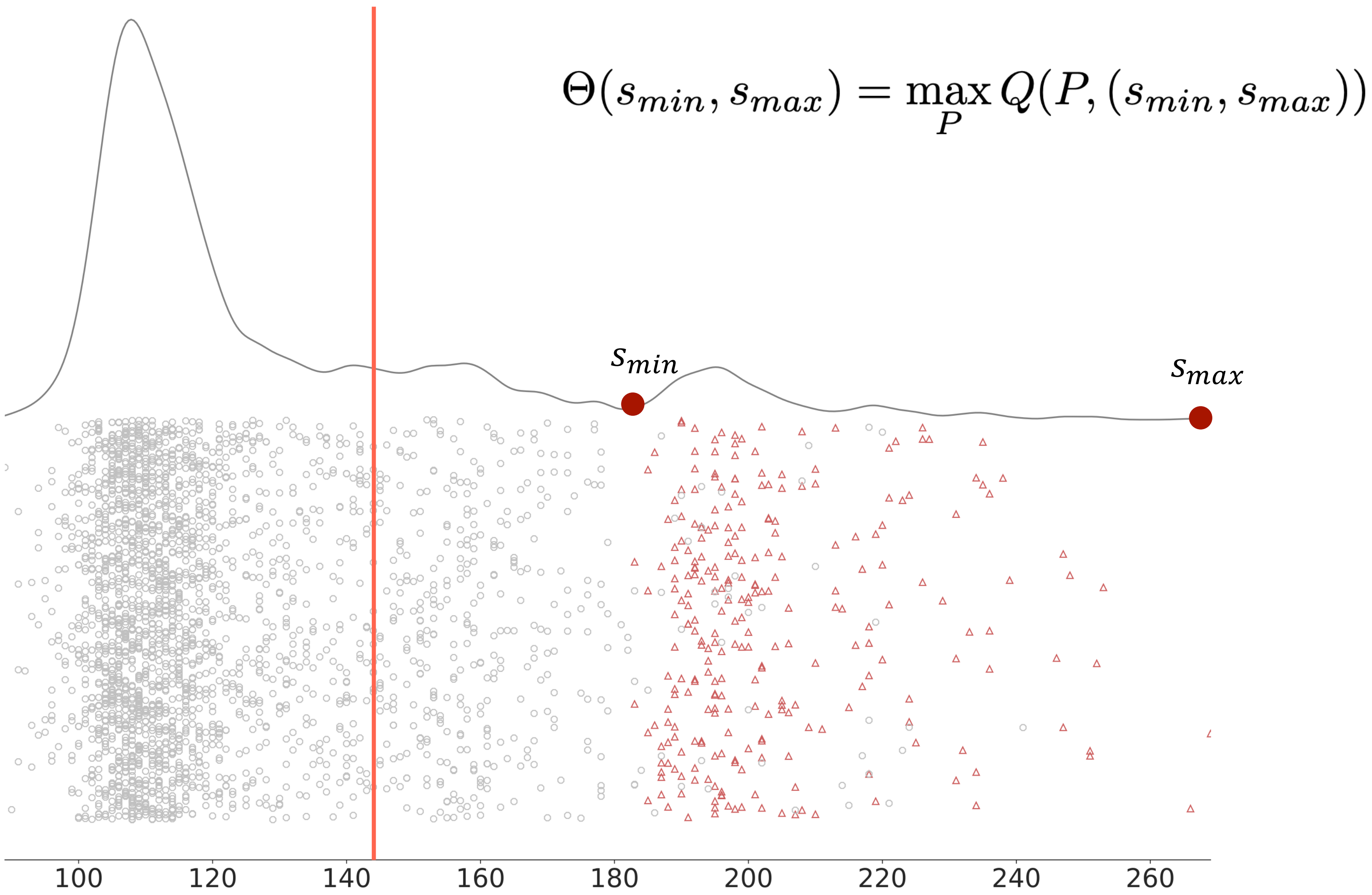
F-score



$$Q(P, I) = 2\frac{precision \cdot recall}{precision + recall}$$

$$P^* = \arg\max_P Q(P, I)$$

Sub-interval analysis

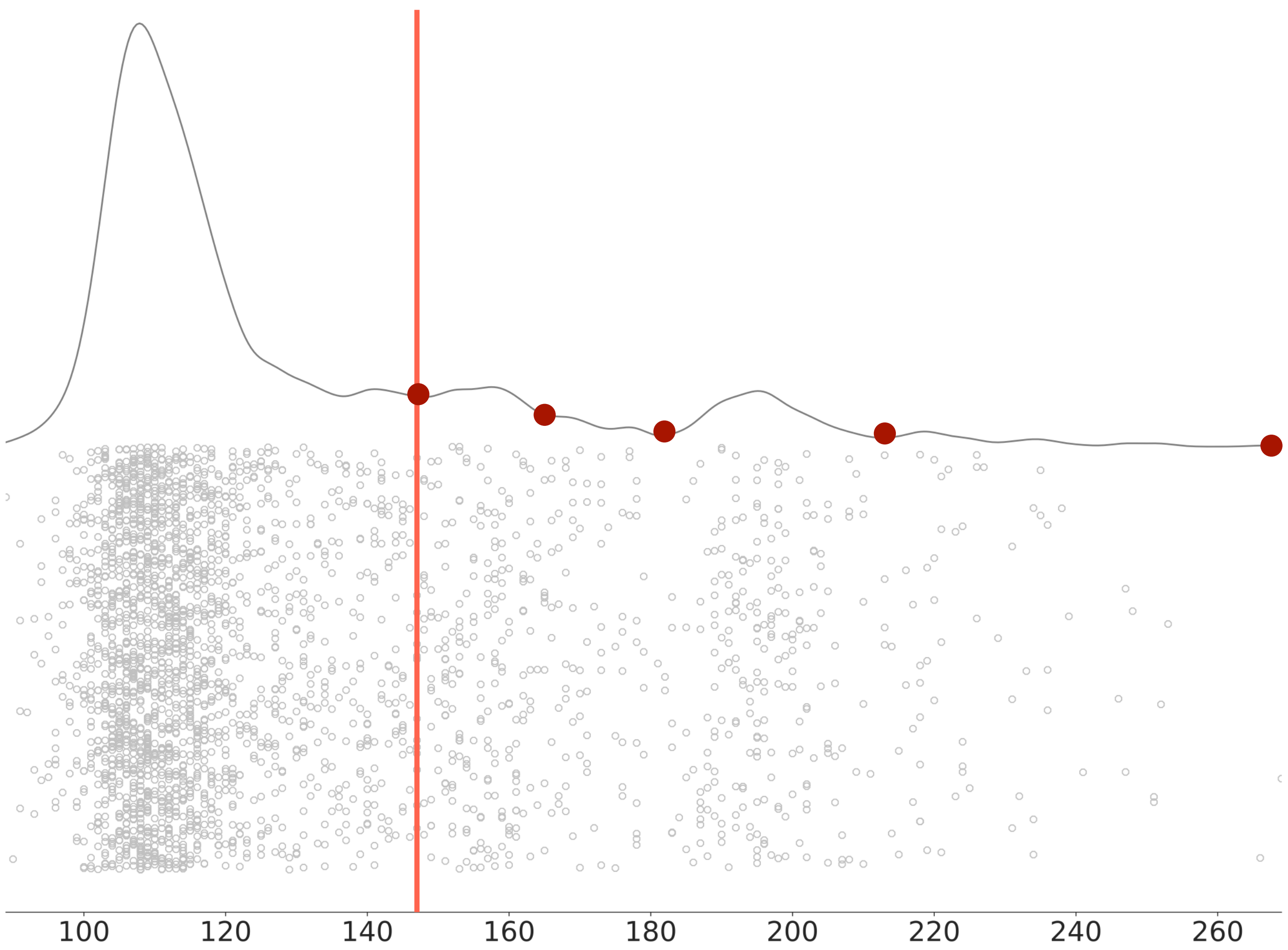


Splitting the interval

We split the latency range with a set of potential split points:

$$\{s_0, s_1, ..., s_k\}$$

Split points are derived using Mean Shift (Comaniciu and Meer, 2002)



Dorin Comaniciu and Peter Meer. 2002. Mean Shift: A Robust Approach Toward Feature Space Analysis. IEEE Trans. Pattern Anal. Mach. Intell. 24, 5 (May 2002), 603–619. DOI:https://doi.org/10.1109/34.1000236

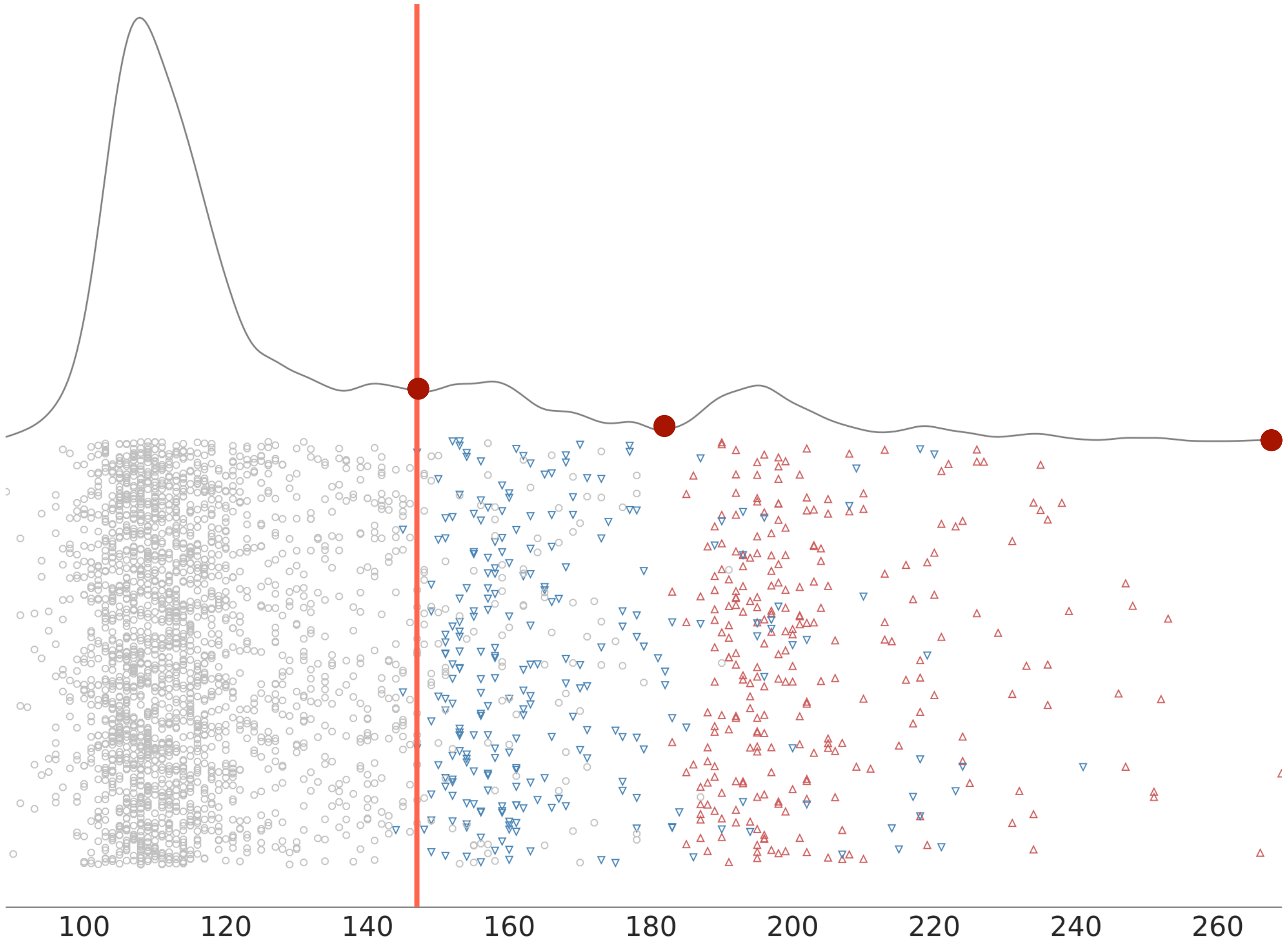
Optimal split

Find subset of split points

$$\{s_0^*, s_1^*, \dots, s_z^*\}$$

such that

$$\sum_{i=0}^{z-1} \Theta(s_i^*, s_{i+1}^*)$$



Optimization Problem

Main problem

$$\max_{s_0^*,\dots,s_z^*} \sum_{i=0}^{z-1} \Theta(s_i^*, s_{i+1}^*)$$

Sub-problem

$$\Theta(s_i, s_j) = \arg\max_{P} Q(P, (s_i, s_j))$$

Main Problem: Dynamic Programming approach

(Krushevskaja and Sandler, 2013)

$D(i)$ denote the best score for a solution that covers interval $[s_0, s_i)$

$D(0) = 0$ initial score

The update step is:

$$D(i) = \max_{0 \leq j < i} (D(j) + \Theta(s_j, s_i))$$

Darja Krushevskaja and Mark Sandler. 2013. Understanding latency variations of black box services. In Proceedings of the 22nd international conference on World Wide Web (WWW '13). Association for Computing Machinery, New York, NY, USA, 703–714. DOI:https://doi.org/10.1145/2488388.2488450

Sub-problem: Genetic Algorithm

j	e_{min}	e_{max}

condition

Representation

c_1	c_2	…	c_k

pattern

Fitness

$$Q(P, I) = 2\frac{precision \cdot recall}{precision + recall}$$

Mutation

randomly ***add***/***remove***/***change*** a *condition*

Crossover

merge P_1 and P_2 in $P_U = P_1 \cup P_2$, then randomly ***split*** P_U in P_1' and P_2'

Evolution strategy

$(\mu + \lambda)$ *evolution strategy*[1]

Hans-Georg Beyer and Hans-Paul Schwefel. 2002. Evolution strategies –A comprehensive introduction. Natural Computing: an international journal 1, 1 (May 2002), 3–52. DOI:https://doi.org/10.1023/A:1015059928466

Fitness evaluation

Performance critical operation

$$tp = \{r \in R_{pos} \mid r \lhd P\}$$
$$fp = \{r \in R_{neg} \mid r \lhd P\}$$

Checking a set of inequalities

RPC1	*RPC2*	*...*	*RPCn*	*L*
300	220	…	120	490
330	250	…	125	530
…	….	…	…	…
320	235	…	140	495
350	230	…	130	500

Optimizing fitness evaluation

Intuition:
Same checks are repeated several times during the evolution process

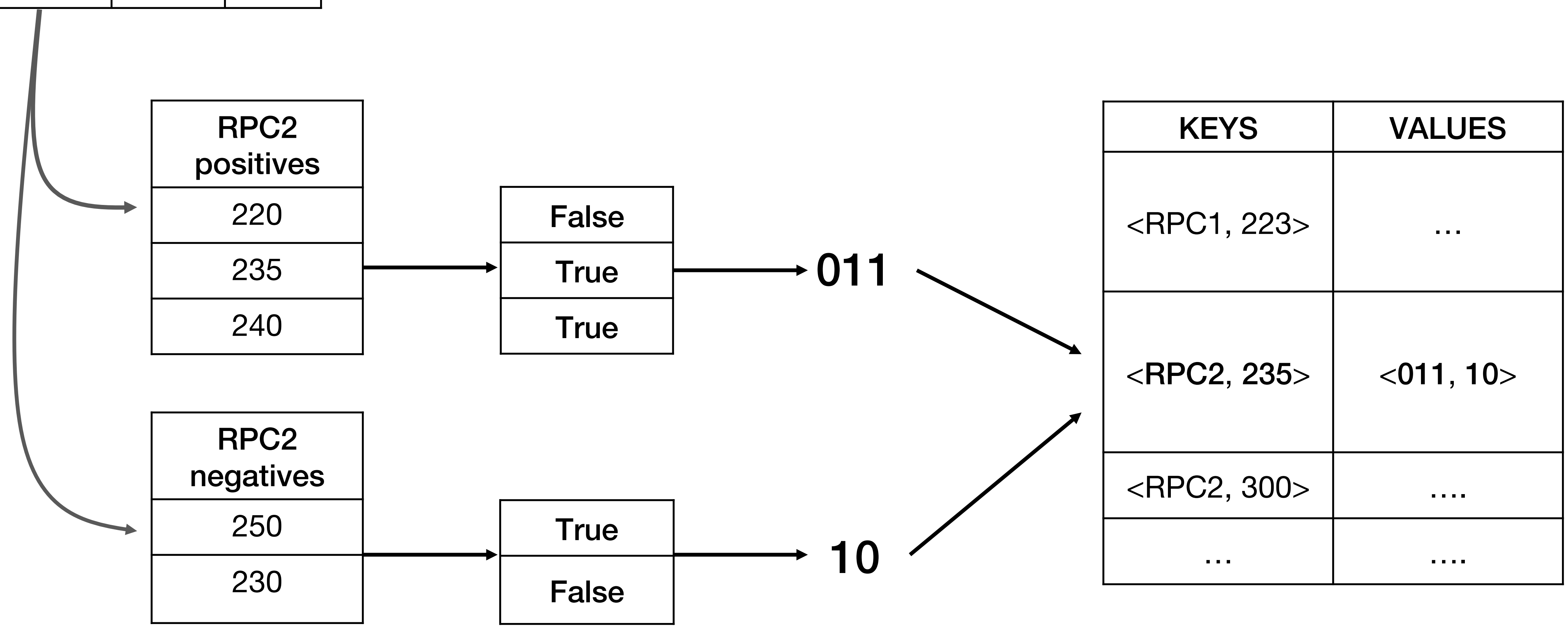
Our Solution:
Meaningful checks are computed and stored upfront and then reused during the evolution process

Precomputation

RPC1	RPC2	RPC3	L
300	220	120	490
330	250	125	530
320	235	140	495
350	230	130	510
340	240	125	515

RPC2 execution time ≥ 235

$$(s_{min}, s_{max}) = (500, 600)$$



Fast fitness evaluation using bitwise operators

$P = \{c_0, c_1, ..., c_k\}$ $\qquad c_i = \langle j, e_{min}, e_{max} \rangle$

$$r \lhd c_i \iff e_{min} \leq e_j < e_{max}$$

$B^{pos}_{c_i} = B^{pos}_{min} \wedge \neg B^{pos}_{max}$ $\qquad B^{neg}_{c_i} = B^{neg}_{min} \wedge \neg B^{neg}_{max}$

$$tp = \bigwedge_{c \in P} B^{pos}_c \qquad fp = \bigwedge_{c \in P} B^{neg}_c$$

KEYS	VALUES
$< j, e_{min} >$	$< B^{pos}_{min}, B^{neg}_{min} >$
$< j, e_{max} >$	$< B^{pos}_{max}, B^{neg}_{max} >$
…	….

Search Space Reduction

$$\langle B_e^{pos}, B_e^{neg} \rangle \;\; \forall \langle RPC, e \rangle$$

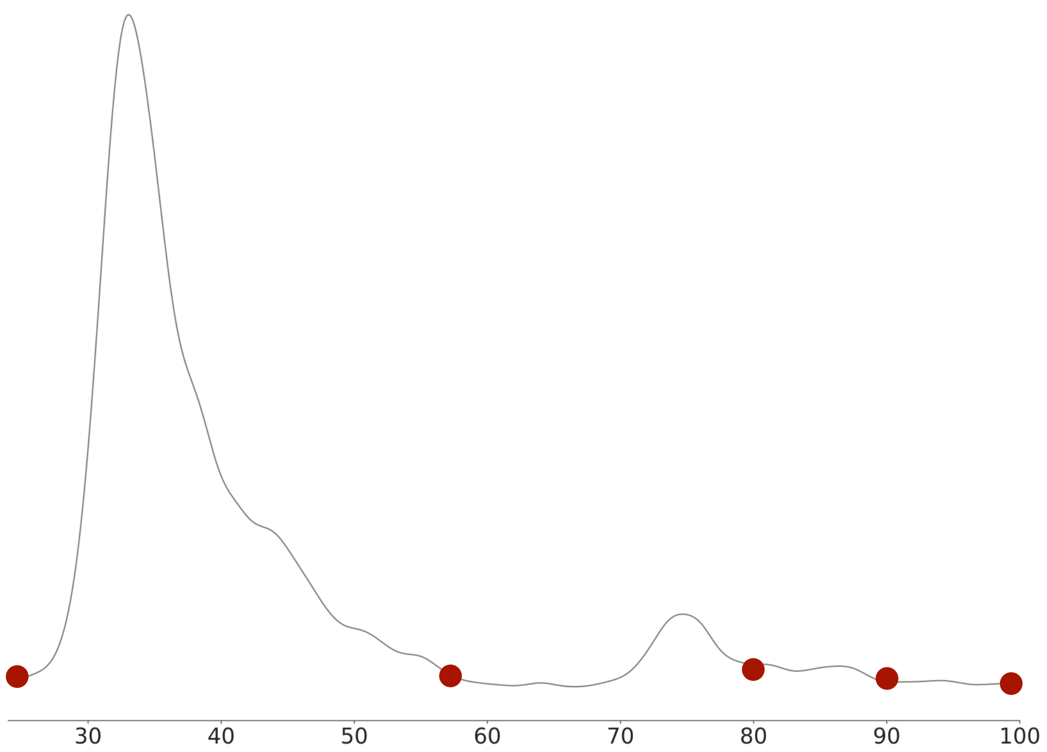
Problem:

Tremendous precomputing effort

Solution:

Consider only meaningful RPC execution times

- Mean Shift algorithm (Comaniciu and Meer, 2002) -



Dorin Comaniciu and Peter Meer. 2002. Mean Shift: A Robust Approach Toward Feature Space Analysis. IEEE Trans. Pattern Anal. Mach. Intell. 24, 5 (May 2002), 603–619. DOI:https://doi.org/10.1109/34.1000236

Research Questions

RQ1 Is our approach effective for clustering requests associated to the same latency degradation pattern, as compared to machine learning algorithms?

RQ2 Is our approach effective with respect to state-of-the-art approaches for latency profile analysis?

RQ3 How robust is our approach to "noise"?

RQ4 What is the efficiency of our approach as compared to other ones?

Experimental setup

Case of study E-commerce application[1], composed by 9 microservices (Spring Cloud[2]). Zipkin[3] is used as distributed tracing solution. Spans are stored on ElasticSearch[4]. Request under analysis involves 13 RPCs (8 unique) over among 5 microservices.

Data generation 60 load testing sessions of 5 minutes with 2 randomly injected artificial degradations. 30 using *normal artificial degradations* and 30 using *noised artificial degradations.* Each load test session generate ~1000 requests

Artificial degradation pattern *Normal*: inject 50ms in subset of RPCs for 10% of traffic
Noised: inject 50ms with some noise in subset of RPCs for 10% of traffic + inject 50ms to portion of async RPCs

[1] https://github.com/SEALABQualityGroup/E-Shopper
[2] https://spring.io/projects/spring-cloud
[3] https://zipkin.io/
[4] https://www.elastic.co/elasticsearch/

Baselines

- **K-means** (MacQueen, 1967)
- **Hierarchical** (Rokach and Maimon, 2005)
- **Mean shift** (Comaniciu and Meer, 2002)
- **Branch and bound** (Krushevskaja and Sandler, 2013)

MacQueen, J. Some methods for classification and analysis of multivariate observations. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, 281--297, University of California Press, Berkeley, Calif., 1967.

Lior Rokach and Oded Maimon. 2005. Clustering Methods. Springer US, Boston, MA, 321–352. https://doi.org/10.1007/0-387-25465-X_15

Dorin Comaniciu and Peter Meer. 2002. Mean Shift: A Robust Approach Toward Feature Space Analysis. IEEE Trans. Pattern Anal. Mach. Intell. 24, 5 (May 2002), 603–619. DOI:https://doi.org/10.1109/34.1000236

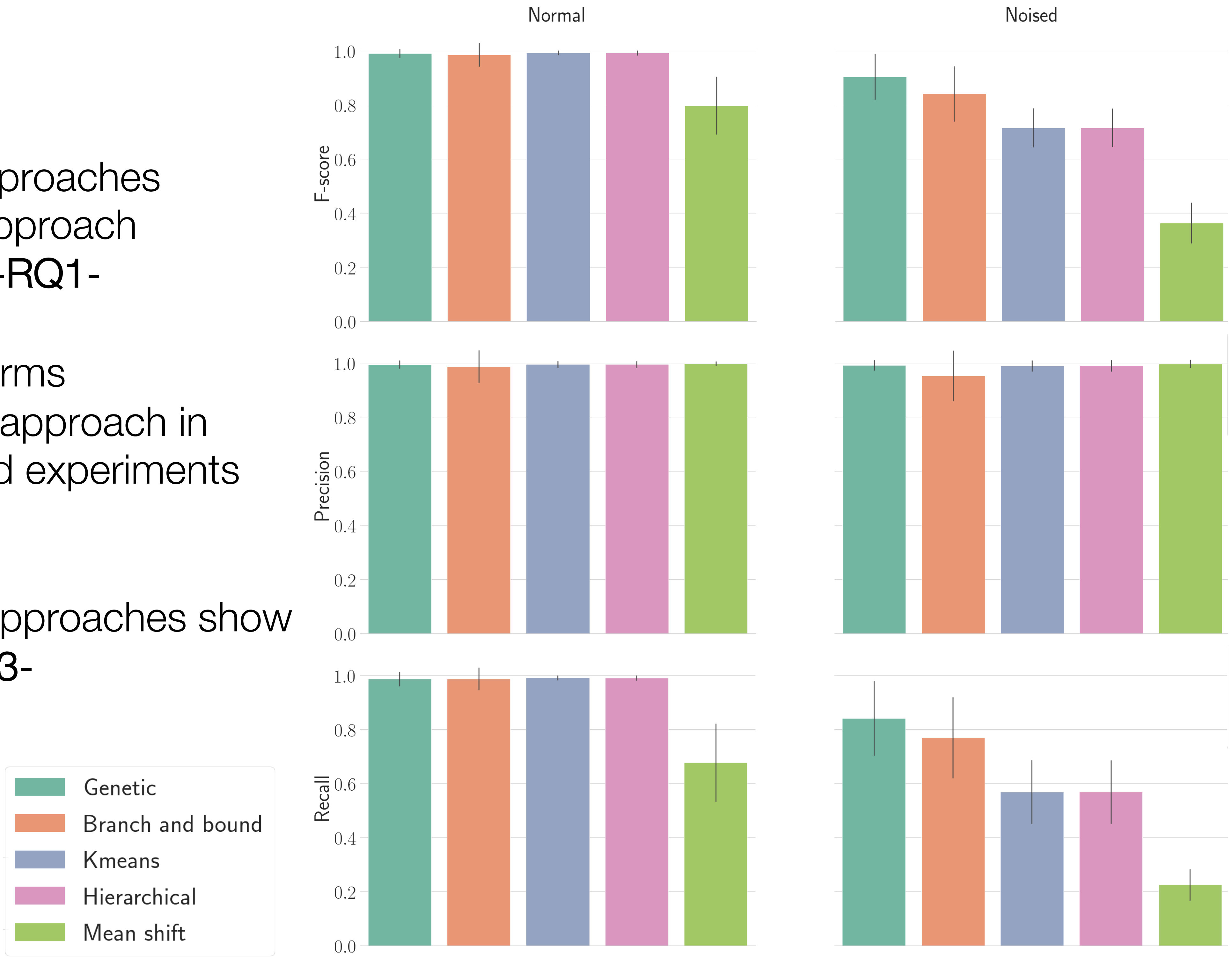
Darja Krushevskaja and Mark Sandler. 2013. Understanding latency variations of black box services. In Proceedings of the 22nd international conference on World Wide Web (WWW '13). Association for Computing Machinery, New York, NY, USA, 703–714. DOI:https://doi.org/10.1145/2488388.2488450

Results: effectiveness

Common clustering approaches outperformed by our approach in noised experiments.-RQ1-

Our approach outperforms the branch and bound approach in both normal and noised experiments ($p < 0.05$) –RQ2-

Both domain specific approaches show resiliency to noise –RQ3-



Results: efficiency

Machine learning methods are faster than optimization-based approaches

Noise slow down the efficiency in both combinatorial methods

	Normal		Noised	
Approach	Execution time mean (sec)	Execution time standard deviation (sec)	Execution time mean (sec)	Execution time standard deviation (sec)
Genetic	17.551	3.390	37.106	21.143
Branch and bound	49.070	12.820	102.259	89.291
Kmeans	0.024	0.006	0.025	0.005
Hierachical	0.004	0.001	0.004	0.001
Mean shift	0.509	0.08	0.307	0.039

Our approach is faster than state-of-the-art approach in both normal and noised experiments –RQ4-

Conclusion and future works

Results shows that our approach outperforms existing approaches, especially when RPC execution time is not very regular

We plan to deeply investigate the application of our approach to other distributed systems, ever more chaotic.

Thoroughly study and improve the scalability of the approach

Generalize the approach to other trace attributes other than RPC execution time