Learning the Linux Kernel Configuration Space: Results and Challenges

Prof. Mathieu Acher (University of Rennes 1, Inria/IRISA)
Talk given at ELC 2019 (30 october 2019)

• Abstract: “Given a configuration, can humans know in advance the size, the compilation time, or the boot time of a Linux kernel? Owing to the huge complexity of Linux (there are more than 15000 options with hard constraints and subtle interactions), machines should rather assist contributors and integrators in mastering the configuration space of the kernel. In this talk, Mathieu Acher will introduce TuxML an OSS tool based on Docker/Python to massively gather data about thousands of kernel configurations. Mathieu will describe how 200K+ configurations have been automatically built and how machine learning can exploit this information to predict properties of unseen Linux configurations, with different use cases (identification of influential/buggy options, finding of small kernels, etc.) The vision is that a continuous understanding of the configuration space is undoubtedly beneficial for the Linux community, yet several technical challenges remain in terms of infrastructure and automation.”

• This research was funded by the ANR-17-CE25-0010-01 VaryVary project
  – https://varyvary.github.io/
Preprints (feedbacks welcome!)

- **Learning From Thousands of Build Failures of Linux Kernel Configurations**
  - Mathieu Acher, Hugo Martin, Juliana Alves Pereira, Arnaud Blouin, Djamel Eddine Khelladi, Jean-Marc Jézéquel
  - [https://hal.inria.fr/hal-02147012](https://hal.inria.fr/hal-02147012)

- **Learning Very Large Configuration Spaces: What Matters for Linux Kernel Sizes**
  - Mathieu Acher, Hugo Martin, Juliana Pereira, Arnaud Blouin, Jean-Marc Jézéquel, Djamel Eddine Khelladi, Luc Lesoil, Olivier Barais
  - [https://hal.inria.fr/hal-02314830](https://hal.inria.fr/hal-02314830)
Linux everywhere since
highly configurable

config X86_X2APIC
  bool "Support x2apic"
  depends on X86_LOCAL_APIC & X86_64 & (IRQ_REMAP || HYPERVERSION_GUEST)
  ---help---
  This enables x2apic support on CPUs that have this feature.

  This allows 32-bit apic IDs (so it can support very large systems),

config IOSF_MBI
  tristate "Intel SoC IOSF Sideband support for SoC platforms"
  depends on PCI
  ---help---
  This option enables sideband register access support for Intel SoC platforms. On these platforms the IOSF sideband is used in lieu of MSR’s for some register accesses, mostly but not limited to thermal and power. Drivers may query the availability of this device to determine if they need the sideband in order to work on these platforms. The sideband is available on the following SoC products.

# # Processor type and features #
# CONFIG_ZONE_DMA is not set
# CONFIG_SMP is not set
# CONFIG_X86_FEATURE_NAMES is not set
# CONFIG_X86_FAST_FEATURE_TESTS is not set
# CONFIG_X86_X2APIC=y
# CONFIG_X86_MPPARSE=y
# CONFIG_GOLDFISH=y
# CONFIG_INTEL_RDT_A is not set
# CONFIG_X86_EXTENDED_PLATFORM is not set
# CONFIG_IOSF_MBI=m
# CONFIG_IOSF_MBI_DEBUG=y
# CONFIG_X86_SUPPORTS_MEMORY_FAILURE=y
# CONFIG_SCHED OMIT_FRAM_E_POINTER is not set

Kconfig files/doc .config
15,000+ options

Configurations: Hell or Heaven?

Stop ou encore?
How to ensure that all configurations of the Linux kernel build/boot?

Many failures are due to buggy (combinations of) options
Devs/maintainers are struggling to track/fix bugs
Linus Torvalds: “random crazy user bugs” (random configurations are certainly a good subset)

Testing Linux kernels (on few configs):
(e.g., 0-day/KernelCI)
Given a kernel configuration, what’s its size/boot time?

Who knows what’s the effect of options?

Default configurations/options’ values
Documentation (Kconfig)
Configurators
Effects of (combinations of) options on build status/boot/size/boot time/performance/security?

```
# Processor type and features
#
# CONFIG_ZONE_DMA is not set
# CONFIG_SMP is not set
# CONFIG_X86_FEATURE_NAMES is not set
# CONFIG_X86_FAST_FEATURE_TESTS is not set
CONFIG_X86_XZAPIC=y
CONFIG_X86_MPPARSE=y
CONFIG_GOLDFISH=y
# CONFIG_INTEL_RDT_A is not set
# CONFIG_X86_EXTENDED_PLATFORM is not set
CONFIG_IOSF_MBI=m
CONFIG_IOSF_MBI_DEBUG=y
CONFIG_X86_SUPPORTS_MEMORY_FAILURE=y
# CONFIG_SCHED_OMIT_FRAME_POINTER is not set
```

General problem:
Taming the configuration space
15,000+ options

≈ $10^{60000}$ configurations
(without constraints)

Linux 5.2.8, arm
(% of types’ options)

- TRISTATE: 61.63
- BOOL: 36.40
- INT: 1.54
- STRING: 0.29
- HEX: 0.14

≈ $10^{60000}$ configurations
(without constraints)
Linux Kernel
≈\(10^{6000}\)
configurations
≈ 10^80 is the estimated number of possible chess positions in the universe.
Linux vs AlphaZero

Building a kernel configuration takes **10 minutes on average** on a recent machine.

Trial and error is **cheap** for Chess/Go, you can experience winning/losing billions of time.
AlphaZero vs Linux

In Chess/Go, you can fully observe the outcome, without noise and with a perfect simulator.

Think about technically measuring the boot time of a kernel out of a configuration.
Is taming the Linux kernel configuration space a harder problem than resolving Chess?

# configurations
≈10^{6000}

exploration
costly and hard to engineer

≈10^{40}

cheap with a perfect simulator
You cannot build $\approx 10^{6000}$ configurations.

**TUXML:** predicting out of a (small) sample of configurations’ kernels

Classification problem: predict the class (BUILD/FAILURE) out of options values
You cannot build $\approx 10^{6000}$ configurations.

**TUXML: predicting out of a (small) sample of configurations’ kernels**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIG_PM_WAKELOCKS=y</td>
<td>7.1Mb</td>
</tr>
<tr>
<td>CONFIG_PM_WAKELOCKS_LIMIT=100</td>
<td>176.8Mb</td>
</tr>
<tr>
<td>CONFIG_PM_GENERIC_DOMAINS=y</td>
<td>16.1Mb</td>
</tr>
<tr>
<td>CONFIG_PM_GENERIC_DOMAINS_OF=y</td>
<td>102.3Mb</td>
</tr>
</tbody>
</table>

Regression problem: predict a quantitative value (eg size) out of options values.
You cannot build $\approx 10^{6000}$ configurations.

TUXML: predicting out of a (small) sample of configurations’ kernels
You cannot build \( \approx 10^{6000} \) configurations.
Is machine learning effective for such very large configurable systems?
Answers in the rest of the talk

- Sampling and Learning with TUXML
- Results over 150K+ configurations
  - build failure understanding and prevention
  - kernel size prediction
- Challenges
  - “smart” build infrastructure
  - with devs/contributors in the loop
TUXML: Sampling, Measuring, Learning

random generation of configurations (randconfig)

TuxML

statistical machine learning

compile/build and measure

build failures (with Make/gcc messages)
Docker for a reproducible environment with tools/packages needed and Python procedures inside

Easy to launch campaign: “python kernel_generator.py 10”

builds/measures
10 random configurations
(information sent to a database)
TUXML: Sampling, Measuring, Learning

https://github.com/TuxML/

<table>
<thead>
<tr>
<th>cid</th>
<th>compilation_date</th>
<th>compilation_time</th>
<th>config_file</th>
<th>stdout_log_file</th>
<th>stderr_log_file</th>
<th>user_output_file</th>
<th>compiled_kernel_size</th>
<th>compressed_compiled_kernel_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>74882</td>
<td>2019-08-12 17:09:42</td>
<td>399.656</td>
<td>[BLOB - 24,3 Kio]</td>
<td>[BLOB - 33,7 Kio]</td>
<td>[BLOB - 14 o]</td>
<td>[BLOB - 702 o]</td>
<td>74559280</td>
<td>GZIP-bzImage : 8855504 , GZIP-vmlinux : 10943304 ,...</td>
</tr>
<tr>
<td>74881</td>
<td>2019-08-12 16:58:09</td>
<td>460.392</td>
<td>[BLOB - 25,8 Kio]</td>
<td>[BLOB - 34,7 Kio]</td>
<td>[BLOB - 14 o]</td>
<td>[BLOB - 704 o]</td>
<td>81377768</td>
<td>GZIP-bzImage : 18375632 , GZIP-vmlinux : 20462408 ,...</td>
</tr>
<tr>
<td>74880</td>
<td>2019-08-12 16:47:28</td>
<td>301.775</td>
<td>[BLOB - 22 Kio]</td>
<td>[BLOB - 24,2 Kio]</td>
<td>[BLOB - 14 o]</td>
<td>[BLOB - 705 o]</td>
<td>83004496</td>
<td>GZIP-bzImage : 14365648 , GZIP-vmlinux : 18452424 ,...</td>
</tr>
<tr>
<td>74879</td>
<td>2019-08-12 16:46:14</td>
<td>1393.61</td>
<td>[BLOB - 24,1 Kio]</td>
<td>[BLOB - 50 Kio]</td>
<td>[BLOB - 571 o]</td>
<td>[BLOB - 712 o]</td>
<td>109098328</td>
<td>GZIP-bzImage : 17183792 , GZIP-vmlinux : 19272160 ,...</td>
</tr>
<tr>
<td>74878</td>
<td>2019-08-12 16:45:03</td>
<td>305.705</td>
<td>[BLOB - 26,1 Kio]</td>
<td>[BLOB - 28,8 Kio]</td>
<td>[BLOB - 14 o]</td>
<td>[BLOB - 703 o]</td>
<td>55523752</td>
<td>GZIP-bzImage : 14767568 , GZIP-vmlinux : 18852988</td>
</tr>
</tbody>
</table>

(information sent to a database)
Data: version 4.13.3 and 4.15 (x86_64)

<table>
<thead>
<tr>
<th>cid</th>
<th>compilation_date</th>
<th>compilation_time</th>
<th>config_file</th>
<th>stdout_log_file</th>
<th>stderr_log_file</th>
<th>user_output_file</th>
<th>compiled_kernel_size</th>
<th>compressed_compiled_kernel_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>74882</td>
<td>2019-08-12 17:09:42</td>
<td>399.866</td>
<td>BLOB - 24.3 Kio</td>
<td>BLOB - 33.7 Kio</td>
<td>BLOB - 14 o</td>
<td>BLOB - 702 o</td>
<td>74559280</td>
<td>GZIP-bzImage : 8855504 , GZIP-vmlinux : 10943304 , ...</td>
</tr>
<tr>
<td>74881</td>
<td>2019-08-12 16:58:09</td>
<td>460.392</td>
<td>BLOB - 25.8 Kio</td>
<td>BLOB - 34.7 Kio</td>
<td>BLOB - 14 o</td>
<td>BLOB - 704 o</td>
<td>81377768</td>
<td>GZIP-bzImage : 18375632 , GZIP-vmlinux : 20462408 , ...</td>
</tr>
<tr>
<td>74880</td>
<td>2019-08-12 16:47:28</td>
<td>301.775</td>
<td>BLOB - 22 Kio</td>
<td>BLOB - 24.2 Kio</td>
<td>BLOB - 14 o</td>
<td>BLOB - 705 o</td>
<td>83004496</td>
<td>GZIP-bzImage : 14365848 , GZIP-vmlinux : 16452424 , ...</td>
</tr>
<tr>
<td>74879</td>
<td>2019-08-12 16:46:14</td>
<td>1393.61</td>
<td>BLOB - 24.1 Kio</td>
<td>BLOB - 50 Kio</td>
<td>BLOB - 571 o</td>
<td>BLOB - 712 o</td>
<td>109098328</td>
<td>GZIP-bzImage : 17183792 , GZIP-vmlinux : 19272160 , ...</td>
</tr>
<tr>
<td>74878</td>
<td>2019-08-12 16:45:03</td>
<td>305.705</td>
<td>BLOB - 26.1 Kio</td>
<td>BLOB - 28.8 Kio</td>
<td>BLOB - 14 o</td>
<td>BLOB - 703 o</td>
<td>55523752</td>
<td>GZIP-bzImage : 14767568 , GZIP-vmlinux : 16852888 , ...</td>
</tr>
</tbody>
</table>

74K+ configurations for Linux 4.15
95K+ configurations for Linux 4.13.3
(and 15K hours of computation on a grid computing)
Application 1: “Smart” build infrastructure

95,854 configurations
3,164 configuration failures
5.83% of build lead to failures

Should we send 3,164 bug reports?
Application 1: “Smart” build infrastructure

One configuration **bug** can lead to many configuration failures

**DRM_VBOXVIDEO & GENERIC_ALLOCATOR**

367 failures like this

```
make: *** [vmlinux] Error 1
```

Statistical learning can **automatically** pinpoint what combinations of options lead to a failure
Classification problem: predict the class (BUILD/FAILURE) out of options values

Do you recognize a pattern?

(matrix is 95K rows and 12K columns)
Classification tree

Do you recognize a pattern?
(matrix is 95K rows and 12K columns)
Application 1: “Smart” build infrastructure

Some config. bugs can mask/dominante other config. bugs!

Solution (see paper): statistical learning combined with clustering of error messages (multi-class classification)
**Application 1: “Smart” build infrastructure**

5.83% of build failures can be explained by 16 config. bugs of Linux and 3 config. bugs of TUXML

<table>
<thead>
<tr>
<th>nb_failures</th>
<th>percentage</th>
<th>bug (faulty option)</th>
<th>Bug?</th>
<th>Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>2464</td>
<td>68.05</td>
<td>AIC7XXX_BUILD_FIRMWARE</td>
<td>TUXML</td>
<td>missing tools / Kconfig doc.</td>
</tr>
<tr>
<td>476</td>
<td>13.15</td>
<td>WANXL_BUILD_FIRMWARE</td>
<td>TUXML</td>
<td>missing tools / Kconfig doc.</td>
</tr>
<tr>
<td>367</td>
<td>10.14</td>
<td>DRM_VBOXVIDEO &amp; GENERIC_ALLOCATOR</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>161</td>
<td>4.45</td>
<td>AIC79XX_BUILD_FIRMWARE</td>
<td>TUXML</td>
<td>missing tools / Kconfig doc.</td>
</tr>
<tr>
<td>83</td>
<td>2.29</td>
<td>FORTIFY_SOURCE &amp; UBSAN_SANITIZE_ALL &amp; INFINIBA...</td>
<td>Linux</td>
<td>source code</td>
</tr>
<tr>
<td>19</td>
<td>0.52</td>
<td>VIDEO_MUX &amp; VIDEO_V4L2</td>
<td>Linux</td>
<td>source code</td>
</tr>
<tr>
<td>15</td>
<td>0.41</td>
<td>BACKLIGHT_CLASS_DEVICE &amp; DRM_I915 &amp; DRM_SAVAGE...</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>13</td>
<td>0.36</td>
<td>DRM_VBOXVIDEO &amp; DRM_TTM</td>
<td>Linux</td>
<td>source code</td>
</tr>
<tr>
<td>6</td>
<td>0.17</td>
<td>NLS &amp; ...</td>
<td>Linux</td>
<td>source code</td>
</tr>
<tr>
<td>3</td>
<td>0.08</td>
<td>SPL_ICORE &amp; ...</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>3</td>
<td>0.08</td>
<td>GPIOLIB &amp; ...</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>2</td>
<td>0.06</td>
<td>CRC32 &amp; VIDEO</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>2</td>
<td>0.06</td>
<td>BT_HCIUART_H4 &amp; ...</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>2</td>
<td>0.06</td>
<td>REGMAP_MMIO &amp; ...</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>1</td>
<td>0.03</td>
<td>VIDEO_SAAT344_GO7007 &amp; SND_SOC_RT5514_SPI</td>
<td>Linux</td>
<td>source code + Kconfig dep.</td>
</tr>
<tr>
<td>1</td>
<td>0.03</td>
<td>USB_F_TCM &amp; ...</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>1</td>
<td>0.03</td>
<td>VIDEO_SOLO6X10 &amp; ...</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>1</td>
<td>0.03</td>
<td>VIDEO_ATOMISP &amp; ...</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
<tr>
<td>1</td>
<td>0.03</td>
<td>NEW_LEDS &amp; ...</td>
<td>Linux</td>
<td>Kconfig dependency</td>
</tr>
</tbody>
</table>
Application 1: “Smart” build infrastructure

5.83% of build failures can be explained by
16 config. bugs of Linux and 3 config. bugs of TUXML

Don’t trust your configuration build infrastructure! Prevent/Fix as early as possible configuration bugs (otherwise you won’t see other bugs!)

Bug location/understanding: TUXML can help to pinpoint responsible options (and avoid sending duplicate bugs)

TUXML can prevent failures and avoid building buggy configs (until a fix is done) with a good accuracy
Unfortunately, nobody knows the precise effect of (combinations of) options on size

Kconfig: (only?) 150 options are explicitly referring to size
Regression problem: predict a quantitative value (e.g., size) out of options values

Smart configuration: prediction model can quantify the effect of (de-)activating options (optimizer/recommender/configurator can be built on top of it)

Documentation/default config. improvement: identification of “influential” options
Application 2: Kernel Size Prediction
Application 2: Kernel Size Prediction

vmlinux and compressed sizes

<table>
<thead>
<tr>
<th></th>
<th>GZIP</th>
<th>BZIP2</th>
<th>LZMA</th>
<th>XZ</th>
<th>LZO</th>
<th>LZ4</th>
</tr>
</thead>
<tbody>
<tr>
<td>GZIPo</td>
<td>0</td>
<td>-28.1465</td>
<td>17.6087</td>
<td>25.5951</td>
<td>-7.48979</td>
<td>-12.3437</td>
</tr>
<tr>
<td>BZIP2o</td>
<td>41.0556</td>
<td>0</td>
<td>65.3317</td>
<td>76.2405</td>
<td>30.6506</td>
<td>23.8874</td>
</tr>
<tr>
<td>XZO</td>
<td>-20.0819</td>
<td>-42.8681</td>
<td>-6.15874</td>
<td>0</td>
<td>-26.0152</td>
<td>-29.8666</td>
</tr>
<tr>
<td>LZ4o</td>
<td>8.14584</td>
<td>-22.1959</td>
<td>27.2664</td>
<td>35.9318</td>
<td>0</td>
<td>-5.26847</td>
</tr>
</tbody>
</table>

config KERNEL_XZ
bool "XZ"
depends on HAVE_KERNEL_XZ
help
XZ uses the LZMA2 algorithm and instruction set specific
BCJ filters which can improve compression ratio of executable
code. The size of the kernel is about 30% smaller with XZ in
comparison to gzip. On architectures for which there is a BCJ
filter (i386, x86_64, ARM, IA-64, PowerPC, and SPARC), XZ
will create a few percent smaller kernel than plain LZMA.
Application 2: Kernel Size Prediction

(size of vmlinux)
Max: 1,698.14Mb
Min: 7Mb (tinyconfig)
We find a sweet spot where only 200–300 features are sufficient to efficiently train a random forest and a Gradient Boosting Tree to obtain a prediction model that outperforms other baselines (7% prediction errors for 40K configurations). We observe similar feature selection benefits for any training set size and tree-based learning algorithms.

Towards smart configuration assistant (optimizer/recommender/configurator)

### Table 1: MAPE of different learning algorithms for the prediction of vmlinux size, without and with feature selection

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Without Feature Selection</th>
<th>With Feature Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N=10</td>
<td>N=20</td>
</tr>
<tr>
<td>OLS Regression</td>
<td>74.54±2.3</td>
<td>68.76±1.03</td>
</tr>
<tr>
<td>Lasso</td>
<td>34.13±1.38</td>
<td>34.32±0.12</td>
</tr>
<tr>
<td>Ridge</td>
<td>139.63±1.13</td>
<td>91.43±1.07</td>
</tr>
<tr>
<td>ElasticNet</td>
<td>79.26±0.9</td>
<td>80.81±1.05</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>15.18±0.13</td>
<td>13.21±0.12</td>
</tr>
<tr>
<td>Random Forest</td>
<td>12.5±0.19</td>
<td>10.75±0.07</td>
</tr>
<tr>
<td>GB Tree</td>
<td>11.13±0.23</td>
<td>9.43±0.07</td>
</tr>
<tr>
<td>N. Networks</td>
<td>16.73±1.30</td>
<td>11.38±0.27</td>
</tr>
<tr>
<td>Polynomial Reg.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Section 2: Kernel Size Prediction

Towards improved documentation/default config. and informed configurations’ decisions

Thanks to our prediction model, we have effectively identified a list of important features that is consistent with the options and strategy of tinyconfig, the Kconfig documentation, and Linux knowledge. We also found options that can be used to refine or augment the documentation.
Challenges

- Retrospectively and despite our investment, we found relatively few bugs of Linux
  - Is it due to the way we sample?
  - Is it due to the stable version of Linux we chose?
  - Is it due to the high-quality of Linux, its contributors and its industry-strength, community-based effort?
Challenges

• Sampling is based on randconfig
  – randconfig does not produce uniform random samples
  – hypothesis: the testing “community” has over-fitted randconfig

• We need other sampling strategies!
  – Uniform (but SAT-based techniques should be improved)
  – Coverage-based sampling (e.g., t-wise)
  – Knowledge-based sampling
Challenges

• The cost of gathering data is important (15K+ hours of computation)
• Incremental build of configurations
• Bugs do not transfer well
• However, kernel size “knowledge” may transfer
  – Instead of starting from scratch, we can transfer a prediction model for another version of Linux (ongoing work)
Challenges

• Kernel CI / 0-day
  – Our focus: testing configurations in the large
  – Complementary!
  – Learning techniques can be used in both contexts
  – Sharing data

• Unify the force!
Challenges

• “Smart” build infrastructure
  – Other properties (e.g., warnings, boot, security)

• With devs/contributors in the loop
  – We need knowledge to validate our learning model
  – We need knowledge to apply “smart” sampling
  – We aim to produce actionable knowledge

• TUXML needs you!
Conclusion (feedbacks welcome!)

• Learning From Thousands of Build Failures of Linux Kernel Configurations
  – Mathieu Acher, Hugo Martin, Juliana Alves Pereira, Arnaud Blouin, Djamel Eddine Khelladi, Jean-Marc Jézéquel
  – https://hal.inria.fr/hal-02147012

• Learning Very Large Configuration Spaces: What Matters for Linux Kernel Sizes
  – Mathieu Acher, Hugo Martin, Juliana Pereira, Arnaud Blouin, Jean-Marc Jézéquel, Djamel Eddine Khelladi, Luc Lesoil, Olivier Barais
  – https://hal.inria.fr/hal-02314830
Some related work

- Julia Lawall and Gilles Muller “JMake: Dependable Compilation for Kernel Janitors.” In 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017
- Iago Abal, Claus Brabrand, and Andrzej Wasowski “42 variability bugs in the linux kernel: a qualitative analysis”. In ACM/IEEE International Conference on Automated Software Engineering, ASE’14
- Jean Melo, Elvis Flesborg, Claus Brabrand, and Andrzej Wasowski “A Quantitative Analysis of VariabilityWarnings in Linux”. In Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS’16)
Some related work

- Austin Mordahl, Jeho Oh, Ugur Koc, Shiyi Wei, Paul Gazzillo: An empirical study of real-world variability bugs detected by variability-obliviouso tools. ESEC/SIGSOFT FSE 2019: 50-61
Thanks!

- DiverSE research team [http://diverse-team.fr](http://diverse-team.fr)
  - Hugo Martin, Juliana Alves Pereira, Arnaud Blouin, Jean-Marc Jézéquel, Djamel Eddine Khelladi, Luc Lesoil, Olivier Barais
- TUXML team at ISTIC / University of Rennes 1
  - Paul Saffray, Alexis Le Masle, Michaël Picard, Corentin Chédotal, Gwendal Didot, Dorian Dumanghet, Antonin Garret, Erwan Le Flem, Pierre Le Luron, Mickaël Lebreton, Fahim Merzouk, Valentin Petit, Julien Royon Chalendard, Cyril Hamon, Luis Thomas, Alexis Bonnet
- IGRIDA [http://igrida.gforge.inria.fr](http://igrida.gforge.inria.fr)
- Tim Bird (Sony) and Greg Kroah-Hartman (Linux foundation)
- Julia Lawall (for challenging us to attend ELC!)
Intrigued by Tux logos?
Have a look and don’t hesitate to contribute!
https://github.com/diverse-project/tuxart

Side project: Tux generator out of arbitrary Linux kernel configurations (.config)

Khaled Arsalane
Eliot Marie
Pierre Pouteau
Zakariae Boukhchen
Richard Faraji-Huon
Mathieu Acher
Application 1: “Smart” build infrastructure

5.83% of build failures
BUT
only due to 16 configuration bugs of Linux and 3 configuration bugs of... TUXML

We come to this insight thanks to our learning procedure