NAB: Automated Large-Scale Multi-language Dynamic Program Analysis in Public Code Repositories

Alex Villazón[1], Haiyang Sun[2], Andrea Rosà[2], Eduardo Rosales[2], Daniele Bonetta[3], Isabella Defillipis[1], Sergio Oporto[1], Walter Binder[2]

[1]Universidad Privada Boliviana (UPB), Bolivia
[2]Università della Svizzera italiana (USI), Switzerland
[3] Oracle Labs, United States

Goal

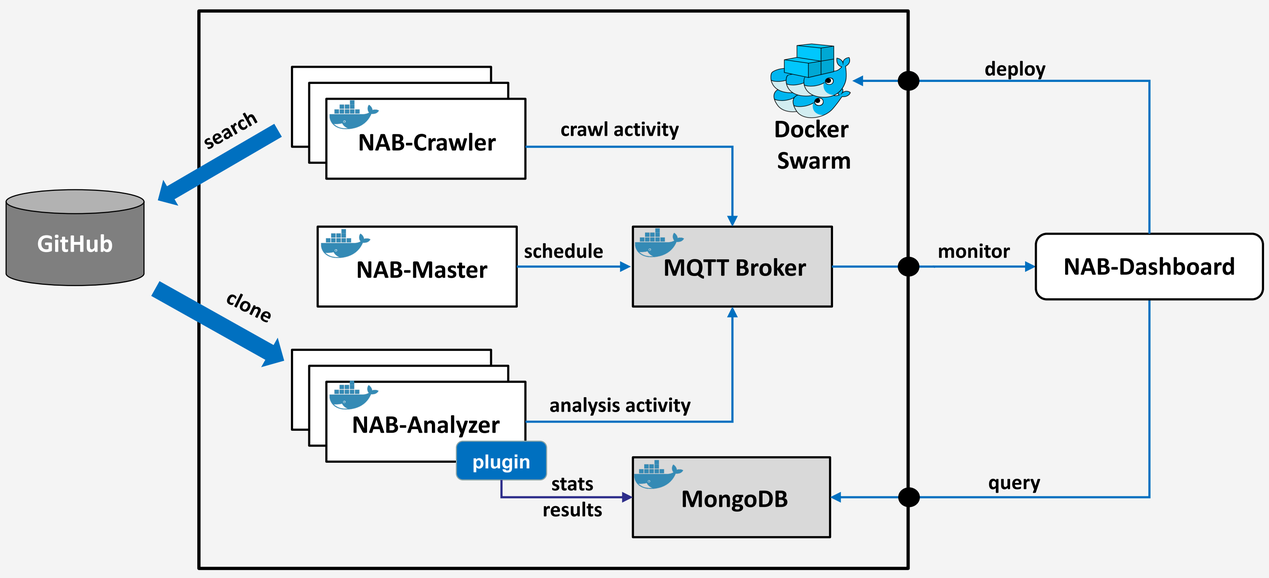
- ❑ Provide a tool for massive Dynamic Program Analysis (DPA) on large code repositories
- ❑ Automatically crawl, filter, clone, build, run tests, apply DPA tool, collect results, and analyze thousands of open-source projects
- ❑ Support of multiple programming languages and runtimes
- ❑ Easy integration of existing DPA tools

Motivation

- ❑ Exponential growth of ready-to-use open-source software in public repositories (e.g., GitHub)
- ❑ Lack of tools to automate DPA at the scale of large code repositories
- ❑ Need to perform large DPA studies in the wild, e.g.:
  - adoption of language constructs, statistics on bad coding practices, finding candidates for domain-specific benchmark suites, …

NAB: A Distributed Infrastructure for Automated DPA at Large Scale

- ❑ Container-based distributed infrastructure (Docker)
- ❑ Efficient publish-subscribe communication layer (MQTT)
- ❑ Supports multiple programming languages, build systems, analysis frameworks, DPA tools and runtimes
- ❑ Easy to deploy in computer clusters or in the Cloud



NAB Architecture

Core NAB Services (Master/Slave):

- ❑ **NAB-Crawler:** mines and crawls code repositories, collects meta-data to decide which projects to analyze
- ❑ **NAB-Master:** orchestrates the distribution of crawling and DPA activities
- ❑ **NAB-Analyzer:** downloads (clone) code from repositories, apply filters, and runs a DPA tool
- ❑ **NAB-Dashboard:** handles deployment of NAB services (using Docker Swarm) and monitors the progress of ongoing DPA

Existing Support Services:

- ❑ **MQTT Broker:** handles asynchronous communication through events
- ❑ **MongoDB:** stores DPA results, metrics and execution statistics

NAB Support

Language	Build System	Analysis Framework	DPA Tool	Runtime
JavaScript	NPM	NodeProf	Deep-Promise JITProf	GraalVM
Java	MVN	DiSL, AspectJ	tgp JavaMOP	HotSpot VM GraalVM
Scala	SBT MVM	DiSL	tgp	HotSpot VM GraalVM

References:

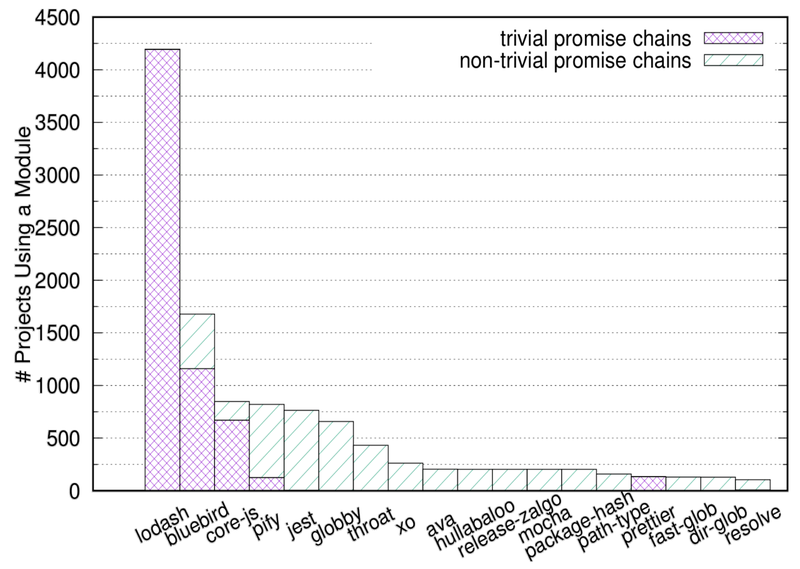
[1] https://github.com/Haiyang-Sun/nodeprof.js
[2] Madsen et al., A Model for Reasoning About JavaScript Promises. OOPSLA, 2017.
[3] Gong et al., JITProf: Pinpointing JIT-unfriendly JavaScript Code. ESEC/FSE, 2015.
[4] Rosà et al., Analyzing and Optimizing Task Granularity on the JVM. CGO, 2018.

GitHub Projects Selected for Case Studies (2013-2017)

- ❑ From **7.6 M projects** → **109,286 Node.js, 25,918 Java, 4,076 Scala projects**
  - correctly build, with test code, > 2 contributors

Case Study I: Use of Promises in Node.js

- ❑ **Goal:** Understand how developers use JavaScript's Promise API
- ❑ **DPA tool:**
  - **Deep-Promise** based on NodeProf [1] instrumentation
  - Creates a "*promise graph*" [2] showing dependencies among promises
  - Promise chain size gives insight on the use of the promise construct
- ❑ **Results:**
  - 23,297 projects analyzed successfully
  - 25.6% use Promise API
  - Only **10%** use non-trivial promise chains (i.e., chain size > 1)
  - Only **0.6%** use promises in application code → many projects do not directly use promises
  - 440 NPM modules use Promise API, 90.6% with non-trivial promise chains



Case Study II: JIT-unfriendly Code Patterns in Node.js

- ❑ **Goal:** Identify bad coding practices that affect Node.js application performance
- ❑ **DPA tool:**
  - **JITProf** [3], adapted to NodeProf instrumentation
  - 7 JIT-unfriendly code patterns
- ❑ **Results:**
  - 26,938 projects analyzed successfully (app-only profiling)
  - At least one JIT-unfriendly code pattern in **37%** projects and **22.8%** dependent NPM modules
  - Most affected NPM modules: *commander*, *glob*, *lodash*

JIT-unfriendly Pattern	# Projects	%
AccessUndefArrayElem	1,253	4.7%
BinaryOpOnUndef	757	2.8%
InconsistentObjectLayout	9,509	35.3%
NonContiguousArray	194	0.7%
PolymorphicOperation	3,073	11.4%
SwitchArrayType	81	0.3%
TypedArray	546	2.0%
At least one	**9,969**	**37.0%**

Case Study III: Discovering Task-parallel Workloads for Java and Scala

- ❑ **Goal:** Discover workloads for domain-specific benchmark suite (i.e., task-parallel applications on the JVM with diverse granularities)
- ❑ **DPA tool:**
  - **tgp** [4] task granularity profiler
  - *Granularity*: # bytecodes executed by a parallel task
- ❑ **Results:**
  - 1,769 Java (6.8%) and 860 Scala (21%) projects contain task-parallel workloads
  - Found 5 good benchmark candidates with high diversity:
    - 2 Java projects (55 and 123 tasks)
    - 3 Scala projects (5.7K, 19.8K and 20.9K tasks)

Granularity Range	Java Tasks	Java Projects	Scala Tasks	Scala Projects
$[10^0$ - $10^1)$	137,468	686	301,066	771
$[10^1$ - $10^2)$	278,765	466	280,244	710
$[10^2$ - $10^3)$	215,211	673	2,795,702	860
$[10^3$ - $10^4)$	285,196	1,092	1,278,974	769
$[10^4$ - $10^5)$	247,284	1,367	124,473	771
$[10^5$ - $10^6)$	128,992	1,492	74,989	769
$[10^6$ - $10^7)$	89,710	1,327	13,002	806
$[10^7$ - $10^8)$	17,178	1,046	4,555	677
$[10^8$ - $10^9)$	5,696	581	1,789	619
$[10^9$ - $10^{10})$	1,164	177	430	276
$[10^{10}$ - $10^{11})$	120	53	22	20
$[10^{11}$ - $10^{12})$	18	8	1	1

Download NAB

- ❑ Evaluation version at
  http://dag.inf.usi.ch/software/nab/
- ❑ Contacts:

  Alex Villazon: **avillazon@upb.edu**
  Andrea Rosà: **andrea.rosa@usi.ch**

Acknowledgments: This work has been supported by Oracle (ERO project 1332), the Swiss National Science Foundation (scientific exchange project IZSEZ0_177215), the Hasler Foundation (project 18012), and by a Bridging Grant with Japan (BG 04-122017).