Investigating Bug Prediction and Static Analysis Tools in the Cloud Context

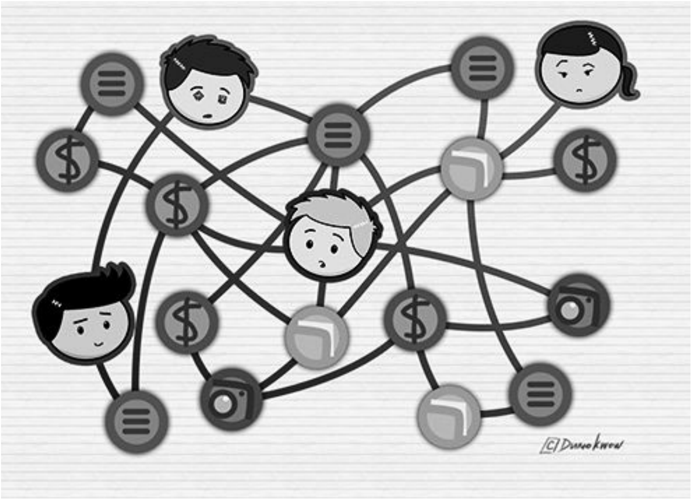
Anna Jancso & Yasara Peiris

Overview

- Context: Cloud computing
- RQ1: Bug prediction
  - Methodology
  - Results & Discussion
- RQ2: ASAT adoption
  - Methodology
  - Results & Discussion
- Summary and Future Work

Context

- Increasing complexity of modern software systems



Large codebases

More users

More instances

More dependencies

Context

- Increase use of cloud applications

But,

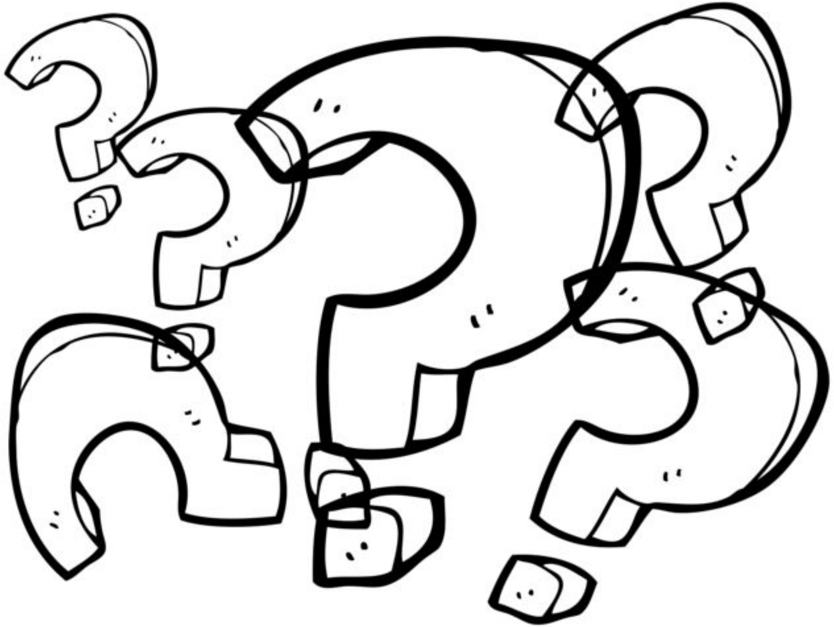
Software maintenance using existing tools is limited

What is cloud computing?

- Use of hardware and software to deliver a service over a network .
- On-demand availability of computer system resources.
- No direct active management by the user.
- More complex than simple, monolithic, single-server systems

RQ1: Bug Prediction

Bug Prediction



How to predict bugs in cloud applications?

Defects in Contemporary Projects

- Logic specific bugs (errors, functional)
- Design issues (e.g. too much coupling)
- Code style violations (e.g. bad indentation)
- Optimisation Bugs
- Data Loss or Data Corruption
- Error Handling Bugs
- Configuration Bugs

Defects in Contemporary Projects

Defect Detection		Positive	Negative
Manual	Code Reviews [7]		Time Consuming Expensive
Automatic	Testing [10]	Fast high precision	Expensive
	Prediction Tools [11,12] (Based on historical analysis, using metrics as features for ML model)	Fast Cheap	
	Automatic Static Analysis Tools (Tools that analyze code without program execution) [8,9]	Fast Cheap	Low Precision

Motivation

- Distributed computing has become backbone of computing. Cloud defects have different characteristics and are typically more difficult to detect.
- Existing static analysis tools support in contemporary projects in different tasks.
  - Detection of defects
  - Design Issues
  - Code Style Violations

Prioritization of warnings depending on the development context

Related Work - Classes of Bugs [2]

Classification	**Labels**
Aspect	Reliability, Performance, Availability,Security,Consistency, Scalability, Topology, Qos
Bug scope	Single Machine, Multiple Machines, Entire Cluster
Hardware	Core/Processor, Disk, Memory, Network

Related Work - Classes of Bugs [2]

Classification	**Labels**
HW Failure	Corrupt, Limp, Stop
Software	Logic, Error Handling, Optimization, config, Race, Hang Space, Load
Implication	Failed Operation, Performance, Component Downtime, Data Loss,

Classes of Cloud Bugs [2]

Topology

New Layering Architecture, Rack Awareness

Data Consistency

Killer Bugs

Cascading Failure, No single point of failure, Time of faults

Scalability

Scale of cluster size,
Scale of data size,
Scale of Request load

Classes of Cloud Bugs [2]

Crash Recovery Bugs

Performance Bugs
- Throughput
- Invocation Time
- No of users impacted

Distributed Concurrency Bugs

Research Question

What bugs affect cloud applications? Which metrics could be useful to predict them? Here we investigate the theories, metrics, and features enabling better bug prediction strategies.

Cloud bugs and their level of predictability

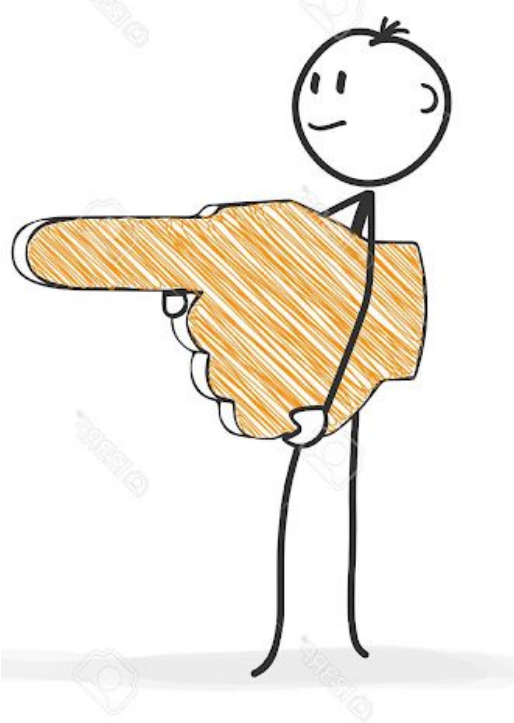
- Investigate bugs and failures affecting the evolution of cloud application.
- Investigating/experimenting potential metrics and criteria to predict such bugs

Methodology - RQ1 (defect prediction)

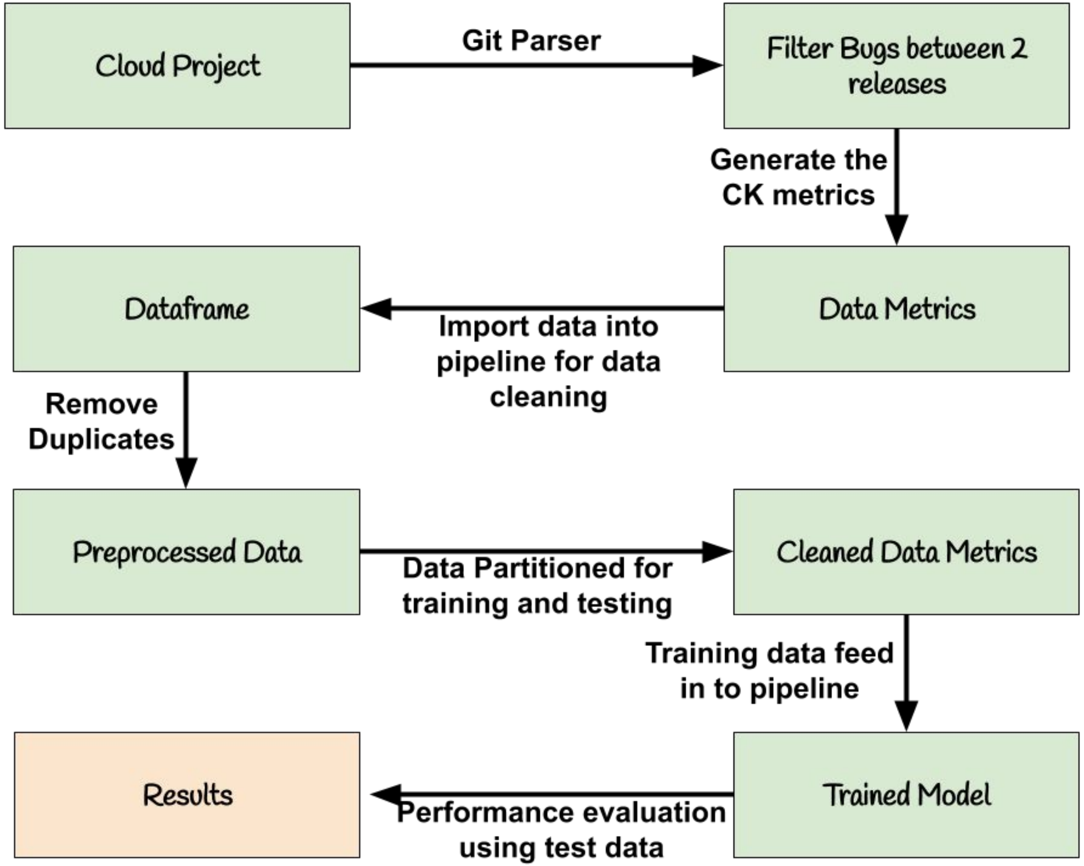
- Find a cloud project
- Investigate defects occurring between releases of these projects
- Define metrics that could predict such defects
- Compute source-code file-based metrics
- Use basic bug prediction pipeline to figure out important features in bug prediction in cloud context.

Metrics for Finding Cloud-Specific Bugs

- CK Metrics
  - LOC (lines of code)
  - CBO (number of children)
  - WMC (Weighted methods per class)
  - RFC(Response for a class)
  - ICOM(Lack of cohesion of methods)
  - DIT(Depth Inheritence Tree)



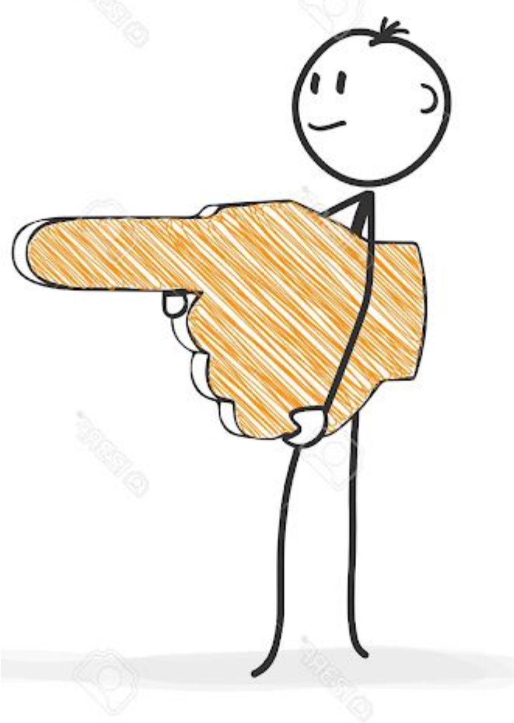
Defect Bug Prediction Process



Data Preparation

Data Ingestion

- Hadoop Project was selected for the experiment.
- JGit Java implementation of GIT version control system was used.
- 2 Releases of the project were selected and retrieved all the commits within the release.
- Keywords were used to filter commits for bug fixes.



Keywords for commit filtering - General Keywords

Failure

Error

Bug

Fix

Wrong

Unexpected

Crash

Bug Filtering Keywords

cloud_specific_bugs = ['distributed concurrency ','performance','single-point-of-failure ']
cloud_concurrency_bugs = ['thread', 'blocked', 'locked', 'race', 'dead-lock', 'deadlock', 'concurrent', 'concurrency', 'atomic', 'synchronize', 'synchronous', 'synchronization', 'starvation', 'suspension', 'order violation', 'atomicity violation', 'single variable atomicity violation', 'multi variable atomicity violation', 'livelock, live-lock', 'multi-threaded', 'multithreading', 'multi-thread']

Cloud Specific Keywords

optimization_bugs = ['optimization','optimize']
logical_bugs = ['logic','logical','programming logic','wrong logic']
performance_bugs = ['performance','load balancing','cloud bursting', 'performance implications']
configuration_bugs = ['configuration']
error_handling_bugs = ['error handling', 'exception', 'exceptions']
hang_bugs = ['hang','freeze','unresponsive''blocking','deadlock','infinite loop', 'user operation error']

Traditional Bug Prediction Pipeline

name	name.pr	version	loc	bug.x	cbo	dit	wmc	lcom	bug.y	bug
org.apache.hadoop.hbase.catalog.MetaEdit...	hadoop	0.92RC0	35	no	10	1	23	115	yes	yes
org.apache.hadoop.hbase.catalog.MetaRea...	hadoop	0.92RC0	228	no	22	2	78	528	yes	yes
org.apache.hadoop.hbase.client.coprocess...	hadoop	0.92RC0	78	no	16	1	15	3	NA	no
org.apache.hadoop.hbase.client.HTablePool	hadoop	0.92RC0	179	no	11	2	27	43	yes	yes
org.apache.hadoop.hbase.client.Mutation	hadoop	0.92RC0	0	no	6	2	20	59	NA	no
org.apache.hadoop.hbase.client.ScannerCal...	hadoop	0.92RC0	0	no	13	2	24	27	NA	no
org.apache.hadoop.hbase.client.TestAdmin	hadoop	0.92RC0	905	no	32	1	127	0	yes	yes
org.apache.hadoop.hbase.client.TestFromC...	hadoop	0.92RC0	2929	no	42	1	186	2411	yes	yes

Traditional Bug Prediction Pipeline - Process

- Python Weka wrapper
- Machine Learning Algorithms
  - Logistic Regression
  - Naive Bayes
  - Random Forest

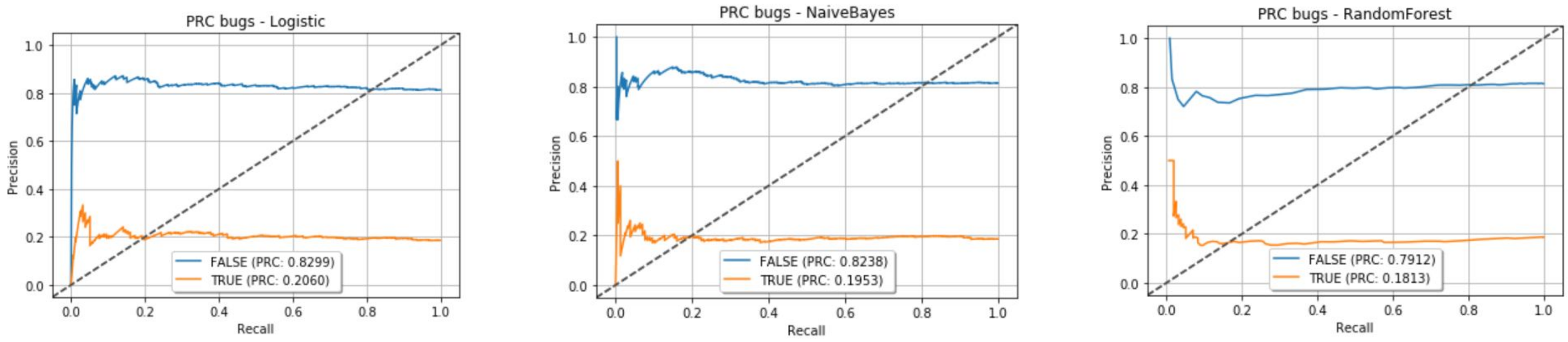
Evaluation

Precision - Fraction of the relevant instances among retreived instances - **True Positive/(False Positive + True Positive)**

Recall - Fraction of total amount of relevant instances were that were actually retrieved. - **True Positive/(False Negative + True Positive)**

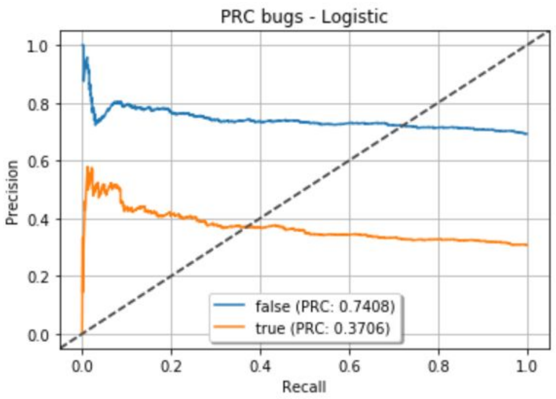
F Measure - F1 score is the weighted average of precision and recall - **2/((1/Recall) + (1/Precision))**

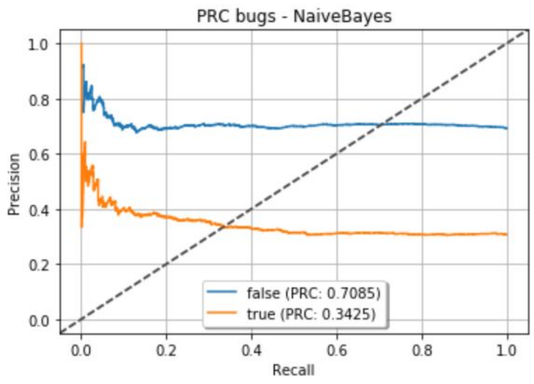
Experiment Results - all type of bugs - 2 Releases

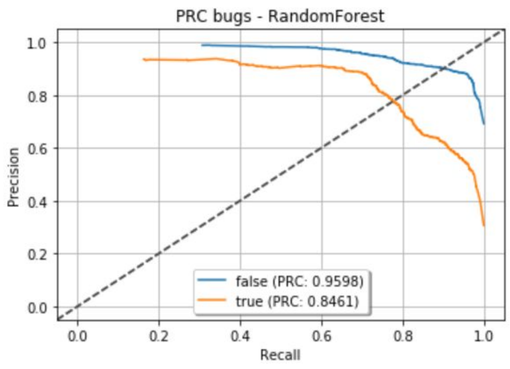


ML Algorithm	Precision	Recall	F Score
Naive Bayes	0.210525	0.025641	0.045714
Random Forest	0.238095	0.032051	0.056497
Logistic	NaN	0.0	NaN

Experiment Results - all type of bugs - 10 Releases







ML Algorithm	Precision	Recall	F Score
Logistic	0.024636	0.5	0.04695
Naive Bayes	0.81166	0.032051	0.056497
Random Forest	0.811684	0.748040	0.77855

Results - Cloud Specific Bugs

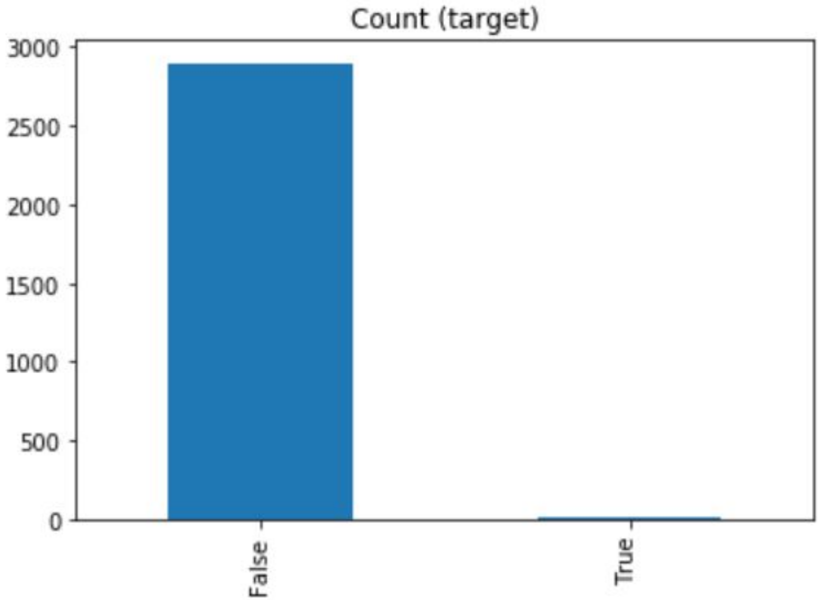
["distributed concurrency","performance","single-point-of-failure"]

Data Imbalance Technique -
imblearn.over_sampling.SMOTE

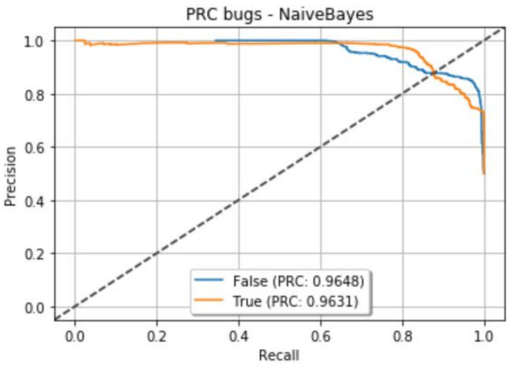
Class to perform over-sampling using SMOTE.

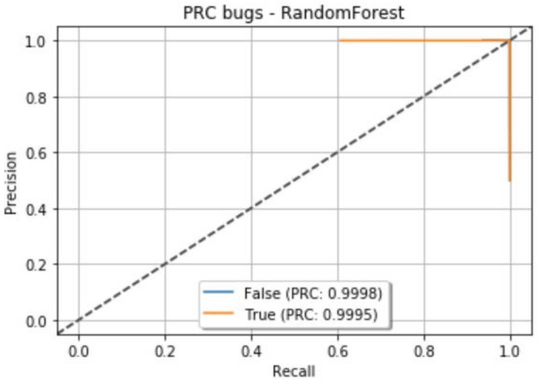
This object is an implementation of SMOTE - Synthetic Minority Over-sampling Technique

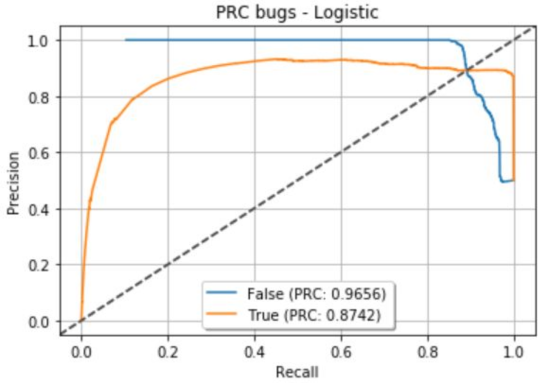
```
Bug "False": 2900
Bug "True": 6
Proportion: 483.33 : 1
```



Results - Cloud Specific Bugs







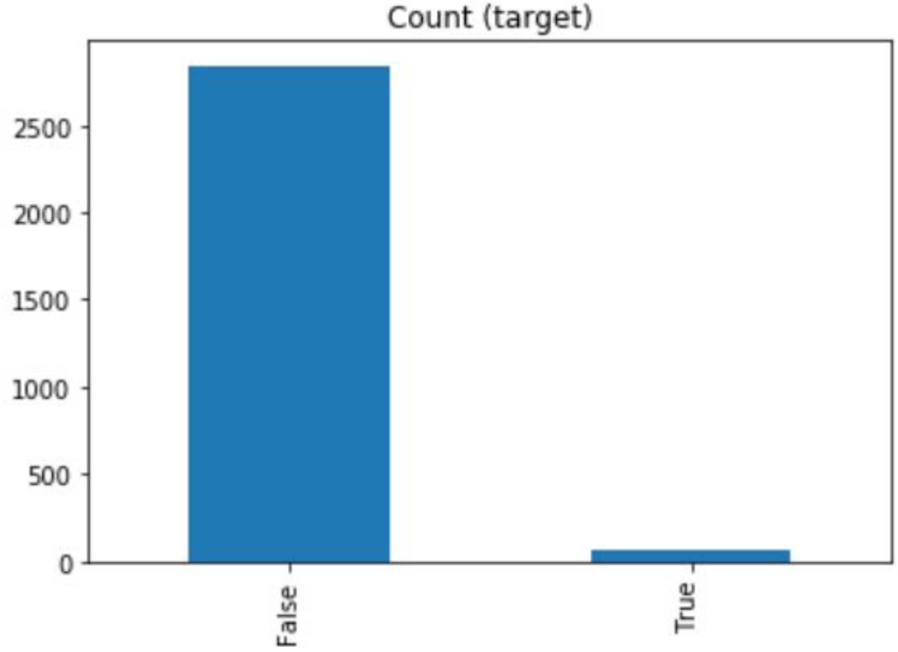
ML Algorithm	Precision	Recall	F Score
Logistic	0.88265	0.99344	0.93478
Naive Bayes	0.73876	0.99172	0.84675
Random Forest	0.99690	0.99965	0.99827

Results - Cloud Concurrency Bugs

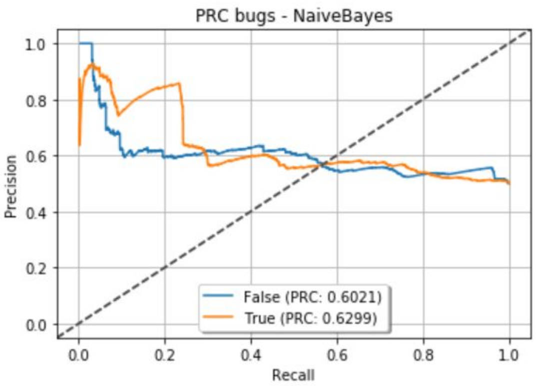
['blocked', 'locked', 'race', 'dead-lock', 'deadlock',, 'atomic',,,, 'starvation', 'suspension', 'order violation', 'atomicity violation', 'single variable atomicity violation', 'multi variable atomicity violation', 'livelock’, ‘live-lock']

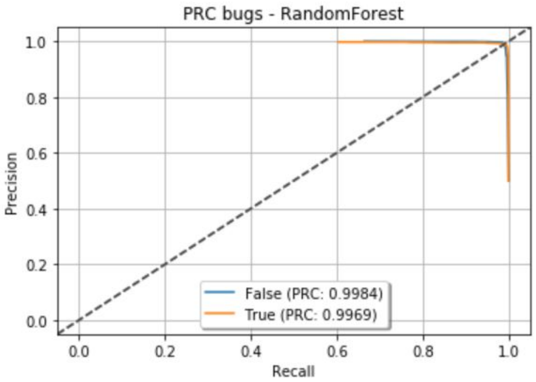
Dataset is imbalance. Cannot accept to giv better results. Need more analysis on

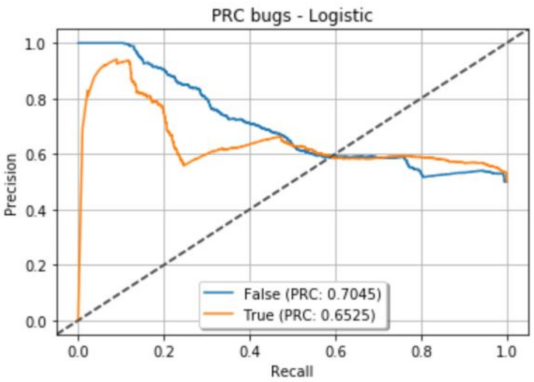
```
Bug "False": 2845
Bug "True": 61
Proportion: 46.64 : 1
```



Results - Cloud Concurrency Bugs







ML Algorithm	Precision	Recall	F Score
Logistic	0.64829	0.47557	0.54866
Naive Bayes	0.51747	0.90544	0.65857
Random Forest	0.98572	0.99507	0.99037

Limitations and Future Work

Experiment was carried out only for one cloud project "Hadoop" and in the future we plan to increase the sample project size and analyse the performance to get a better idea on the results.

In the current context, commit filtering was done using manually selected keywords and next step of the project is to use natural language processing help on commit filtering.

Complement to filtering bugs through keywords, from recent studies it is observed that analysing system log of the cloud environment would be a better choice to find defected modules but from higher architecture level except of class level.

RQ2: ASAT adoption

ASATs

Automatic static analysis tools (ASATs)

- Analysis of code without executing it
- Find defects, style violations, security issues, inefficiencies, …
- Automated: fast
- Low precision

Research Question

How do developers use ASATs for cloud applications?

- What kind of ASATs are used in cloud apps (compared to non-cloud apps)?
- How are ASATs configured in cloud apps?

Methodology

- ASAT collection
- ASAT classification
- Cloud and non-cloud projects sampling
- ASAT usage extraction in the projects

ASAT Collection

- ~ 50 active ASATs for Go
- Mostly highly specialized
- 2 linter aggregators

ASAT Classification

- 4 categories:
  - Readability
  - Efficiency
  - Correctness
  - Security

ASAT Classification

Readability

wsl (enforcing empty lines at the right places)

whitespace (unnecessary new lines)

nakedret (unused function arguments)

misspell (misspelled words)

lll (file line length)

gosimple (simplifying code)

golint (coding style issues)

gofmt (code formatting)

gocyclo (cyclomatic complexities)

goconst (Repeated strings that could be replaced by a constant)

flen (function length)

errcheck (error return values)

dupl (duplicated code)

Efficiency

varcheck (unused global variables and constants)

unused (unused constants, variables, functions and types)

unconvert (unnecessary type conversions)

structcheck (unused struct fields)

prealloc (slice declarations that could be preallocated)

nargs (unused function arguments)

ineffassign (ineffectual assignments)

deadcode (unused code)

Security

gosec (security problems)

safesql (SQL injections)

Correctness

govet (code correctness)

goimports (missing or unreferenced package imports)

Project Sampling

- GitHub’s search functionality
- Go projects
- >500 stars
- 15 cloud projects
  - query words: cloud, PaaS/SaaS
- 12 non-cloud projects (initially 15, but 3 did not use ASATs)

ASAT usage extraction

- Clone projects and scan files for ASAT commands
- Filtering:
  - comments, installation instructions, strings
  - Common words: ‘unused’, ‘misspell’
  - Text files (READMEs, txt files, html files)
- Parsing GolangCI configuration files
- Name, arguments, files

Results

Cloud projects

12

Non-cloud projects

5

ASATs on average per project

Results

Cloud projects

None: 1.3

Correctness: 1.7

Readability: 5.5

Efficiency: 3.7

Security: 0.2

Non-cloud projects

None: 0.5

Correctness: 0.7

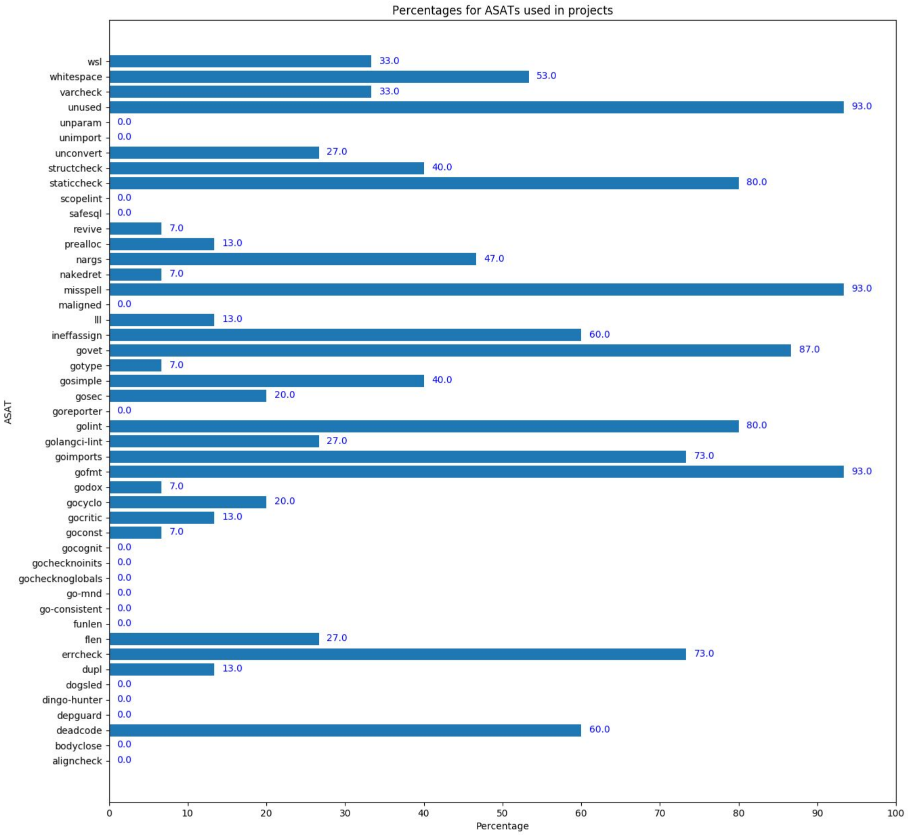
Readability: 2.3

Efficiency: 1.5

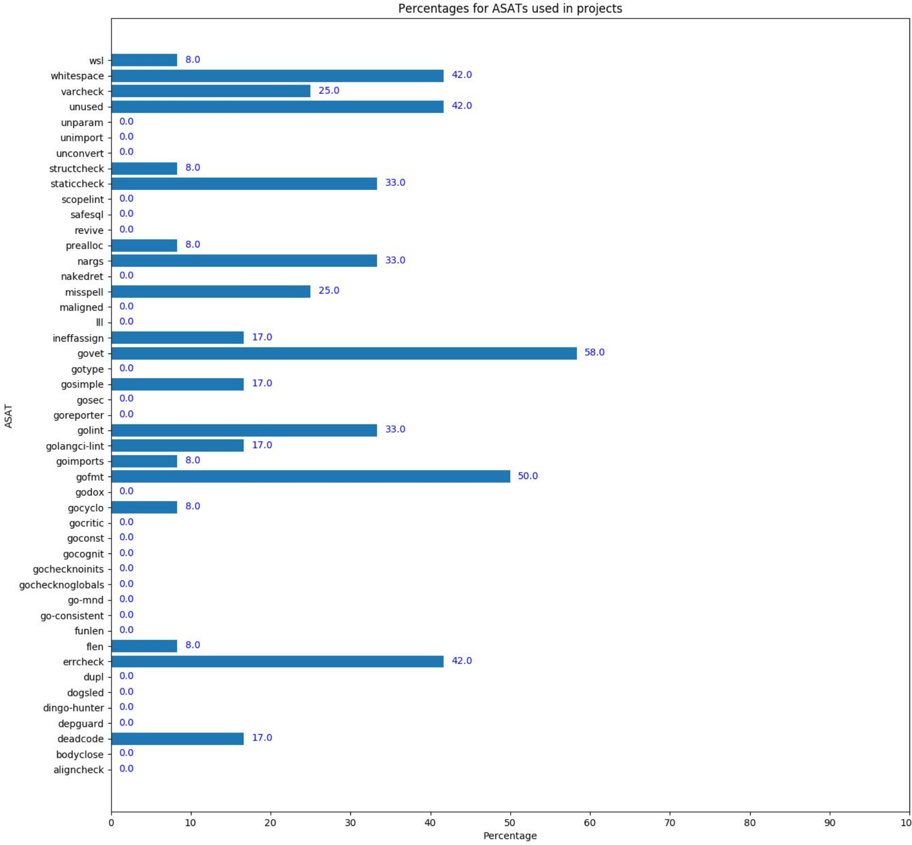
Security: 0.0

ASATs on average per project (by category)

Cloud projects



Non-cloud projects

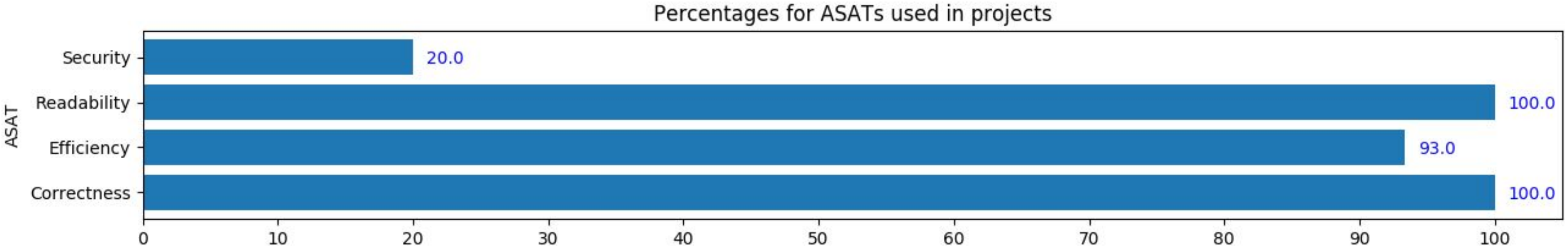


Percentage of projects using a specific ASAT

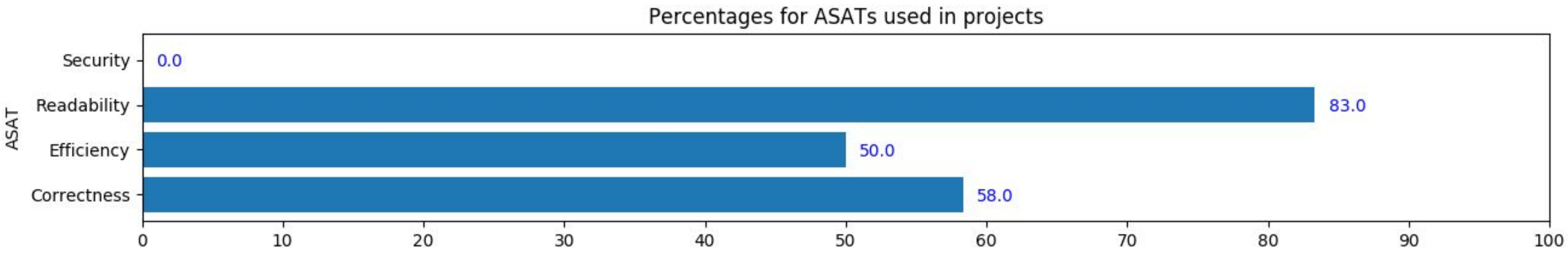
	Non-cloud vs. cloud
wsl (empty lines):	8 vs. 33 %
unused (unused code):	42 vs. 93 %
unconvert (unnecessary type conversions):	0 vs. 27 %
misspell (misspellings):	25 vs. 93 %
ineffassign (ineffectual assignments):	17 vs. 60 %
govet (code correctness):	58 vs. 87 %
gosimple (simplifying code):	17 vs. 40 %
gosec (security issues):	0 vs. 20 %
golint (coding style issues):	33 vs. 80 %
goimports (package import issues):	8 vs. 73 %
gofmt (formatting issues):	50 vs. 93 %
errcheck (error return values):	42 vs. 73 %
deadcode (unused code):	17 vs. 60 %

Percentage of projects using a ASAT category

Cloud projects



Non-cloud projects



ASAT configuration in cloud projects

ASAT: deadcode

ASAT: dupl

- Parameter: -files, 1/2

ASAT: errcheck

- Parameter: -ignorepkg, 3/11
- Parameter: -ignore, 2/11

ASAT: flen

ASAT: goconst

ASAT: gocritic

ASAT: gocyclo

- Parameter: -over, 1/3

ASAT: godox

ASAT: golint

- Parameter: -go..., 3/12
- Parameter: -vE, 5/12
- Parameter: -set_exit_status, 6/12
- Parameter: -min_confidence, 1/12
- Parameter: -v, 3/12
- Parameter: --invert-match, 1/12
- Parameter: -E, 1/12
- Parameter: -f, 1/12
- Parameter: -a, 1/12

ASAT: gofmt

- Parameter: -w, 12/14
- Parameter: -s, 12/14
- Parameter: -l, 10/14
- Parameter: -,,$@)/*.go, 1/14
- Parameter: -d, 8/14
- Parameter: -q, 1/14
- Parameter: -type, 3/14
- Parameter: -print, 1/14
- Parameter: -store/*, 1/14
- Parameter: -not, 1/14
- Parameter: -path, 2/14
- Parameter: ->, 2/14
- Parameter: -r, 2/14
- Parameter: -v, 2/14
- Parameter: -name, 3/14
- Parameter: -z, 1/14
- Parameter: -o, 1/14
- Parameter: -print));, 1/14
- Parameter: -prune, 1/14
- Parameter: -e, 2/14

ASAT: goimports

- Parameter: -w, 5/11
- Parameter: -vE, 4/11
- Parameter: -l, 6/11
- Parameter: --enable, 1/11
- Parameter: --vendor, 1/11
- Parameter: -d, 2/11
- Parameter: -eq, 1/11
- Parameter: -e, 1/11
- Parameter: -name, 1/11

ASAT: staticcheck

- Parameter: -go, 4/12
- Parameter: -ignore, 5/12
- Parameter: -ST1015, 4/12
- Parameter: -checks, 4/12

ASAT standard configuration

Cloud projects

65.8

Non-cloud projects

68.3

Percentage of ASATs for which projects use default configuration

Conclusion and Future Work

- Cloud projects
  - use more ASATs
  - use a larger variety of ASATs
  - typically use default configuration
- Future work:
  - Increase project sample
  - Configuration config files

Thank You

References

- [1] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella. Defect prediction as a multiobjective optimization problem. Softw.Test., Verif. Reliab., 25(4):426–459, 2015.
- [2] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In Proceedings of the ACM Symposium on Cloud Computing, SOCC '14, pages 7:1–7:14, New York, NY, USA, 2014. ACM.
- [3] H. Liu, X. Wang, G. Li, S. Lu, F. Ye, and C. Tian. Fcatch: Automatically detecting time-of-fault bugs in cloud systems. SIGPLAN Not., 53(2):419–431, Mar. 2018.

- [4] A. Panichella, C. V. Alexandru, S. Panichella, A. Bacchelli, and H. C. Gall. A search-based training algorithm for cost-aware defect prediction. In Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016, pages 1077–1084, 2016.
- [5] S. Panichella, V. Arnaoudova, M. D. Penta, and G. Antoniol. Would static analysis tools help developers with code reviews? In 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015, pages 161–170, 2015.
- [6] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, A. Zaidman, and H. C. Gall. Context is king: The developer perspective on the usage of static analysis tools.In 25th International Conference on Software Analysis, Evolution and Reengineering,SANER 2018, Campobasso, Italy, March 20-23, 2018, pages 38–49, 2018.

- [7] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In Proceedings of the 2013 international conference on software engineering, pages 712–721. IEEE Press, 2013.
- [8] C. Inc. Effective management of static analysis vulnerabilities and defects. 2009.
- [9] M. Di Penta, L. Cerulo, and L. Aversano. The life and death of statically detected vulnerabilities: An empirical study. Information & Software Technology, 51(10):1469–1484, 2009.
- [10] B. Johnson, Y. Song, E. R. Murphy-Hill, and R. W. Bowdidge. Why don’t software developers use static analysis tools to find bugs? In D. Notkin, B. H. C. Cheng, and K. Pohl, editors, 35th International Conference on Software Engineering, ICSE ’13, San Francisco, CA, USA, May 18-26, 2013, pages 672–681. IEEE Computer Society, 2013.

- [11] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella. Defect prediction as a multiobjective optimization problem. Softw. Test., Verif. Reliab., 25(4):426–459, 2015.
- [12] A. Panichella, C. V. Alexandru, S. Panichella, A. Bacchelli, and H. C. Gall. A search-based training algorithm for cost-aware defect prediction. In Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016, pages 1077–1084, 2016.