Perun: Keep Your Project's Performance Under Control

DevConf.cz Mini 2022

Tomas Fiedor, Jiri Pavela, Adam Rogalewicz, Tomas Vojnar

Brno University of Technology, Faculty of Information Technology

Supported by:

Red Hat, Inc.
Czech Science Foundation Project 20-07487S
JCMM PhD Talent Scholarship Programme

Motivation:
Why Care About Performance?

Performance is Important

- Software performance bugs are an omnipresent problem[1]:

[1] Source: `https://accidentallyquadratic.tumblr.com/`

Performance is Important

- Software performance bugs are an omnipresent problem[1]:
  - **Cluster computing engine** may *freeze* after an update!

- An internal check for uniqueness
  → *hanging effectively forever* for large job batch.

[1] Source: `https://accidentallyquadratic.tumblr.com/`

Performance is Important

- Software performance bugs are an omnipresent problem[1]:
  - **Cluster computing engine** may *freeze* after an update!
  - **Cloud services** may *crash*!

APACHE Spark

- An internal check for uniqueness
  → *hanging effectively forever* for large job batch.

stackoverflow

- A regular expression for stripping whitespaces
  → *34 minutes long outage*.

[1] Source: `https://accidentallyquadratic.tumblr.com/`

Performance is Important

- Software performance bugs are an omnipresent problem[1]:
  - **Cluster computing engine** may *freeze* after an update!
  - **Cloud services** may *crash*!
  - **Parsers** may experience significant *slowdown*!

- An internal check for uniqueness
  → *hanging effectively forever* for large job batch.
- A regular expression for stripping whitespaces
  → *34 minutes long outage*.
- One of *Chrome*’s parsers
  → *noticeable slowdown* for long lines.

[1] Source: `https://accidentallyquadratic.tumblr.com/`

Performance is Important

- $\mathcal{O}(n^2)$ space implementation of C# constant folding → *compiler runs out of memory*.

Performance is Important

- $\mathcal{O}(n^2)$ space implementation of C# constant folding
  → *compiler runs out of memory*.

- $\mathcal{O}(n^2)$ pattern matching algorithm in Elasticsearch
  → up to $\frac{1}{2}$ *CPU-time spent in* `Regex.simpleMatch`.

Performance is Important

- $\mathcal{O}(n^2)$ space implementation of C# constant folding
  → *compiler runs out of memory*.
- $\mathcal{O}(n^2)$ pattern matching algorithm in Elasticsearch
  → up to $\frac{1}{2}$ *CPU-time spent in* `Regex.simpleMatch`.
- Array used for tags lookup in Vim
  → $\mathcal{O}(n^2)$ complexity in the number of matches.

Performance is Important

- $\mathcal{O}(n^2)$ space implementation of C# constant folding
  → *compiler runs out of memory*.

- $\mathcal{O}(n^2)$ pattern matching algorithm in Elasticsearch
  → up to $\frac{1}{2}$ *CPU-time spent in* `Regex.simpleMatch`.

- Array used for tags lookup in Vim
  → $\mathcal{O}(n^2)$ complexity in the number of matches.

- Godoc source code parsing
  → $\mathcal{O}(n^2)$ loop for Go structs definitions.

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
  - Recently introduced bugs, as opposed to dormant bugs[2],
    - take on average less time to fix;
    - can be fixed by less experienced developers;
    - the fix is generally smaller.

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
  - Recently introduced bugs, as opposed to dormant bugs[2],
    - take on average **less time to fix** (5 vs. 8 days);
    - can be fixed by less experienced developers;
    - the fix is generally smaller.

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
  - Recently introduced bugs, as opposed to dormant bugs[2],
    - take on average **less time to fix** (5 vs. 8 days);
    - can be fixed by **less experienced developers**;
    - the fix is generally smaller.

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
  - Recently introduced bugs, as opposed to dormant bugs[2],
    - take on average **less time to fix** (5 vs. 8 days);
    - can be fixed by **less experienced developers**;
    - the fix is generally **smaller** (10 vs. 19 LoC).

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
    - Recently introduced bugs, as opposed to dormant bugs[2],
        - take on average **less time to fix** (5 vs. 8 days);
        - can be fixed by **less experienced developers**;
        - the fix is generally **smaller** (10 vs. 19 LoC).
- Hence new bugs should be discovered **as soon as possible**.

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
  - Recently introduced bugs, as opposed to dormant bugs[2],
    - take on average **less time to fix** (5 vs. 8 days);
    - can be fixed by **less experienced developers**;
    - the fix is generally **smaller** (10 vs. 19 LoC).

- Hence new bugs should be discovered **as soon as possible**.

⇒ Continuous Integration (CI) and Automated Testing!

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
  - Recently introduced bugs, as opposed to dormant bugs[2],
    - take on average **less time to fix** (5 vs. 8 days);
    - can be fixed by **less experienced developers**;
    - the fix is generally **smaller** (10 vs. 19 LoC).

- Hence new bugs should be discovered **as soon as possible**.

⇒ Continuous Integration (CI) and Automated Testing!

  - Already commonly utilized for testing the project's functionality.

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
    - Recently introduced bugs, as opposed to dormant bugs[2],
        - take on average **less time to fix** (5 vs. 8 days);
        - can be fixed by **less experienced developers**;
        - the fix is generally **smaller** (10 vs. 19 LoC).
- Hence new bugs should be discovered **as soon as possible**.

⇒ Continuous Integration (CI) and Automated Testing!

- Already commonly utilized for testing the project’s functionality.

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
    - Recently introduced bugs, as opposed to dormant bugs[2],
        - take on average **less time to fix** (5 vs. 8 days);
        - can be fixed by **less experienced developers**;
        - the fix is generally **smaller** (10 vs. 19 LoC).
- Hence new bugs should be discovered **as soon as possible**.

⇒ Continuous Integration (CI) and Automated Testing!

- Already commonly utilized for testing the project’s functionality.

- However, what about performance bugs?
    - Most solutions are either ad-hoc or proprietary.

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

Recency is Important

- **Recency is important as well:** it pays off to discover bugs quickly.
  - Recently introduced bugs, as opposed to dormant bugs[2],
    - take on average **less time to fix** (5 vs. 8 days);
    - can be fixed by **less experienced developers**;
    - the fix is generally **smaller** (10 vs. 19 LoC).
- Hence new bugs should be discovered **as soon as possible**.

⇒ Continuous Integration (CI) and Automated Testing!

  - Already commonly utilized for testing the project's functionality.

- However, what about performance bugs?
  - Most solutions are either ad-hoc or proprietary.
  - We are not aware of any **complex open-source** solutions.

[2] T.-H. Chen et al.: *An empirical study of dormant bugs*

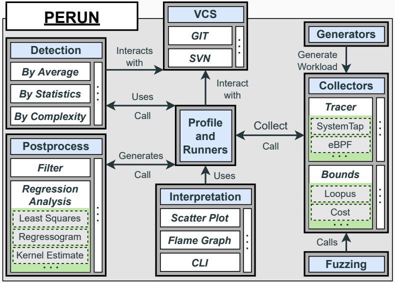
Meet Perun: Performance Version System

Perun[3] = Complex Solution for Performance Analysis and Testing

[3] Available at: `https://github.com/tfiedor/perun/`

Perun[3] = Complex Solution for Performance Analysis and Testing =

= **Collects** performance data



[3] Available at: `https://github.com/tfiedor/perun/`

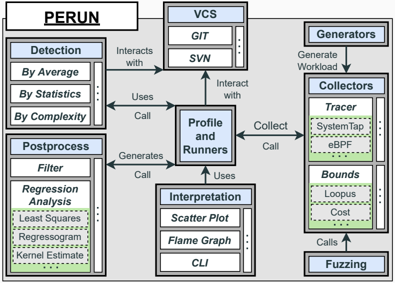
Perun: Beyond Mere Profiling

Perun[3] = Complex Solution for Performance Analysis and Testing =

= **Collects** performance data

+ **Creates** performance models

- Constant c, linear $an + b$, . . .



[3] Available at: `https://github.com/tfiedor/perun/`

Perun[3] = Complex Solution for Performance Analysis and Testing =

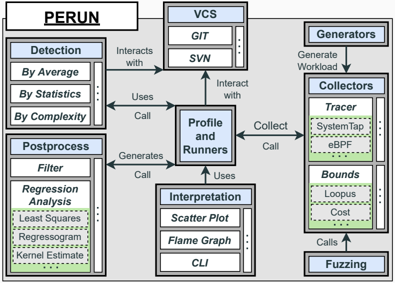
= **Collects** performance data

+ **Creates** performance models

- Constant c, linear $an + b$, ...

+ **Integrates** VCS

- Access to project history.



[3] Available at: `https://github.com/tfiedor/perun/`

Perun: Beyond Mere Profiling

Perun[3] = Complex Solution for Performance Analysis and Testing =

= **Collects** performance data

+ **Creates** performance models

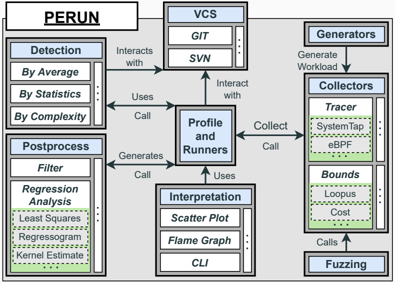
- Constant c, linear $an + b$, ...

+ **Integrates** VCS

- Access to project history.

+ **Detects** performance changes

- Degradations, optimizations.



[3] Available at: `https://github.com/tfiedor/perun/`

Perun[3] = Complex Solution for Performance Analysis and Testing =

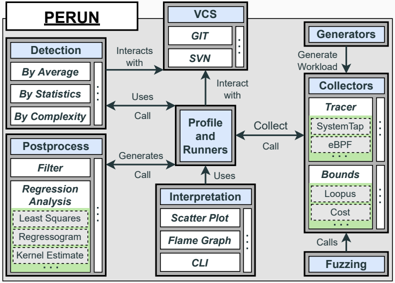
= **Collects** performance data

+ **Creates** performance models
  - Constant c, linear $an + b$, ...

+ **Integrates** VCS
  - Access to project history.

+ **Detects** performance changes
  - Degradations, optimizations.

+ **Visualizes** performance



[3] Available at: `https://github.com/tfiedor/perun/`

Perun: Beyond Mere Profiling

Perun[3] = Complex Solution for Performance Analysis and Testing =

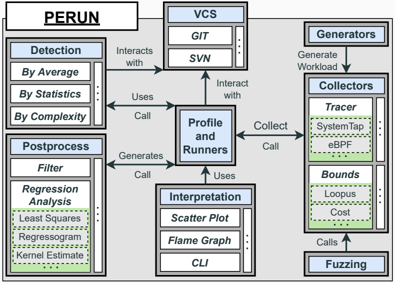
= **Collects** performance data*

+ **Creates** performance models
  - Constant c, linear $an + b$, ...

+ **Integrates** VCS
  - Access to project history.

+ **Detects** performance changes
  - Degradations, optimizations.

+ **Visualizes** performance



PERUN

Detection
By Average
By Statistics
By Complexity

Postprocess
Filter
Regression Analysis
Least Squares
Regressogram
Kernel Estimate

VCS
GIT
SVN

Interacts with
Uses
Call
Generates
Call
Interact with

Profile and Runners

Uses

Interpretation
Scatter Plot
Flame Graph
CLI

Generators
Generate Workload

Collectors
Tracer
SystemTap
eBPF
Bounds
Loopus
Cost

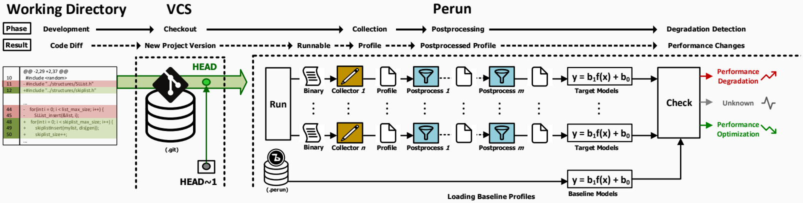
Collect
Call
Calls

Fuzzing

* Often the only thing done by traditional profilers.

[3] Available at: `https://github.com/tfiedor/perun/`

Perun Workflow: Overview

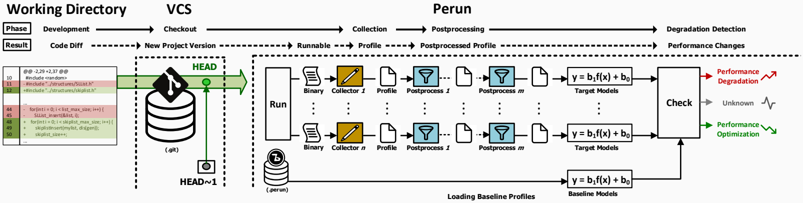


```
@@ -2,29 +2,37 @@
 #include <random>
- #include "../structures/SLList.h"
+#include "../structures/skiplist.h"

...
-    for(int i = 0; i < list_max_size; i++) {
-        SLList_insert(&list, i);
+    for(int i = 0; i < skiplist_max_size; i++) {
+        skiplistInsert(mylist, dis(gen));
+        skiplist_size++;
...
```

- **Four** major steps: Repository → Profiles → Models → Detection

Perun Workflow: Overview

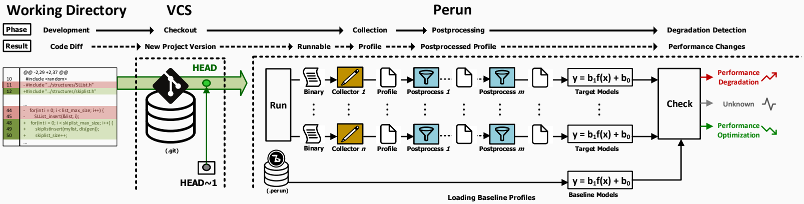


```
@@ -2,29 +2,37 @@
 #include <random>
-#include "../structures/SLList.h"
+#include "../structures/skiplist.h"

...
-    for(int i = 0; i < list_max_size; i++) {
-        SLList_insert(&list, i);
+    for(int i = 0; i < skiplist_max_size; i++) {
+        skiplistInsert(mylist, dis(gen));
+        skiplist_size++;
...
```

- **Four** major steps: Repository → Profiles → Models → Detection

Perun Workflow: Overview

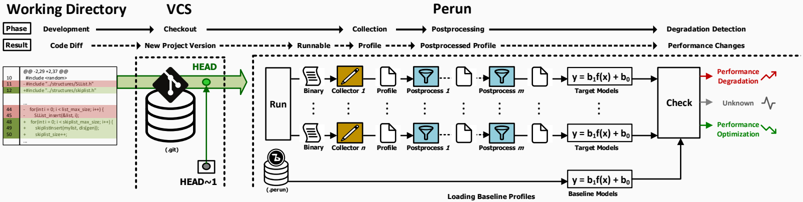


```
@@ -2,29 +2,37 @@
 #include <random>
- #include "../structures/SLList.h"
+#include "../structures/skiplist.h"

...
-    for(int i = 0; i < list_max_size; i++) {
-        SLList_insert(&list, i);
+    for(int i = 0; i < skiplist_max_size; i++) {
+        skiplistInsert(mylist, dis(gen));
+        skiplist_size++;
...
```

- **Four** major steps: Repository → Profiles → Models → Detection

Perun Workflow: Overview

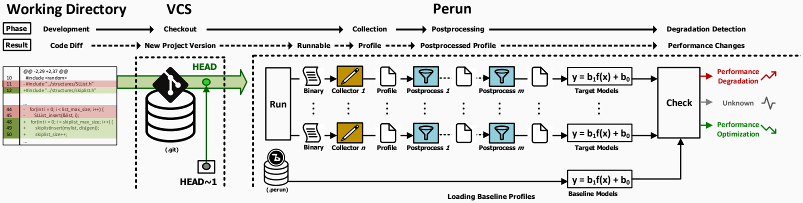


```
@@ -2,29 +2,37 @@
 #include <random>
- #include "../structures/SLList.h"
+#include "../structures/skiplist.h"

...
-    for(int i = 0; i < list_max_size; i++) {
-        SLList_insert(&list, i);
+    for(int i = 0; i < skiplist_max_size; i++) {
+        skiplistInsert(mylist, dis(gen));
+        skiplist_size++;
...
```

- **Four** major steps: Repository → Profiles → Models → Detection

Perun Workflow: Overview



```
@@ -2,29 +2,37 @@
 #include <random>
- #include "../structures/SLList.h"
+#include "../structures/skiplist.h"

...
-    for(int i = 0; i < list_max_size; i++) {
-        SLList_insert(&list, i);
+    for(int i = 0; i < skiplist_max_size; i++) {
+        skiplistInsert(mylist, dis(gen));
+        skiplist_size++;
...
```

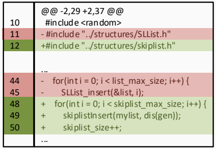
- **Four** major steps: Repository → Profiles → Models → Detection

Perun Workflow: Repository

Working Directory

Phase **Development**

Result **Code Diff**

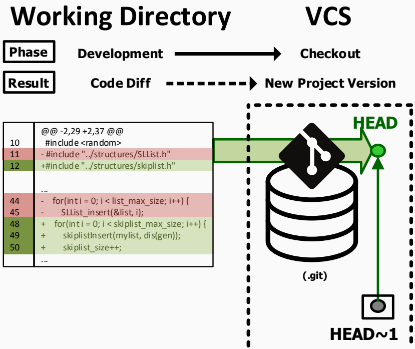


```
@@ -2,29 +2,37 @@
 #include <random>
- #include "../structures/SLList.h"
+#include "../structures/skiplist.h"

...
-   for(int i = 0; i < list_max_size; i++) {
-       SLList_insert(&list, i);
+   for(int i = 0; i < skiplist_max_size; i++) {
+       skiplistInsert(mylist, dis(gen));
+       skiplist_size++;
...
```

1. We create the project’s **working directory**.

Perun Workflow: Repository

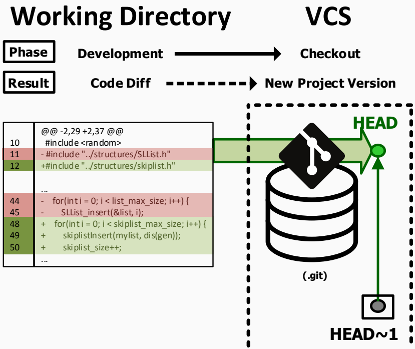


```
@@ -2,29 +2,37 @@
 #include <random>
- #include "../structures/SLList.h"
+#include "../structures/skiplist.h"

...
-   for(int i = 0; i < list_max_size; i++) {
-       SLList_insert(&list, i);
+   for(int i = 0; i < skiplist_max_size; i++) {
+       skiplistInsert(mylist, dis(gen));
+       skiplist_size++;
...
```

1. We create the project's **working directory**.
2. We initialize a **VCS** (e.g., Git) for project versioning.

Perun Workflow: Repository

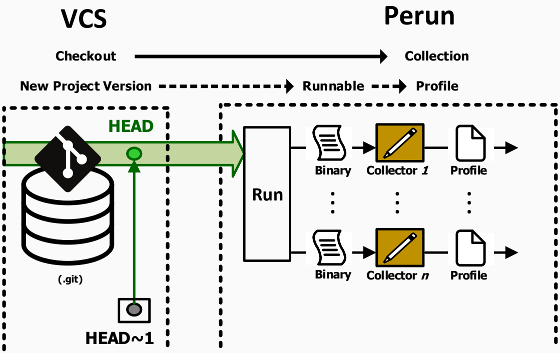


```
@@ -2,29 +2,37 @@
 #include <random>
- #include "../structures/SLList.h"
+#include "../structures/skiplist.h"

...
-   for(int i = 0; i < list_max_size; i++) {
-       SLList_insert(&list, i);
+   for(int i = 0; i < skiplist_max_size; i++) {
+       skiplistInsert(mylist, dis(gen));
+       skiplist_size++;
...
```

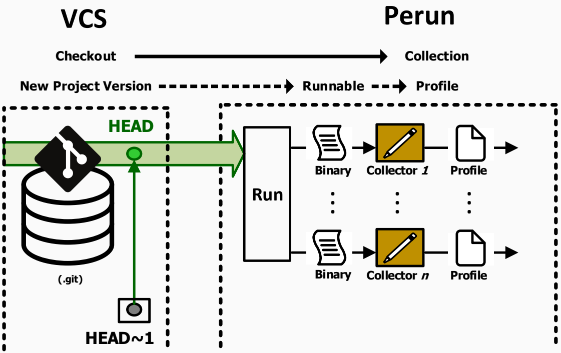
1. We create the project’s **working directory**.
2. We initialize a **VCS** (e.g., Git) for project versioning.
3. We initialize **Perun** in the repository alongside the VCS.

Perun Workflow: Profiles



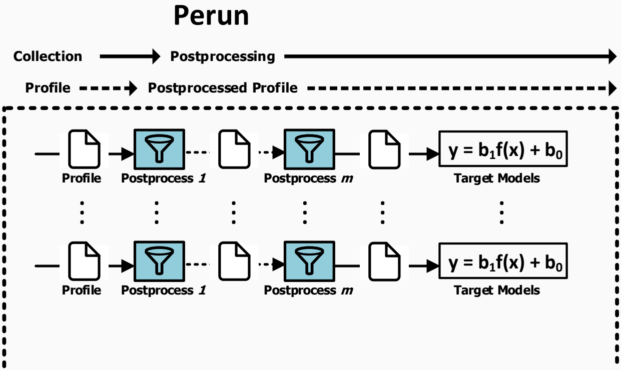
4. We measure project's performance and obtain **profiles**.
   - Profiles are stored within Perun and linked to the corresponding VCS version (e.g., commit).

Perun Workflow: Profiles



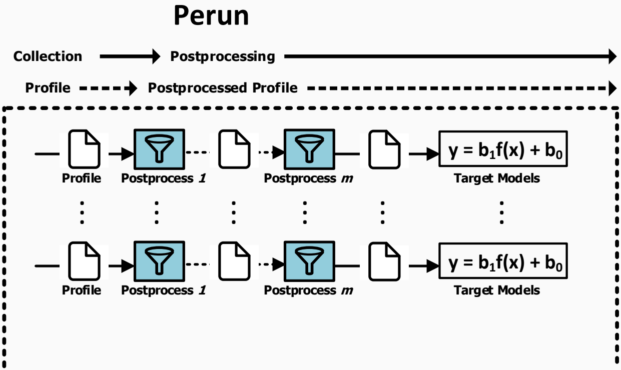
4. We measure project's performance and obtain **profiles**.
   - Profiles are stored within Perun and linked to the corresponding VCS version (e.g., commit).

Perun Workflow: Models



5. We create **performance models** from profiles using postprocessors.
    - Models are stored within Perun alongside the profiles.

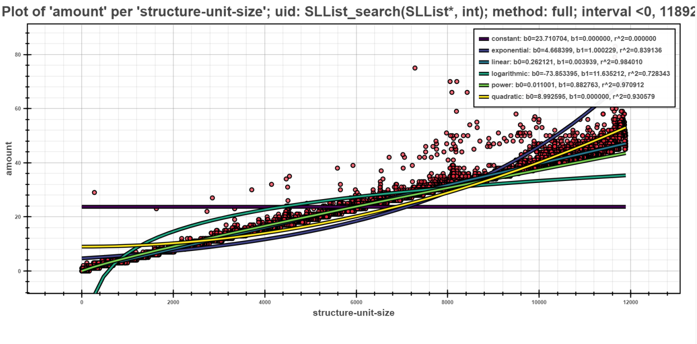
Perun Workflow: Models



5. We create **performance models** from profiles using postprocessors.
    - Models are stored within Perun alongside the profiles.

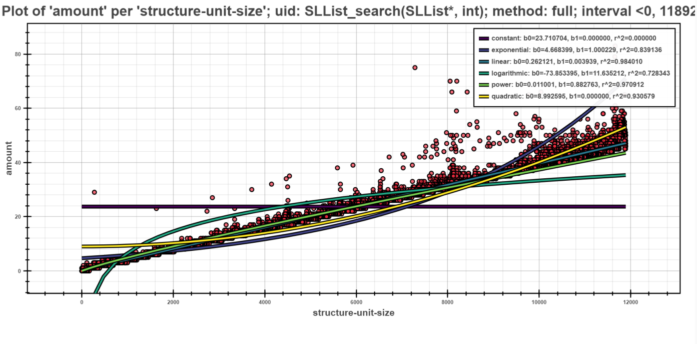
Perun Workflow: Models Example

- Models in Perun are **mathematical functions** of the input size or **statistical summaries** describing the main features of the profile.



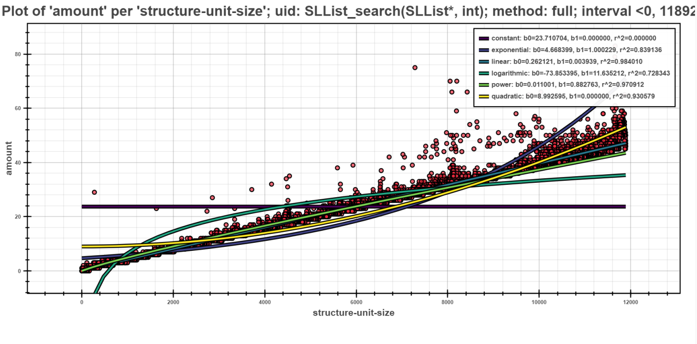
Perun Workflow: Models Example

- Models in Perun are **mathematical functions** of the input size or **statistical summaries** describing the main features of the profile.
- For example, in regression analysis:



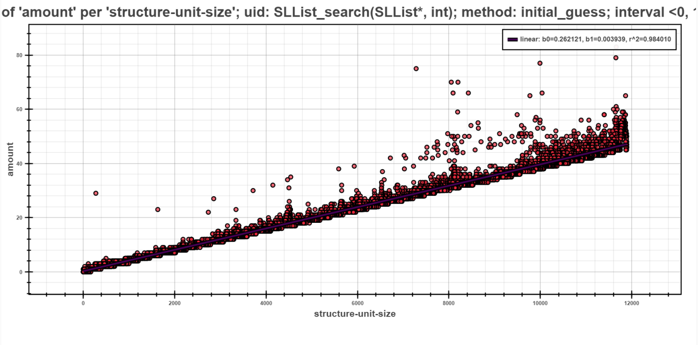
Perun Workflow: Models Example

- Models in Perun are **mathematical functions** of the input size or **statistical summaries** describing the main features of the profile.
- For example, in regression analysis:
  - the curve is described using **function** $y = b_1 f(x) + b_0$,

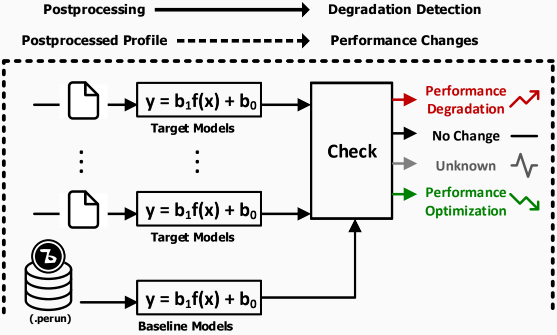


Perun Workflow: Models Example

- Models in Perun are **mathematical functions** of the input size or **statistical summaries** describing the main features of the profile.
- For example, in regression analysis:
  - the curve is described using **function** $y = b_1 f(x) + b_0$,
  - and the model quality as **coefficient of determination** R^2.

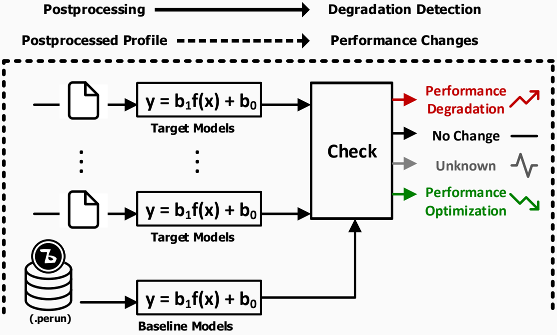


Perun Workflow: Detection



6. We detect **performance changes** using models or directly profiles.
    - Target refers to the current version.
    - Baseline refers to the previous version used for comparison.

Perun Workflow: Detection



6. We detect **performance changes** using models or directly profiles.
   - Target refers to the current version.
   - Baseline refers to the previous version used for comparison.

Perun Workflow: Detection Example

- Multiple **detection algorithms** are implemented in Perun:
    - *Best Model Order Equality*
    - *Integral Comparison*
    - . . .
    - *Exclusive-Time Outliers*

Perun Workflow: Detection Example

- Multiple **detection algorithms** are implemented in Perun:
  - *Best Model Order Equality*
  - *Integral Comparison*
  - ...
  - *Exclusive-Time Outliers*

Perun Workflow: Detection Example

- Multiple **detection algorithms** are implemented in Perun:
  - *Best Model Order Equality*
  - *Integral Comparison*
  - . . .
  - *Exclusive-Time Outliers*
- **Exclusive-Time Outliers:**
  - Based on per-function comparison of exclusive-time **deltas** Δ.
    - *Exclusive-time (self)*: time spent exclusively in a function.

Perun Workflow: Detection Example

- Multiple **detection algorithms** are implemented in Perun:
  - *Best Model Order Equality*
  - *Integral Comparison*
  - . . .
  - *Exclusive-Time Outliers*

- **Exclusive-Time Outliers:**
  - Based on per-function comparison of exclusive-time **deltas** Δ.
    - *Exclusive-time (self)*: time spent exclusively in a function.
  - A hierarchy of outlier detection methods determines the **severity**:
    - Modified Z-score $y_i = \frac{x_i - \tilde{X}}{k \cdot median(|x_i - \tilde{X}|)}$

Perun Workflow: Detection Example

- Multiple **detection algorithms** are implemented in Perun:
  - *Best Model Order Equality*
  - *Integral Comparison*
  - ...
  - *Exclusive-Time Outliers*
- **Exclusive-Time Outliers:**
  - Based on per-function comparison of exclusive-time **deltas** Δ.
    - *Exclusive-time (self)*: time spent exclusively in a function.
  - A hierarchy of outlier detection methods determines the **severity**:
    - Modified Z-score $y_i = \frac{x_i - \tilde{X}}{k \cdot median(|x_i - \tilde{X}|)}$
    - IQR multiple $Q_1 - k \cdot IQR < x < Q_3 + k \cdot IQR$

Perun Workflow: Detection Example

- Multiple **detection algorithms** are implemented in Perun:
  - *Best Model Order Equality*
  - *Integral Comparison*
  - . . .
  - *Exclusive-Time Outliers*

- **Exclusive-Time Outliers:**
  - Based on per-function comparison of exclusive-time **deltas** Δ.
    - *Exclusive-time (self)*: time spent exclusively in a function.
  - A hierarchy of outlier detection methods determines the **severity**:
    - Modified Z-score $y_i = \frac{x_i - \tilde{X}}{k \cdot median(|x_i - \tilde{X}|)}$
    - IQR multiple $Q_1 - k \cdot IQR < x < Q_3 + k \cdot IQR$
    - Standard deviation multiple $\overline{X} - k \cdot \sigma < x < \overline{X} - k \cdot \sigma$

Demonstration of Perun #1: Finding Performance Changes

CPython: Issue #92356

- **CPython**: Reference C implementation of a Python interpreter.

[4] Reported by user `mdboom`: `https://github.com/python/cpython/issues/92356`

CPython: Issue #92356

- **CPython**: Reference C implementation of a Python interpreter.
- **Issue #92356**[4]: A performance regression in `ctypes` module.
  - ≈ 8% higher function call overhead (py3.11.0a7 vs. py3.10.4).

[4] Reported by user `mdboom`: `https://github.com/python/cpython/issues/92356`

CPython: Issue #92356

- **CPython**: Reference C implementation of a Python interpreter.
- **Issue #92356**[4]: A performance regression in `ctypes` module.
  - $\approx$ 8% higher function call overhead (py3.11.0a7 vs. py3.10.4).
  - Replicated using the `pyperformance ctypes` benchmark.

[4] Reported by user `mdboom`: `https://github.com/python/cpython/issues/92356`

CPython: Issue #92356

- **CPython**: Reference C implementation of a Python interpreter.
- **Issue #92356**[4]: A performance regression in `ctypes` module.
  - ≈ 8% higher function call overhead (py3.11.0a7 vs. py3.10.4).
  - Replicated using the `pyperformance ctypes` benchmark.
  - **Fixed soon after the report.**

[4] Reported by user `mdboom`: `https://github.com/python/cpython/issues/92356`

- **CPython**: Reference C implementation of a Python interpreter.
- **Issue #92356**[4]: A performance regression in `ctypes` module.
  - ≈ 8% higher function call overhead (py3.11.0a7 vs. py3.10.4).
  - Replicated using the `pyperformance ctypes` benchmark.
  - **Fixed soon after the report.**

So problem solved. Why use Perun?

[4] Reported by user `mdboom`: `https://github.com/python/cpython/issues/92356`

- **CPython**: Reference C implementation of a Python interpreter.
- **Issue #92356**[4]: A performance regression in `ctypes` module.
  - $\approx$ 8% higher function call overhead (py3.11.0a7 vs. py3.10.4).
  - Replicated using the `pyperformance` `ctypes` benchmark.
  - **Fixed soon after the report.**

So problem solved. Why use Perun?

- Discovering the issue and finding the root cause is the hard part.

[4] Reported by user `mdboom`: `https://github.com/python/cpython/issues/92356`

- **CPython**: Reference C implementation of a Python interpreter.
- **Issue #92356**[4]: A performance regression in `ctypes` module.
  - ≈ 8% higher function call overhead (py3.11.0a7 vs. py3.10.4).
  - Replicated using the `pyperformance ctypes` benchmark.
  - **Fixed soon after the report.**

So problem solved. Why use Perun?

- Discovering the issue and finding the root cause is the hard part.
  - Often requires **significant** manual effort by the developers.

[4] Reported by user `mdboom`: `https://github.com/python/cpython/issues/92356`

CPython: Issue #92356

- **CPython**: Reference C implementation of a Python interpreter.
- **Issue #92356**[4]: A performance regression in `ctypes` module.
  - ≈ 8% higher function call overhead (py3.11.0a7 vs. py3.10.4).
  - Replicated using the `pyperformance` `ctypes` benchmark.
  - **Fixed soon after the report.**

So problem solved. Why use Perun?

- Discovering the issue and finding the root cause is the hard part.
  - Often requires **significant** manual effort by the developers.
- **Perun** reduces this effort and helps the developers.
  - Perun utilizes the **recency** principle and results from past profiling.

[4] Reported by user `mdboom`: `https://github.com/python/cpython/issues/92356`

CPython: Setup and Baseline

Using **Perun**, we could handle the **issue #92356** as follows:

CPython: Setup and Baseline

Using **Perun**, we could handle the **issue #92356** as follows:

1. We **initialize** a CPython repository with Perun.

Perun commands

```
perun init
```

CPython: Setup and Baseline

Using **Perun**, we could handle the **issue #92356** as follows:

1. We **initialize** a CPython repository with Perun.
2. We **store** a profile for CPython **v3.10.4** ctypes benchmark in Perun.

Perun commands

```
perun init
```

CPython: Setup and Baseline

Using **Perun**, we could handle the **issue #92356** as follows:

1. We **initialize** a CPython repository with Perun.
2. We **store** a profile for CPython **v3.10.4** ctypes benchmark in Perun.
    - We denote this profile as baseline.
    - Perun handles the *profile-commit* link internally.

Perun commands

```
perun init
```

CPython: Detecting Changes

3. CPython v3.11.0a7 **rolls out**.

CPython: Detecting Changes

3. CPython v3.11.0a7 **rolls out**.
4. We **profile** the ctypes benchmark for CPython **v3.11.0a7**.

Perun commands

```
perun collect -c <py3.11.0a7> -a <benchmark> trace -b <files>
```

CPython: Detecting Changes

3. CPython v3.11.0a7 **rolls out**.
4. We **profile** the ctypes benchmark for CPython **v3.11.0a7**.
   - We denote the resulting profile as target.

Perun commands

```
perun collect -c <py3.11.0a7> -a <benchmark> trace -b <files>
perun add <target>
```

CPython: Detecting Changes

3. CPython v3.11.0a7 **rolls out**.
4. We **profile** the ctypes benchmark for CPython **v3.11.0a7**.
    - We denote the resulting profile as target.
5. We **compare** the baseline and target profiles.

Perun commands

```
perun collect -c <py3.11.0a7> -a <benchmark> trace -b <files>
perun add <target>
perun check -f profiles <baseline> <target>
```

CPython: Detecting Changes

3. CPython v3.11.0a7 **rolls out**.
4. We **profile** the ctypes benchmark for CPython **v3.11.0a7**.
    - We denote the resulting profile as target.
5. We **compare** the baseline and target profiles.
    - Perun supports multiple comparison algorithms.
    - For this particular issue, we used *Exclusive-Time Outliers*.

Perun commands

```
perun collect -c <py3.11.0a7> -a <benchmark> trace -b <files>
perun add <target>
perun check -f profiles <baseline> <target>
```

CPython: Regression Detected

Location	Result	TΔ [ms]	TΔ [%]
_ctypes_init_fielddesc	NotInBaseline	77.95	5.23
_ctypes_get_fielddesc	SevereDegradation	52.9	3.55
_ctypes_callproc	Degradation	2.84	0.19
...			
_ctypes.cpython-311	TotalDegradation	136.92	9.19

⋆ TΔ: exclusive-time delta of *target* − *baseline*.

CPython: Regression Detected

Location	Result	TΔ [ms]	TΔ [%]
_ctypes_init_fielddesc	NotInBaseline	77.95	**5.23**
_ctypes_get_fielddesc	SevereDegradation	52.9	**3.55**
_ctypes_callproc	Degradation	2.84	0.19
	...		
_ctypes.cpython-311	TotalDegradation	136.92	9.19

⋆ TΔ: exclusive-time delta of *target* − *baseline*.

CPython: Regression Detected

Location	Result	TΔ [ms]	TΔ [%]
_ctypes_init_fielddesc	NotInBaseline	77.95	**5.23**
_ctypes_get_fielddesc	SevereDegradation	52.9	**3.55**
_ctypes_callproc	Degradation	2.84	0.19
	...		
_ctypes.cpython-311	TotalDegradation	136.92	9.19

* TΔ: exclusive-time delta of *target* − *baseline*.

- **Root cause** of the issue: repeated calls of an init function.

Function `_ctypes_get_fielddesc`

```
if (!initialized) {
    _ctypes_init_fielddesc();
}
```

CPython: Hotfix

6. We **create** a new hotfix branch and **fix** the issue.

Fixing `_ctypes_get_fielddesc`

```
  if (!initialized) {
+     initialized = 1;
      _ctypes_init_fielddesc();
  }
```

CPython: Hotfix

6. We **create** a new hotfix branch and **fix** the issue.
7. We **Profile** the CPython hotfixed version.

Fixing _ctypes_get_fielddesc

```
  if (!initialized) {
+     initialized = 1;
      _ctypes_init_fielddesc();
  }
```

Perun commands

```
perun collect -c <py3.11.0a7-fix> -a <benchmark> trace <...>
```

CPython: Hotfix

6. We **create** a new hotfix branch and **fix** the issue.
7. We **Profile** the CPython hotfixed version.
    - We denote the resulting profile as hotfix.

Fixing _ctypes_get_fielddesc

```
  if (!initialized) {
+     initialized = 1;
      _ctypes_init_fielddesc();
  }
```

Perun commands

```
perun collect -c <py3.11.0a7-fix> -a <benchmark> trace <...>
perun add <hotfix>
```

CPython: Hotfix

6. We **create** a new hotfix branch and **fix** the issue.
7. We **Profile** the CPython hotfixed version.
    - We denote the resulting profile as hotfix.
8. We **compare** the baseline and hotfix profiles.

Fixing `_ctypes_get_fielddesc`

```
  if (!initialized) {
+      initialized = 1;
       _ctypes_init_fielddesc();
  }
```

Perun commands

```
perun collect -c <py3.11.0a7-fix> -a <benchmark> trace <...>
perun add <hotfix>
perun check -f profiles <baseline> <hotfix>
```

CPython: Issue Fixed!

Location	**Result**	**Δ [ms]**	**Δ_{old} [ms]**	**Δ [%]**	**Δ_{old} [%]**
	...				
_ctypes_get_fielddesc	MaybeDegradation	0.89	52.9	0.06	3.55
_ctypes_init_fielddesc	NotInBaseline	0.02	77.95	0.00	5.23
_ctypes.cpython-311	TotalDegradation	23.45	136.92	1.70	9.19

$\star$ Δ: exclusive-time delta of *hotfix*−*baseline*.

$\star$ Δ_{old}: exclusive-time delta of *target*−*baseline*.

CPython: Issue Fixed!

Location	Result	Δ [ms]	Δ_{old} [ms]	Δ [%]	Δ_{old} [%]
		. . .			
_ctypes_get_fielddesc	MaybeDegradation	0.89	52.9	**0.06**	3.55
_ctypes_init_fielddesc	NotInBaseline	0.02	77.95	0.00	5.23
_ctypes.cpython-311	TotalDegradation	23.45	136.92	1.70	9.19

⋆ Δ: exclusive-time delta of *hotfix*−*baseline*.

⋆ Δ_{old}: exclusive-time delta of *target*−*baseline*.

- The **_ctypes_get_fielddesc** Δ has improved significantly.

CPython: Issue Fixed!

Location	Result	Δ [ms]	Δ_{old} [ms]	Δ [%]	Δ_{old} [%]
		. . .			
_ctypes_get_fielddesc	MaybeDegradation	0.89	52.9	**0.06**	3.55
_ctypes_init_fielddesc	NotInBaseline	0.02	77.95	**0.00**	5.23
_ctypes.cpython-311	TotalDegradation	23.45	136.92	1.70	9.19

⋆ Δ: exclusive-time delta of *hotfix*−*baseline*.

⋆ Δ_{old}: exclusive-time delta of *target*−*baseline*.

- The **_ctypes_get_fielddesc** Δ has improved significantly.
- The **_ctypes_init_fielddesc** Δ is now negligible.

CPython: Issue Fixed!

Location	Result	Δ [ms]	Δ_{old} [ms]	Δ [%]	Δ_{old} [%]
	...				
_ctypes_get_fielddesc	MaybeDegradation	0.89	52.9	**0.06**	3.55
_ctypes_init_fielddesc	NotInBaseline	0.02	77.95	**0.00**	5.23
_ctypes.cpython-311	TotalDegradation	23.45	136.92	1.70	9.19

⋆ Δ: exclusive-time delta of *hotfix*−*baseline*.

⋆ Δ_{old}: exclusive-time delta of *target*−*baseline*.

- The **_ctypes_get_fielddesc** Δ has improved significantly.
- The **_ctypes_init_fielddesc** Δ is now negligible.

⇒ Perun leverages **VCS** and **Recency** to successfully discover and help locate performance issues in new project versions.

Demonstration of Perun #2: Generating Workloads

Stack Overflow: Issue

- Recall the **Stack Overflow** issue:

stackoverflow

- A regular expression for stripping whitespaces → *34 minutes long outage*.

Stack Overflow: Issue

- Recall the **Stack Overflow** issue:

- A regular expression for stripping whitespaces → *34 minutes long outage*.

The offending regular expression:

```
^[\s\u200c]+|[\s\u200c]+$
```

Stack Overflow: Issue

- Recall the **Stack Overflow** issue:

stackoverflow

- A regular expression for stripping whitespaces → *34 minutes long outage*.

The offending regular expression:

```
^[\s\u200c]+|[\s\u200c]+$
```

- This Regex can lead to **extensive backtracking**.
- Can **Perun** help us with detecting this potential performance issue?

Stack Overflow: Issue

- Recall the **Stack Overflow** issue:
  - A regular expression for stripping whitespaces → *34 minutes long outage*.

The offending regular expression:

```
^[\s\u200c]+|[\s\u200c]+$
```

- This Regex can lead to **extensive backtracking**.
- Can **Perun** help us with detecting this potential performance issue?

Perun's Performance Fuzzing![5]

[5] Builds on a principle originally proposed by C. Lemieux et al.: *PerfFuzz: automatically generating pathological inputs.*

Stack Overflow: Introducing Fuzzing

- **Fuzzing (Fuzz testing)** is a form of fault injection stress testing.

Stack Overflow: Introducing Fuzzing

- **Fuzzing (Fuzz testing)** is a form of fault injection stress testing.
  - Malformed inputs are generated and fed to the program.
  - The program response is then monitored for crashes or errors.

- **Fuzzing (Fuzz testing)** is a form of fault injection stress testing.
    - Malformed inputs are generated and fed to the program.
    - The program response is then monitored for crashes or errors.
- **Perun's Fuzzer**[6] is profiling-guided.

[6] M. Liscinsky: *Fuzz Testing of Program Performance*.

Stack Overflow: Introducing Fuzzing

- **Fuzzing (Fuzz testing)** is a form of fault injection stress testing.
    - Malformed inputs are generated and fed to the program.
    - The program response is then monitored for crashes or errors.
- **Perun’s Fuzzer**[6] is profiling-guided.
    - The program is not monitored for errors or crashes, but **slowdown**.
    - We evaluate the results using profiling.

[6] M. Liscinsky: *Fuzz Testing of Program Performance*.

- **Fuzzing (Fuzz testing)** is a form of fault injection stress testing.
  - Malformed inputs are generated and fed to the program.
  - The program response is then monitored for crashes or errors.
- **Perun's Fuzzer**[6] is profiling-guided.
  - The program is not monitored for errors or crashes, but **slowdown**.
  - We evaluate the results using profiling.
  - **The goal** is to find inputs that cause severe slowdown[7].

[6] M. Liscinsky: *Fuzz Testing of Program Performance*.

[7] In some of our experiments, Perun's Fuzzer achieved a **slowdown of several hours**!

Stack Overflow: Fuzzing Setup

- For this experiment, we used the following **settings**:

Stack Overflow: Fuzzing Setup

- For this experiment, we used the following **settings**:
    - **Profiled program:** 25 LoC C++ program matching the offending regex (using the `regex_search` function) in the input file.

Stack Overflow: Fuzzing Setup

- For this experiment, we used the following **settings**:
  - **Profiled program:** 25 LoC C++ program matching the offending regex (using the `regex_search` function) in the input file.
  - **Seed:** 150 LoC `C` implementation of a parallel grep.

Stack Overflow: Fuzzing Setup

- For this experiment, we used the following **settings**:
  - **Profiled program:** 25 LoC C++ program matching the offending regex (using the `regex_search` function) in the input file.
  - **Seed:** 150 LoC `C` implementation of a parallel grep.
  - **Perun's own mutation rules.** Particularly useful rule is:
    - `T.10:` Insert whitespaces to a random position in a string.

Stack Overflow: Fuzzing Setup

- For this experiment, we used the following **settings**:
  - **Profiled program:** 25 LoC C++ program matching the offending regex (using the `regex_search` function) in the input file.
  - **Seed:** 150 LoC `C` implementation of a parallel grep.
  - **Perun's own mutation rules.** Particularly useful rule is:
    - `T.10`: Insert whitespaces to a random position in a string.
  - **Size limits:** (a) 5 000 bytes, (b) 10 000 bytes.

Stack Overflow: Fuzzing Results

Input	Size [B]	Lines	Whitespaces	Duration [s]	Slowdown
seed	3535	150	306	0.096	–
worst-case$_a$	5000	5	4881	1.566	16.3x
worst-case$_b$	10000	17	9603	2.611	27.2x

⋆ We let the fuzzing run for several hours to obtain the shown workloads.

⋆ The generated workloads can then be used for future performance testing.

Stack Overflow: Fuzzing Results

Input	Size [B]	Lines	Whitespaces	Duration [s]	Slowdown
seed	3535	150	306	0.096	–
worst-case$_a$	5000	5	4881	**1.566**	**16.3x**
worst-case$_b$	10000	17	9603	2.611	27.2x

⋆ We let the fuzzing run for several hours to obtain the shown workloads.

⋆ The generated workloads can then be used for future performance testing.

Stack Overflow: Fuzzing Results

Input	Size [B]	Lines	Whitespaces	Duration [s]	Slowdown
seed	3535	150	306	0.096	–
worst-case$_a$	5000	5	4881	**1.566**	**16.3x**
worst-case$_b$	10000	17	9603	**2.611**	**27.2x**

$\star$ We let the fuzzing run for several hours to obtain the shown workloads.

$\star$ The generated workloads can then be used for future performance testing.

Stack Overflow: Fuzzing Results

Input	Size [B]	Lines	Whitespaces	Duration [s]	Slowdown
seed	3535	150	306	0.096	–
worst-case$_a$	5000	5	4881	**1.566**	**16.3x**
worst-case$_b$	10000	17	9603	**2.611**	**27.2x**

⋆ We let the fuzzing run for several hours to obtain the shown workloads.

⋆ The generated workloads can then be used for future performance testing.

⇒ Perun Fuzzer can force potential performance issues to manifest!

Stack Overflow: Fuzzing Results

Input	Size [B]	Lines	Whitespaces	Duration [s]	Slowdown
seed	3535	150	306	0.096	–
worst-case$_a$	5000	5	4881	**1.566**	**16.3x**
worst-case$_b$	10000	17	9603	**2.611**	**27.2x**

⋆ We let the fuzzing run for several hours to obtain the shown workloads.

⋆ The generated workloads can then be used for future performance testing.

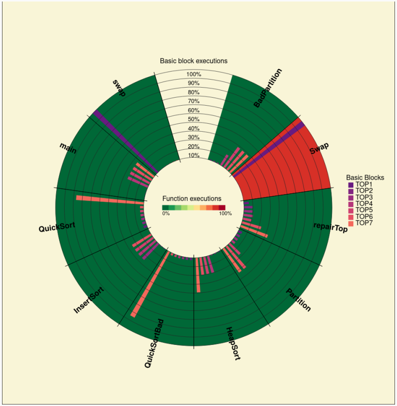
⇒ Perun Fuzzer can force potential performance issues to manifest!

- We employ different **mutation strategies** based on the input type.
  - Text files, binary files, domain-specific (e.g., XML), . . .

Ongoing and Future Work

Ongoing Work: Granularity

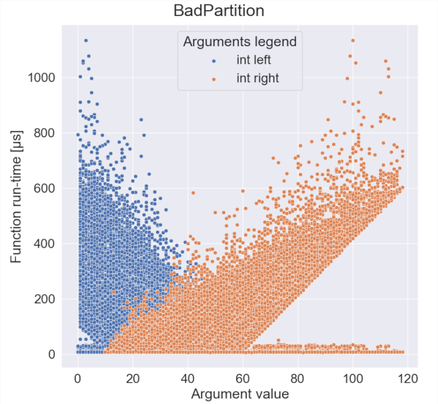
- We focus on increasing profiling **granularity**.
    - Measuring time spent per function basic block[8].



[8] P. Mocary: *Performance Analysis of Programs Based on PIN Framework*.

Ongoing Work: Precision

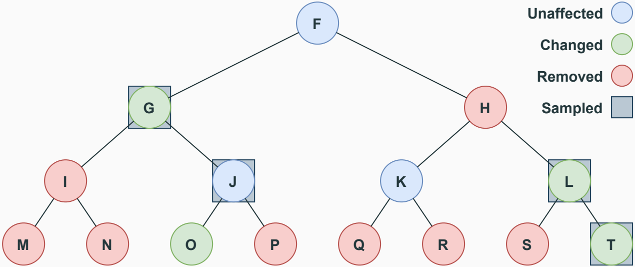
- Moreover, we focus on increasing profiling **precision**.
  - Measuring time spent w.r.t. function parameter values[9].



[9] P. Mocary: *Performance Analysis of Programs Based on PIN Framework*.

Ongoing Work: Efficiency

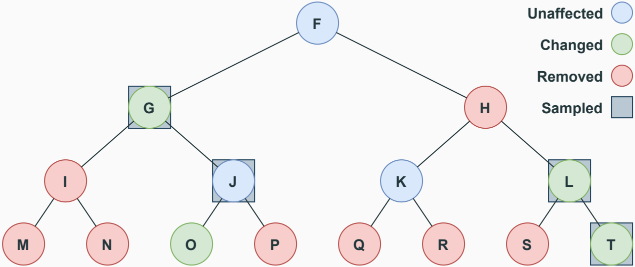
- Finally, we focus on increasing profiling **efficiency**.
  - Utilizing a collection of heuristics to speed up the profiling[10].
  - We propose to select and sample particularly **important** functions.



[10] J. Pavela: *Efficient Techniques for Program Performance Analysis*.

Ongoing Work: Efficiency

- Finally, we focus on increasing profiling **efficiency**.
  - Utilizing a collection of heuristics to speed up the profiling[10].
  - We propose to select and sample particularly **important** functions.

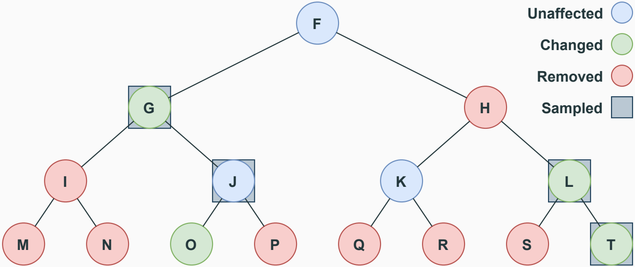


- The main **challenge**: achieving sufficient precision.

[10] J. Pavela: *Efficient Techniques for Program Performance Analysis*.

Ongoing Work: Efficiency

- Finally, we focus on increasing profiling **efficiency**.
  - Utilizing a collection of heuristics to speed up the profiling[10].
  - We propose to select and sample particularly **important** functions.



- The main **challenge**: achieving sufficient precision.

  ⇒ Fully profiling all the important functions.

[10] J. Pavela: *Efficient Techniques for Program Performance Analysis*.

Future Work

- **Future work:**

Future Work

- **Future work:**
  - Support for **more languages**:
    - Currently mainly C/C++.

Future Work

- **Future work:**
  - Support for **more languages**:
    - Currently mainly C/C++.
    - ⇒ C#, Java, Python, . . .

Future Work

- **Future work:**
  - Support for **more languages**:
    - Currently mainly C/C++.

    ⇒ C#, Java, Python, . . .

Future Work

- **Future work:**
    - Support for **more languages**:
        - Currently mainly C/C++.
        - ⇒ C#, Java, Python, . . .
    - More performance **metrics**:
        - Currently mainly time consumption of functions.
        - Memory consumption in experimental stage.

Future Work

- **Future work:**
  - Support for **more languages**:
    - Currently mainly C/C++.

    ⇒ C#, Java, Python, ...
  - More performance **metrics**:
    - Currently mainly time consumption of functions.
    - Memory consumption in experimental stage.

    ⇒ Energy consumption, cache hit/miss, scheduler, ...

Future Work

- **Future work:**
  - Support for **more languages**:
    - Currently mainly C/C++.

    ⇒ C#, Java, Python, . . .
  - More performance **metrics**:
    - Currently mainly time consumption of functions.
    - Memory consumption in experimental stage.

    ⇒ Energy consumption, cache hit/miss, scheduler, . . .

Future Work

- **Future work:**
  - Support for **more languages**:
    - Currently mainly C/C++.
    - ⇒ C#, Java, Python, . . .
  - More performance **metrics**:
    - Currently mainly time consumption of functions.
    - Memory consumption in experimental stage.
    - ⇒ Energy consumption, cache hit/miss, scheduler, . . .
  - Supporting well-established and/or state-of-the-art **tools**:
    - Currently Facebook Infer Cost, Loopus.

Future Work

- **Future work:**
    - Support for **more languages**:
        - Currently mainly C/C++.

        ⇒ C#, Java, Python, . . .
    - More performance **metrics**:
        - Currently mainly time consumption of functions.
        - Memory consumption in experimental stage.

        ⇒ Energy consumption, cache hit/miss, scheduler, . . .
    - Supporting well-established and/or state-of-the-art **tools**:
        - Currently Facebook Infer Cost, Loopus.

        ⇒ Valgrind tool suite, Perf, Gprof, . . .

Future Work

- **Future work:**
  - Support for **more languages**:
    - Currently mainly C/C++.

    ⇒ C#, Java, Python, . . .
  - More performance **metrics**:
    - Currently mainly time consumption of functions.
    - Memory consumption in experimental stage.

    ⇒ Energy consumption, cache hit/miss, scheduler, . . .
  - Supporting well-established and/or state-of-the-art **tools**:
    - Currently Facebook Infer Cost, Loopus.

    ⇒ Valgrind tool suite, Perf, Gprof, . . .
  - Performance analysis of **Dynamic Data Structures**

Conclusion

Conclusion

- **Perun** = Complex Performance Analysis and Testing Solution.

Conclusion

- **Perun** = Complex Performance Analysis and Testing Solution.
    - **Integrates** VCS.
    - **Collects** performance data.
    - **Derives** performance models.
    - **Detects** performance changes.
    - **Visualises** performance.

Conclusion

- **Perun** = Complex Performance Analysis and Testing Solution.
  - **Integrates** VCS.
  - **Collects** performance data.
  - **Derives** performance models.
  - **Detects** performance changes.
  - **Visualises** performance.

  ⇒ Not just mere profiling!

Conclusion

- **Perun** = Complex Performance Analysis and Testing Solution.
  - **Integrates** VCS.
  - **Collects** performance data.
  - **Derives** performance models.
  - **Detects** performance changes.
  - **Visualises** performance.

  ⇒ Not just mere profiling!

- **Nicely complements cheap benchmarking.**
  - Shown on a real-world `CPython` performance bug.
- **Forces potential performance issues to manifest.**
  - Replication of a `Stack Overflow` regex performance bug.

Conclusion

- **Perun** = Complex Performance Analysis and Testing Solution.
  - **Integrates** VCS.
  - **Collects** performance data.
  - **Derives** performance models.
  - **Detects** performance changes.
  - **Visualises** performance.

  ⇒ Not just mere profiling!

- **Nicely complements cheap benchmarking.**
  - Shown on a real-world `CPython` performance bug.
- **Forces potential performance issues to manifest.**
  - Replication of a `Stack Overflow` regex performance bug.

- **Ongoing and Future work:**
  - Improving **granularity**, **precision** and **efficiency** of profiling.
  - Support for more **languages**, performance **metrics**, existing **tools**.

Perun: Keep Your Project's Performance Under Control

DevConf.cz Mini 2022

Tomas Fiedor, Jiri Pavela, Adam Rogalewicz, Tomas Vojnar

Brno University of Technology, Faculty of Information Technology

Supported by:

Red Hat, Inc.
Czech Science Foundation Project 20-07487S
JCMM PhD Talent Scholarship Programme