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

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


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\(^2\))

\(^1\) https://zipkin.io/
\(^2\) https://www.jaegertracing.io/
Triaging requests
Triaging requests

Time-consuming computation in RPC1

Slow DB query in both RPC2 and RPC3
Latency Degradation Patterns

getProfile execution time is > 30ms

getRecommended execution time is > 20ms AND getCart execution time is > 15ms
Latency Degradation Patterns

- 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:

\[ c = \langle j, e_{\text{min}}, e_{\text{max}} \rangle \]

where \( j \) refers to the RPC \( j \).

A request trace \( r \) satisfies \( c \) denoted as

\[ r \triangleleft c \quad \text{if} \quad r = (..., e_j, ...) \quad \text{and} \quad e_{\text{min}} \leq e_j < e_{\text{max}} \]
A pattern is denoted:

\[ P = \{ c_0, c_1, \ldots, c_k \} \]

where \( c_i \) is a condition and \( k > 0 \)

A request trace \( r \) satisfies a pattern \( P \) denoted as

\[ r \triangleleft P \quad \text{if} \quad \forall c \in P, \ r \triangleleft c \]
Formal notation

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

latency interval considered as \textit{degraded} denoted as \( I \)
Precision and recall

\[ tp = \{ r \in R_{pos} \mid r < P \} \]
\[ fp = \{ r \in R_{neg} \mid r < P \} \]
\[ \text{precision} = \frac{|tp|}{|tp| + |fp|} \]
\[ \text{recall} = \frac{|tp|}{|R_{pos}|} \]
Precision and recall

\[ tp = \{ r \in R_{pos} \mid r < P \} \]
\[ fp = \{ r \in R_{neg} \mid r < P \} \]

\[ \text{precision} = \frac{|tp|}{|tp| + |fp|} \]

\[ \text{recall} = \frac{|tp|}{|R_{pos}|} \]
F-score

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

\[ P^* = \arg \max_P Q(P, I) \]
Sub-interval analysis

\[ \Theta(s_{\text{min}}, s_{\text{max}}) = \max_{P} Q(P, (s_{\text{min}}, s_{\text{max}})) \]
Splitting the interval

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

\[ \{s_0, s_1, \ldots, s_k\} \]

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

---

Optimal split

Find subset of split points

\[ \{s_0^*, s_1^*, \ldots, s_z^*\} \]

such that

\[ \sum_{i=0}^{z-1} \Theta(s_i^*, s_{i+1}^*) \]
Optimization Problem

Main problem

$$\max_{s^*_0, \ldots, s^*_z} \sum_{i=0}^{z-1} \Theta(s^*_i, s^*_{i+1})$$

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:

$$D(i) = \max_{0 \leq j < i} \left( D(j) + \Theta(s_j, s_i) \right)$$
Sub-problem: Genetic Algorithm

**Representation**

<table>
<thead>
<tr>
<th>j</th>
<th>$e_{\text{min}}$</th>
<th>$e_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>condition</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>$c_2$</th>
<th>...</th>
<th>$c_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Fitness**

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

**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

---

Fitness evaluation

Performance critical operation

\[ tp = \{ r \in R_{pos} \mid r \triangleleft P \} \]

\[ fp = \{ r \in R_{neg} \mid r \triangleleft P \} \]

Checking a set of inequalities

<table>
<thead>
<tr>
<th></th>
<th>RPC1</th>
<th>RPC2</th>
<th>...</th>
<th>RPCn</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>220</td>
<td>...</td>
<td>120</td>
<td>490</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td>250</td>
<td>...</td>
<td>125</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>235</td>
<td>...</td>
<td>140</td>
<td>495</td>
<td></td>
</tr>
<tr>
<td>350</td>
<td>230</td>
<td>...</td>
<td>130</td>
<td>500</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>RPC1</th>
<th>RPC2</th>
<th>RPC3</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>220</td>
<td>120</td>
<td>490</td>
</tr>
<tr>
<td>330</td>
<td>250</td>
<td>125</td>
<td>530</td>
</tr>
<tr>
<td>320</td>
<td>235</td>
<td>140</td>
<td>495</td>
</tr>
<tr>
<td>350</td>
<td>230</td>
<td>130</td>
<td>510</td>
</tr>
<tr>
<td>340</td>
<td>240</td>
<td>125</td>
<td>515</td>
</tr>
</tbody>
</table>

**RPC2 execution time \( \geq 235 \)**

\((s_{min}, s_{max}) = (500, 600)\)
Fast fitness evaluation using bitwise operators

\[ P = \{ c_0, c_1, \ldots, c_k \} \quad \quad c_i = \langle j, e_{\text{min}}, e_{\text{max}} \rangle \]

\[ r \triangleleft c_i \iff e_{\text{min}} \leq e_j < e_{\text{max}} \]

\[ B^\text{pos}_{c_i} = B^\text{pos}_{\text{min}} \land \neg B^\text{pos}_{\text{max}} \quad \quad B^\text{neg}_{c_i} = B^\text{neg}_{\text{min}} \land \neg B^\text{neg}_{\text{max}} \]

\[ t_P = \bigwedge_{c \in P} B^\text{pos}_c \quad \quad f_P = \bigwedge_{c \in P} B^\text{neg}_c \]

<table>
<thead>
<tr>
<th>KEYS</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>\langle j, e_{\text{min}} \rangle</td>
<td>\langle B^\text{pos}<em>{\text{min}}, B^\text{neg}</em>{\text{min}} \rangle</td>
</tr>
<tr>
<td>\langle j, e_{\text{max}} \rangle</td>
<td>\langle B^\text{pos}<em>{\text{max}}, B^\text{neg}</em>{\text{max}} \rangle</td>
</tr>
<tr>
<td>...</td>
<td>....</td>
</tr>
</tbody>
</table>
Search Space Reduction

\[ \langle B_{e}^{pos}, B_{e}^{neg} \rangle \land \langle RPC, e \rangle \]

**Problem:**
Tremendous precomputing effort

**Solution:**
Consider only meaningful RPC execution times
- Mean Shift algorithm (Comaniciu and Meer, 2002) -

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, composed by 9 microservices (Spring Cloud). Zipkin is used as distributed tracing solution. Spans are stored on ElasticSearch. 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

Baselines

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


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.

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.