Automated Large-scale Multi-language Dynamic Program Analysis in the Wild

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

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

ECOOP 2019
July 18, 2019
London, United Kingdom

Our Work

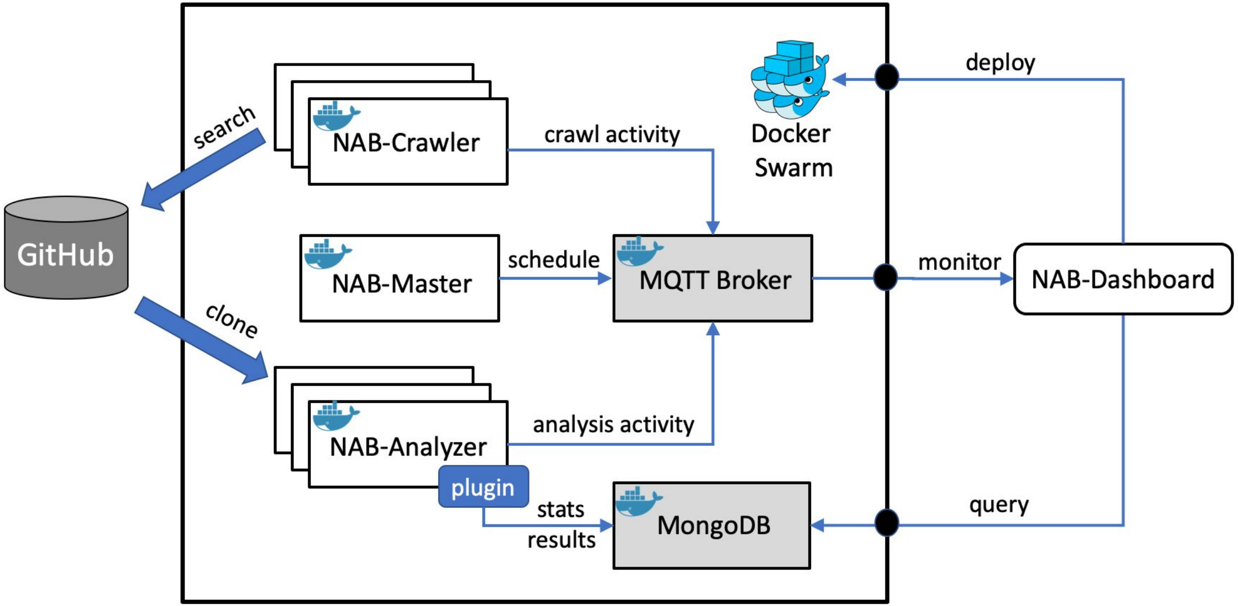
- **Goal:** Propose a methodology for **automatically** applying **Dynamic Program Analysis (DPA)** at a **large-scale** on projects hosted in public **open-source repositories**

- Motivation:
  - Applying DPA in large code repositories is increasingly important
  - Existing infrastructures focus mainly on static analysis

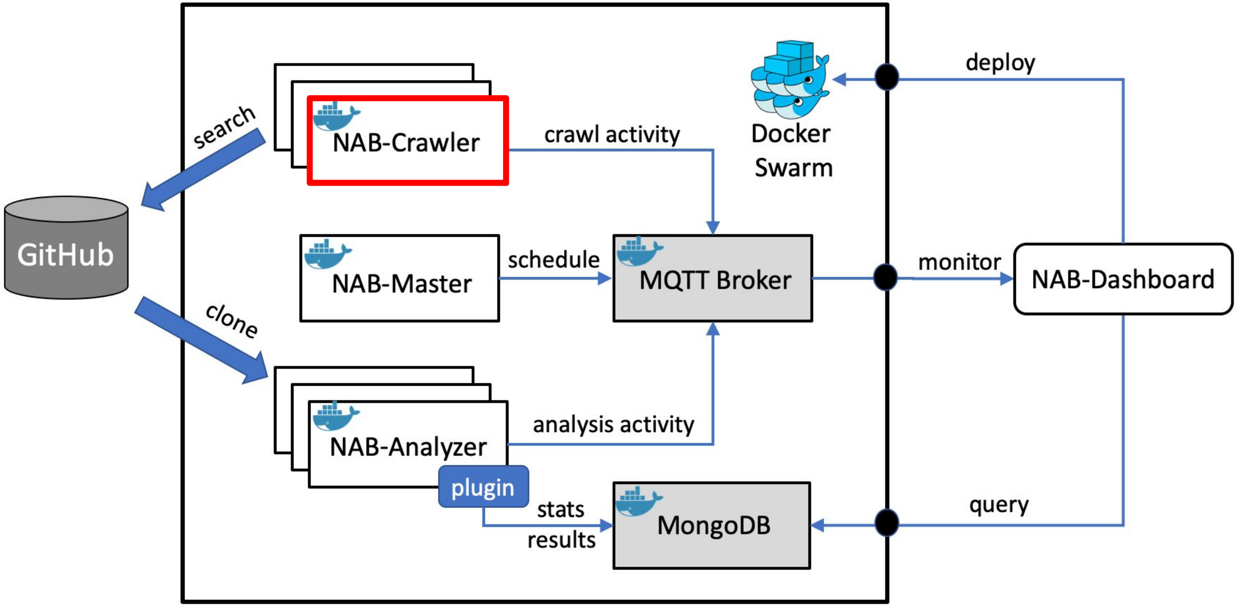
NAB: A Distributed Infrastructure for Automated DPA at Large Scale

- Automatically looks for executable code in public repositories
  - E.g., GitHub
- Filters out projects according to user-defined criteria
  - E.g., programming language, date of last commit, # contributors
- Attempts to apply DPA on workloads that can be automatically executed
  - E.g., tests (via build systems such as Maven, NPM, SBT)
- Uses containerization (Docker)
  - Simplified distributed deployment to increase scalability
  - Easy to integrate different runtimes; support for multiple languages
  - Natural and efficient sandboxing to protect from buggy or malicious code

NAB Architecture

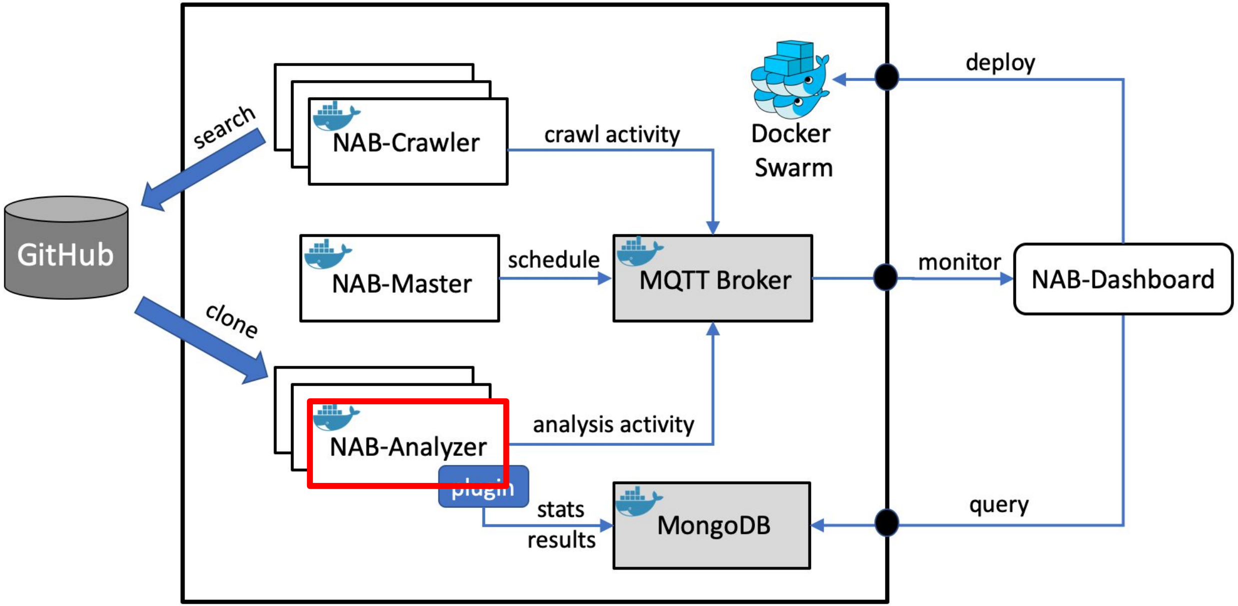


NAB Architecture



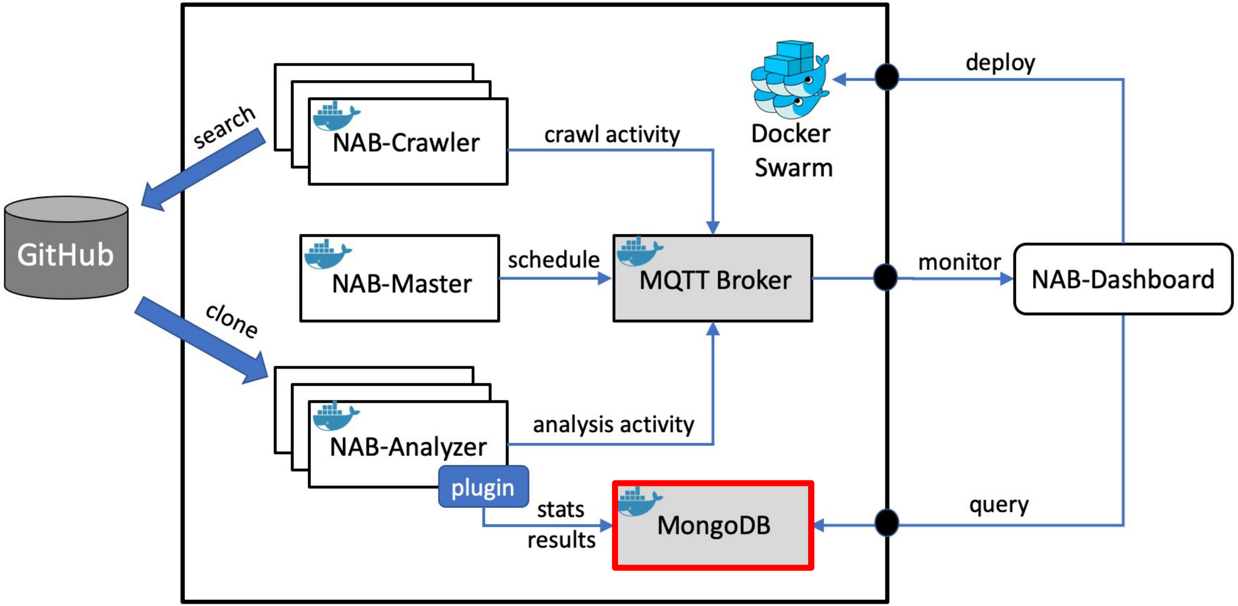
NAB-Crawler: crawls and mines code repositories,
determine projects to analyze (according to user-defined criteria)

NAB Architecture



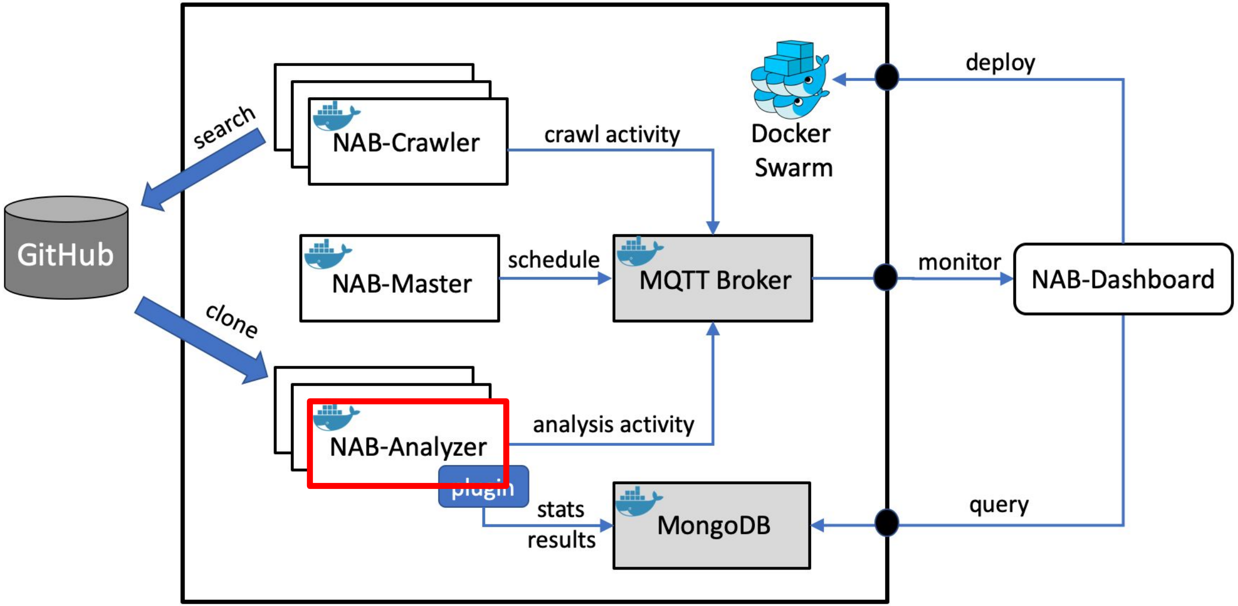
NAB-Analyzer: clones code from repositories, builds code, runs DPA on executable workloads

NAB Architecture



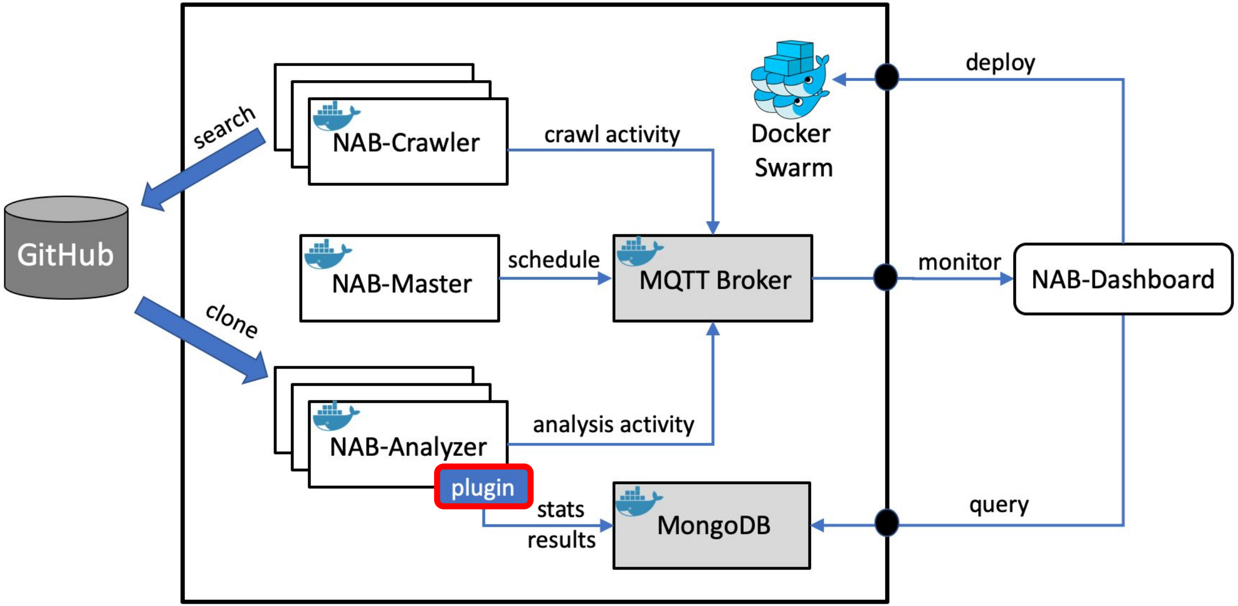
MongoDB: stores DPA results, metrics, and execution statistics

NAB Architecture



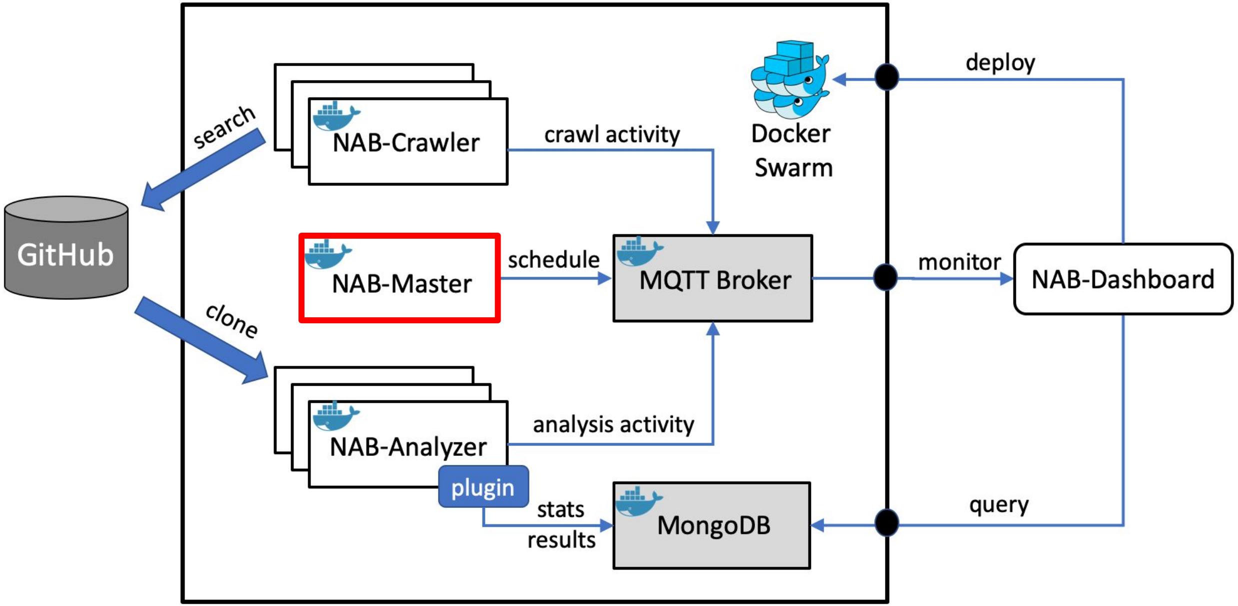
- Reports reasons of failures
- Configurable analysis timeout (default: 1 hour)

NAB Architecture



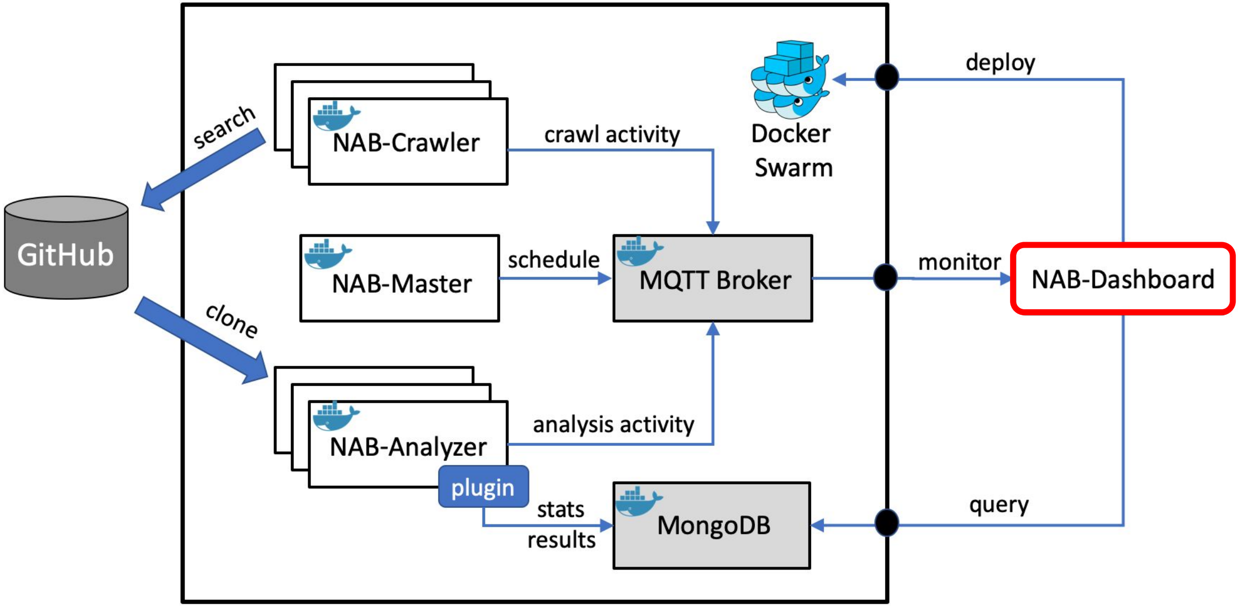
Plugin: mechanism to integrate *existing* DPA

NAB Architecture



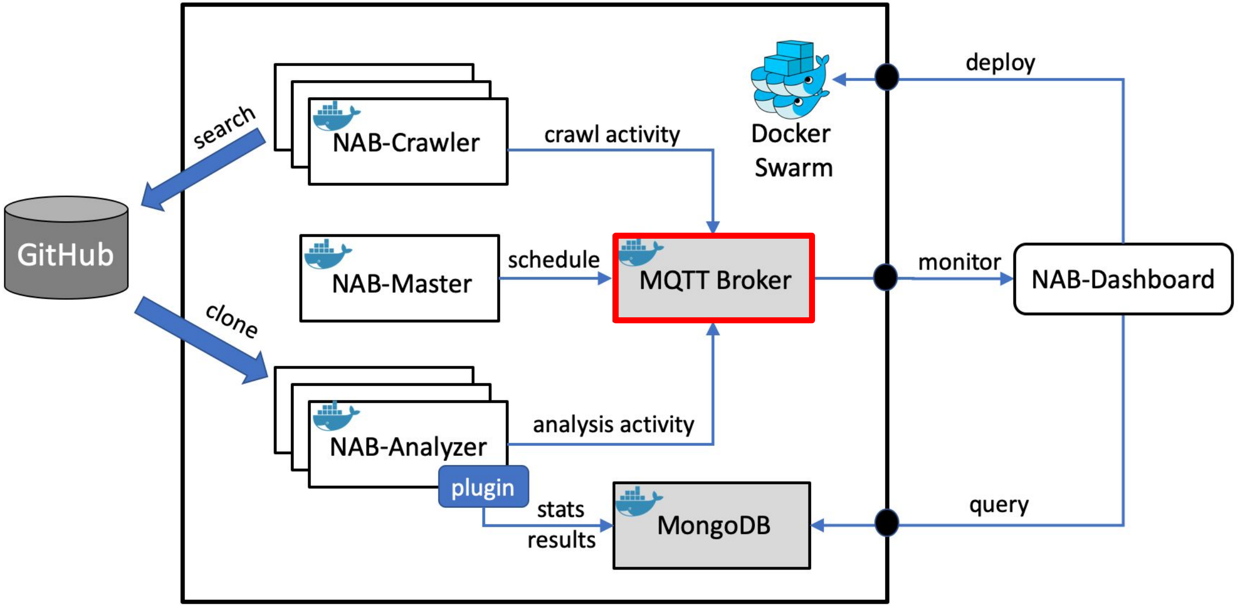
NAB-Master: orchestrates the distribution of crawling and DPA activities

NAB Architecture



NAB-Dashboard: handles deployment of NAB services (using Docker Swarm), allows users to monitor DPA progress

NAB Architecture

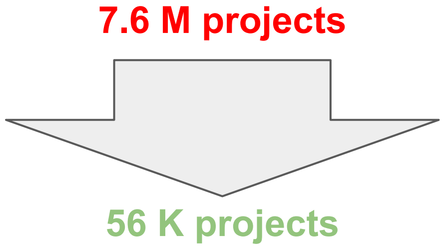


MQTT Broker: handles asynchronous communication through events (publish-subscribe communication protocol)

Case Studies

I Use of promises in Node.js applications
II JIT-unfriendly code patterns in Node.js applications
III Discovering Java and Scala task-parallel workloads for domain-specific benchmarking

- Codebase:
    - 5 years (2013-2017) of Node.js, Java, and Scala projects from GitHub



Case Study I: Use of Promises in Node.js

- **Goal:** Understand how developers use the JavaScript Promise API
- **DPA: Deep-Promise**
  - Generate **promise chain:** sequence of asynchronous events with logical dependencies
- Promise chain size gives insight on the use of promise construct
  - Size 1 (trivial) = not used to handle asynchronous executions

Case Study I: Results (1/2)

Use of Promise API in Node.js projects

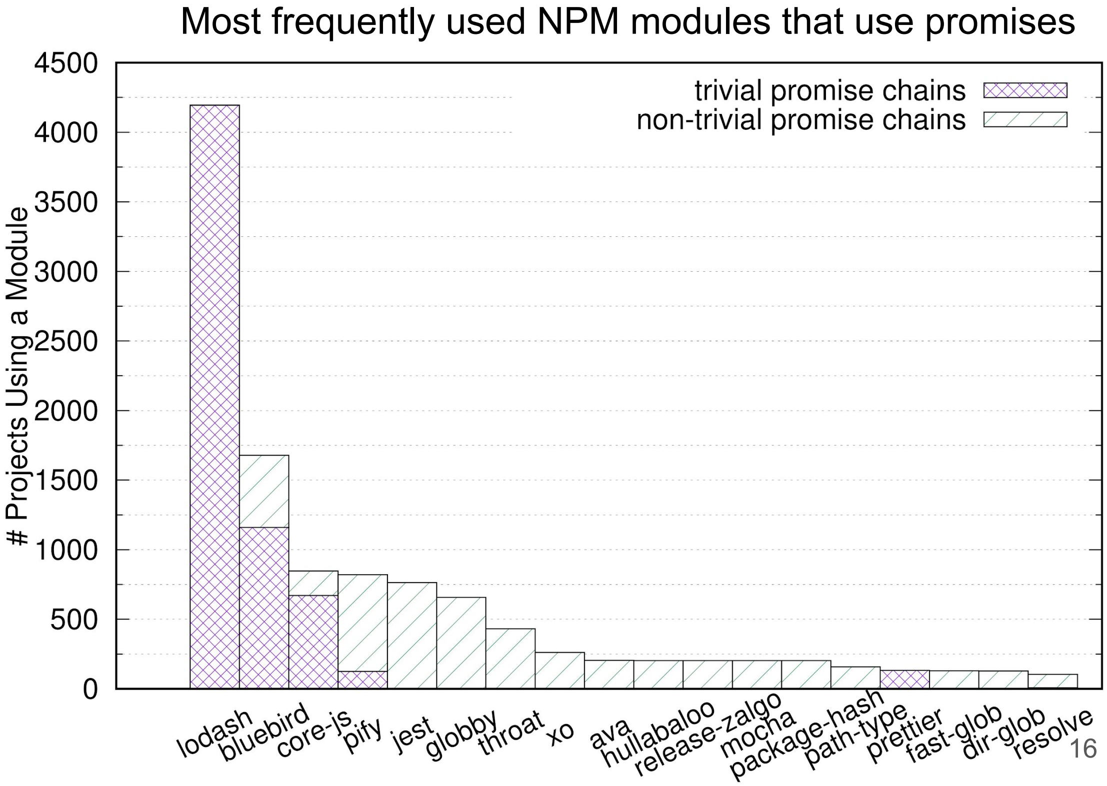
- 23,297 projects successfully analyzed
- 5,971 projects (25.6%) use Promise API
- **Only 10%** use non-trivial promises (chain size > 1)
- **Only 0.6%** use promises in application code

- **Many projects do not use promises directly**

Case Study I: Results (2/2)

Use of Promise API in NPM modules

- 440 NPM modules use Promise API
- 399 modules (90.7%) use non-trivial promise chains
- **Good candidates for evaluation and optimization**



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

- **Goal:** Identify bad coding practices that affect Node.js application performance
  - JIT-unfriendly code patterns
    - May prevent typical JIT compiler optimizations
- **DPA: JITProf** [1]
  - Detect 7 JIT-unfriendly code patterns in application code and NPM modules

[1] Gong et al., JITProf: Pinpointing JIT-unfriendly JavaScript Code. ESEC/FSE 2015

Case Study II: Results (1/2)

Application code

- 26,938 projects successfully analyzed
- **37% of the projects have at least one JIT-unfriendly pattern**
- Most common pattern: `InconsistentObjectLayout`
  - Indicates sub-optimal read/write operations to objects

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 II: Results (2/2)

NPM module code

- **900 NPM modules execute at least one JIT-unfriendly pattern**
- Most affected modules: ‘*commander*’, ‘*glob*’ ‘*lodash*’
  - Imported by > 130k other modules

JIT-unfriendly Pattern	# Modules	Top 3 NPM Modules		
AccessUndefArrayElem	252	*commander* (637)	*glob* (569)	*abbrev* (178)
BinaryOpOnUndef	83	*strip-json-comments* (110)	*jsbn* (110)	*sinon* (83)
InconsistentObjectLayout	523	*commander* (687)	*chai* (369)	*tape* (337)
NonContiguousArray	49	*semver* (167)	*jsbn* (110)	*eslint* (51)
PolymorphicOperation	453	*lodash* (311)	*glob* (178)	*mime-types* (174)
SwitchArrayType	16	*babylon* (4)	*lodash* (3)	*eslint* (3)
TypedArray	144	*lodash* (51)	*jshint* (48)	*regenerate* (38)
At least one	**900**	***commander* (963)**	***glob* (569)**	***lodash* (432)**

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

- **Goal:** Discover Java and Scala task-parallel workloads with diverse task granularity to analyze concurrency-related aspects
  - Granularity: number of bytecode instructions executed by a parallel task
- **DPA: tgp [1]** task granularity profiler
  - Collects granularity of all spawned tasks
    - Task = subtypes of **`Runnable`**, **`Callable`**, **`ForkJoinTask`**

[1] Rosà et al., Analyzing and Optimizing Task Granularity on the JVM. CGO 2018

Case Study III: Results (1/2)

Java workloads

- 1,769 projects successfully analyzed with task-parallel workloads
- Two workloads with granularities spanning all ranges
  - `https://github.com/rolfl/MicroBench`
  - `https://github.com/47Billion/netty-http`
- **Good candidates for benchmarking task execution in Java workloads**

Granularity	**Java**	
Range	**Tasks**	**Projects**
$[10^0 - 10^1)$	137,468	686
$[10^1 - 10^2)$	278,765	466
$[10^2 - 10^3)$	215,211	673
$[10^3 - 10^4)$	285,196	1,092
$[10^4 - 10^5)$	247,284	1,367
$[10^5 - 10^6)$	128,992	1,492
$[10^6 - 10^7)$	89,710	1,327
$[10^7 - 10^8)$	17,178	1,046
$[10^8 - 10^9)$	5,696	581
$[10^9 - 10^{10})$	1,164	177
$[10^{10} - 10^{11})$	120	53
$[10^{11} - 10^{12})$	18	8

Case Study III: Results (2/2)

Scala workloads

- 860 projects successfully analyzed with task-parallel workloads
- Three workloads with granularities spanning all ranges
  - `https://github.com/iheartradio/asobu`
  - `https://github.com/TiarkRompf/virtualization-lms-core`
  - `https://github.com/ryanlsg/gbf-raidfinder`
- **Good candidates for benchmarking task execution in Scala workloads**

Granularity Range	Scala Tasks	Scala Projects
$[10^0 - 10^1)$	301,066	771
$[10^1 - 10^2)$	280,244	710
$[10^2 - 10^3)$	2,795,702	860
$[10^3 - 10^4)$	1,278,974	769
$[10^4 - 10^5)$	124,473	771
$[10^5 - 10^6)$	74,989	769
$[10^6 - 10^7)$	13,002	806
$[10^7 - 10^8)$	4,555	677
$[10^8 - 10^9)$	1,789	619
$[10^9 - 10^{10})$	430	276
$[10^{10} - 10^{11})$	22	20
$[10^{11} - 10^{12})$	1	1

Conclusions

- We presented NAB: a novel, distributed infrastructure for executing massive custom DPA on open-source code repositories
- Three large-scale analyses thanks to NAB:
  - Use of promises in Node.js
    - Many projects don't use promises directly
    - We determined popular modules to optimize
  - JIT-unfriendly code patterns in Node.js
    - Node.js developers frequently use bad code patterns
  - Discovering task-parallel workloads for Java and Scala
    - We identified five candidate workloads to benchmarking task parallelism on the JVM

NAB

- Evaluation version at
  http://research.upb.edu/NAB/nab-artifact.tgz

- Contacts:
  Andrea Rosà:
  andrea.rosa@usi.ch
  http://www.inf.usi.ch/phd/rosaa

Thanks for your attention