Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction

Di Mo, Robert Cordingly, Donald Chinn, Wes Lloyd
Presented by Robert Cordingly

School of Engineering and Technology
University of Washington Tacoma

14th IEEE International Conference on Cloud Computing Technology and Science

Outline

- Background and Motivation
- Research Questions
- Methodology
- Experimental Results
- Conclusions

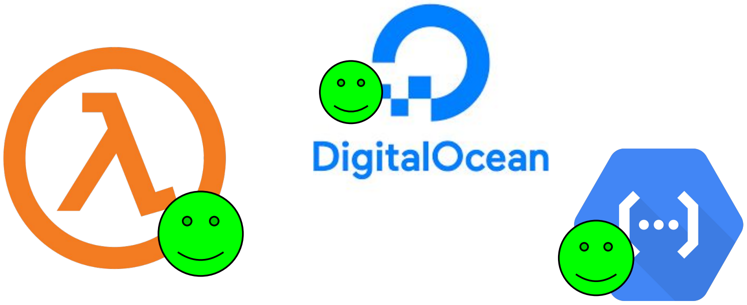
Why Serverless?

Serverless function-as-a-service (FaaS) platforms offer many desirable features:

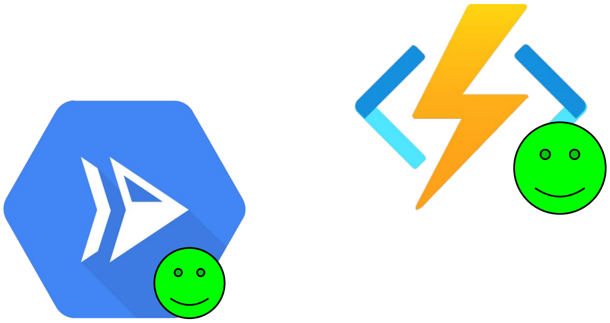
- Rapid elastic scaling
- Scale to zero
- No infrastructure management
- Fine grained billing
- Fault tolerance
- High availability

Vendor Lock-In

- FaaS platforms use vendor specific APIs and services that require code to be written specifically for one platform
- Migrating code to another platform may require significant refactoring
- Maintaining code supporting multiple platforms is challenging due to inconsistent feature sets and constantly changing services



Vendor Lock-In Solutions



- Cloud service abstraction libraries provide a common interface for multiple cloud providers
- Enabling portable code eases the challenge of migrating to different clouds
- In this study, we investigated the utility of a cloud service abstraction library in the context of FaaS

Apache Jclouds

- Open source multi-cloud toolkit for Java that aids in creating portable applications for multiple cloud providers
- Includes APIs for managed computer services (IaaS), blob storage, and load balancers (beta)
- Supports all major cloud providers such as AWS, GCP, Azure, Digital Ocean, and more...

Outline

- Background and Motivation
- Research Questions
- Methodology
- Experimental Results
- Conclusions

Research Questions

- RQ-1 **(Abstraction Overhead)**: What are the performance implications of using cloud service abstraction libraries to interface with object storage services in FaaS code?
- RQ-2 **(Code Quality)**: How do cloud service abstraction libraries impact FaaS code quality measured using static code analysis metrics?
- RQ-3 **(Portability)**: How do cloud service abstraction libraries impact the portability of FaaS code when migrating functions between cloud providers? What factors help predict successful code migration?

Outline

- Background and Motivation
- Research Questions
- ➤ Methodology
- Experimental Results
- Conclusions

Experiment 1a (RQ-1): Abstraction Library FaaS Performance

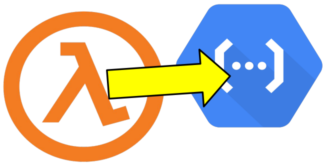
Function	Description
Transform_CSV	Reads and transforms CSV sales data
Read_File	Reads any file
Read_Key-value	Reads 1k key-value pairs
Write_Key-value	Writes 1k key-value pairs
Delete_Key-value	Deletes 1k key-values pairs
Create_Buckets	Creates 10 buckets
Delete_Buckets	Deletes 10 buckets

- Refactored 7 FaaS-native functions to use Apache Jclouds to access object storage on AWS and GCP
- Compared the performance of jclouds to the original functions
- Measured runtime (ms) and data read throughput (MB/sec)

Experiment 1b (RQ-2):
Abstraction Library FaaS Code Quality

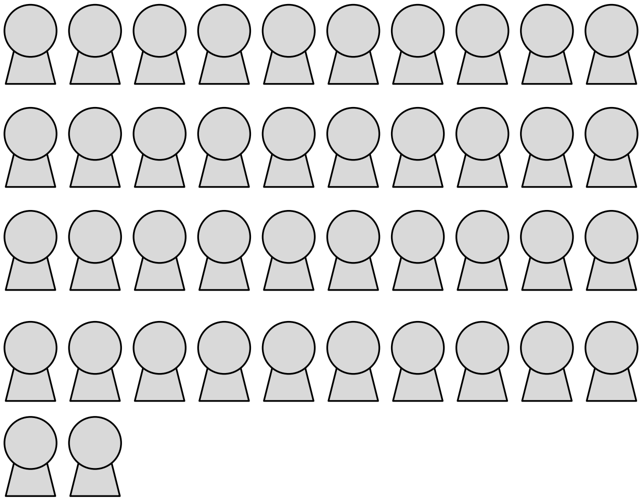
- Investigated code quality implications of using cloud abstraction libraries
- Used the static analysis tool JArchitect to compare three implementations of the Read_File function (AWS native, Google native, and Jclouds)
- Compared source code using Jar file size (MB), # of source files, # of third-party elements, LOC, Refactored LOC, and Average Cyclomatic Complexity (CC)

Experiment 2 (RQ-3):
Code Portability Empirical Study

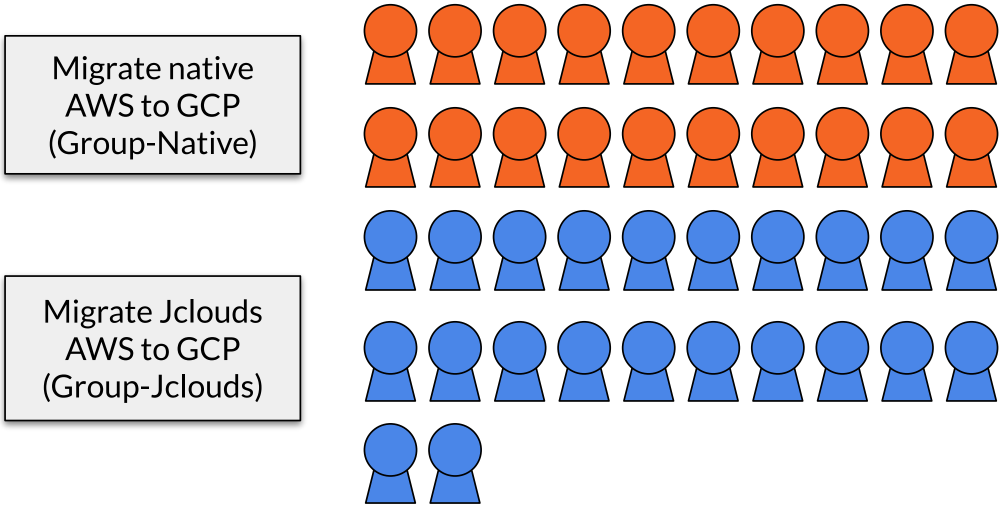


- Conducted empirical study using undergrad seniors and graduate cloud computing students to migrate an application from one FaaS platform to another
- Participants migrated a function to GCP, originally implemented natively for AWS or implemented with Apache Jclouds for object storage
- We had 42 participants and divided them into two groups…

Participant Demographics



Participant Demographics

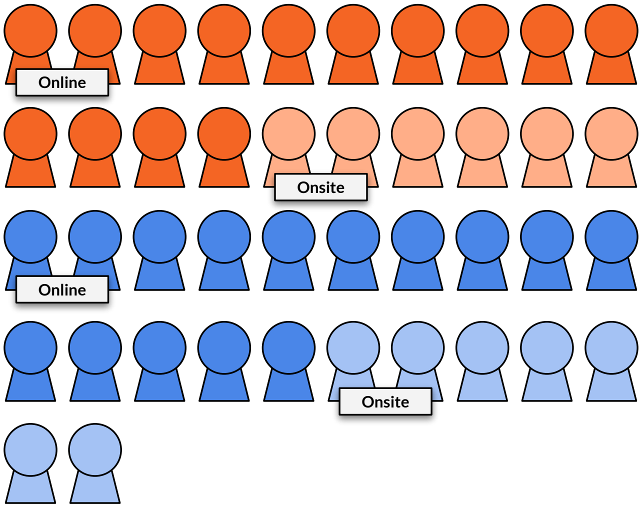


Migrate native AWS to GCP (Group-Native)

Migrate Jclouds AWS to GCP (Group-Jclouds)

Participant Demographics

Migrate native AWS to GCP (Group-Native)

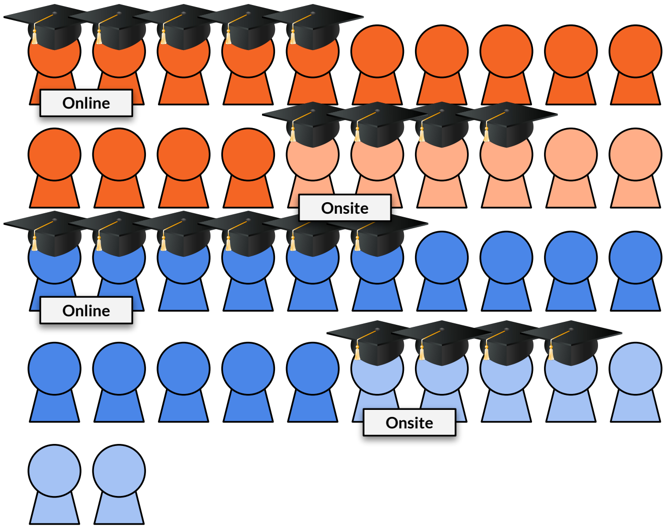


Each group was divided 2:1 online vs. onsite

Migrate Jclouds AWS to GCP (Group-Jclouds)

Participant Demographics

Migrate native AWS to GCP (Group-Native)



Each group was divided 2:1 online vs. onsite

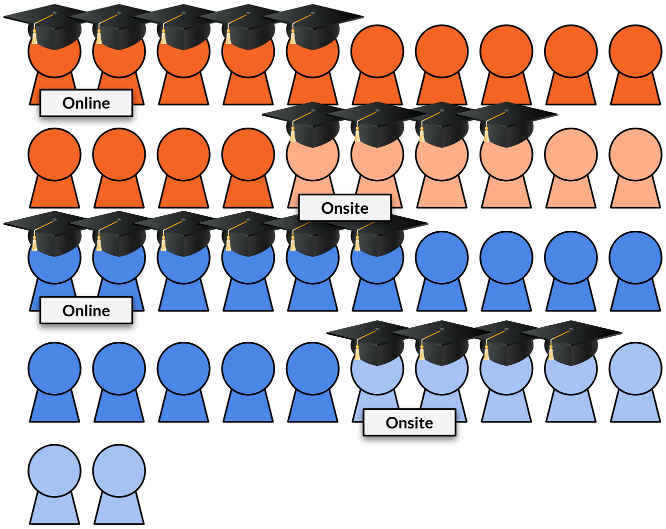
Migrate Jclouds AWS to GCP (Group-Jclouds)

Of each group the number of graduate vs undergrad students were balanced

Participant Demographics

Migrate native AWS to GCP (Group-Native)

Migrate Jclouds AWS to GCP (Group-Jclouds)

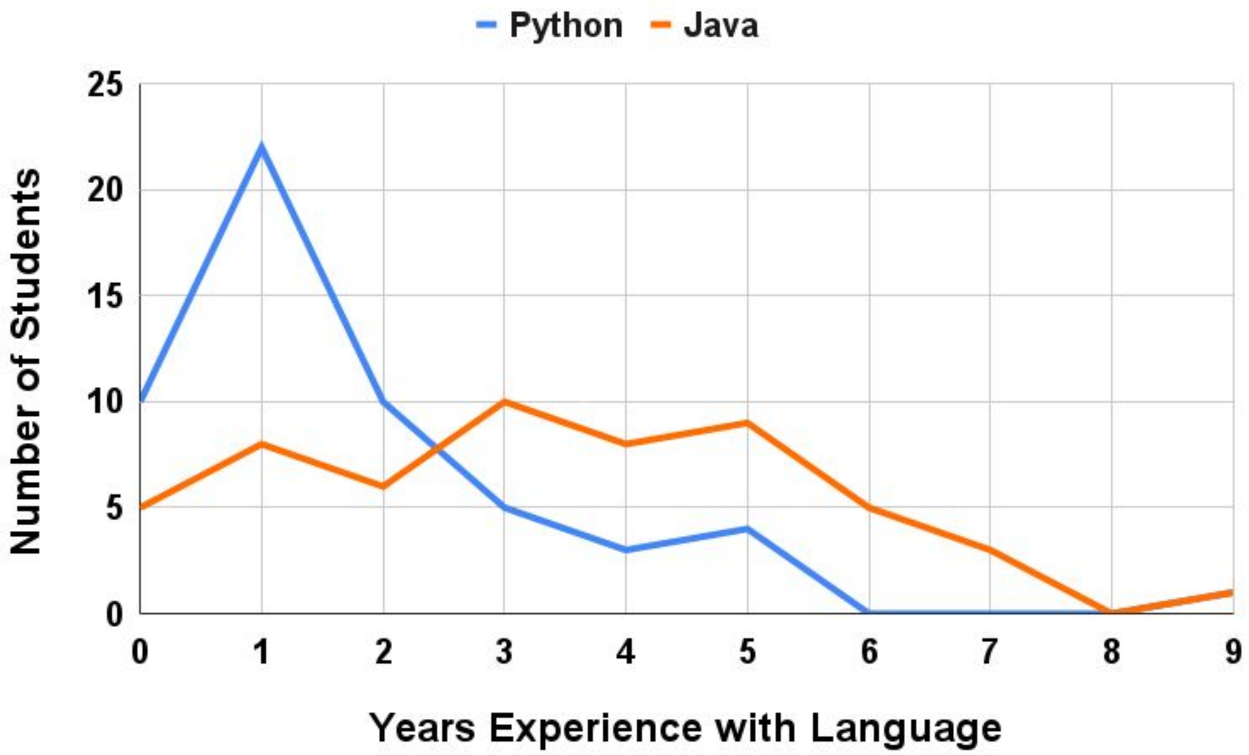


Each group was divided 2:1 online vs. onsite

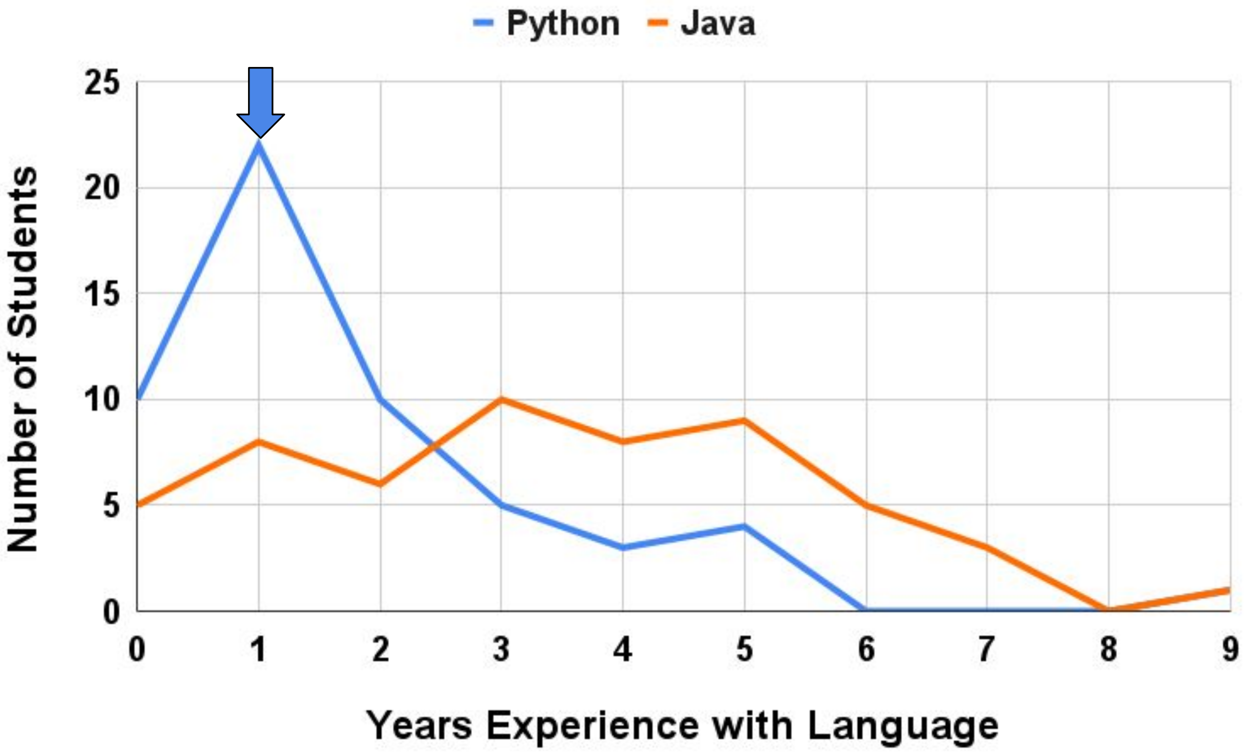
Of each group the number of graduate vs undergrad students were balanced

Participants were evaluated on whether they completed the function migration. Each participant completed pretrial and post-trial surveys

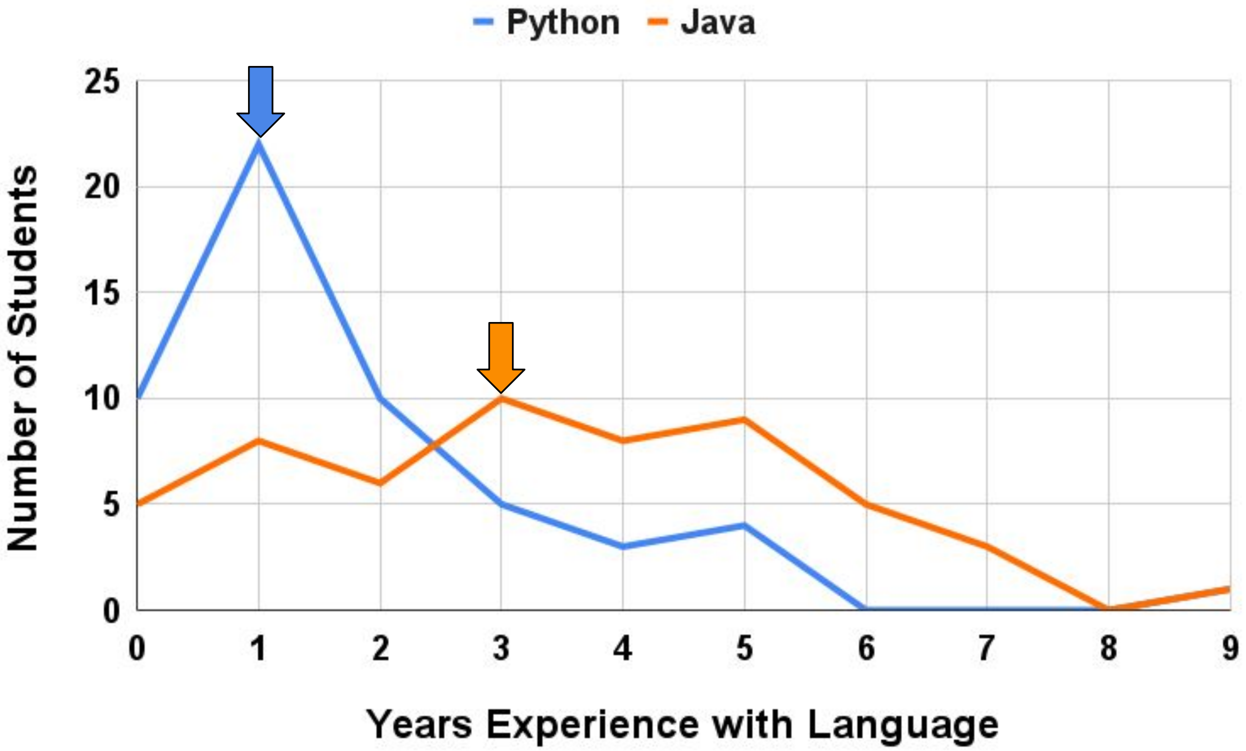
Demographics: Years of Experience per Language



Demographics: Years of Experience per Language



Demographics: Years of Experience per Language



Tasks

Each group had 4 hours to complete three activities:

Group-GCP

- Training Upload Object Task
  - 2 parameters
- Training Read Object Task
  - 6 lines of code
  - 1 method
- Code Migration Activity (Image Processing Function)
  - ~24 lines of code
  - 2 methods

Group-Jclouds

- Training Upload Object Task
  - 1 parameter
- Training Read Object Task
  - 7 lines of code
  - 1 method
- Code Migration Activity (Image Processing Function)
  - ~10 lines of code
  - 1 method

Outline

- Background and Motivation
- Research Questions
- Methodology
- ➤ Experimental Results
- Conclusions

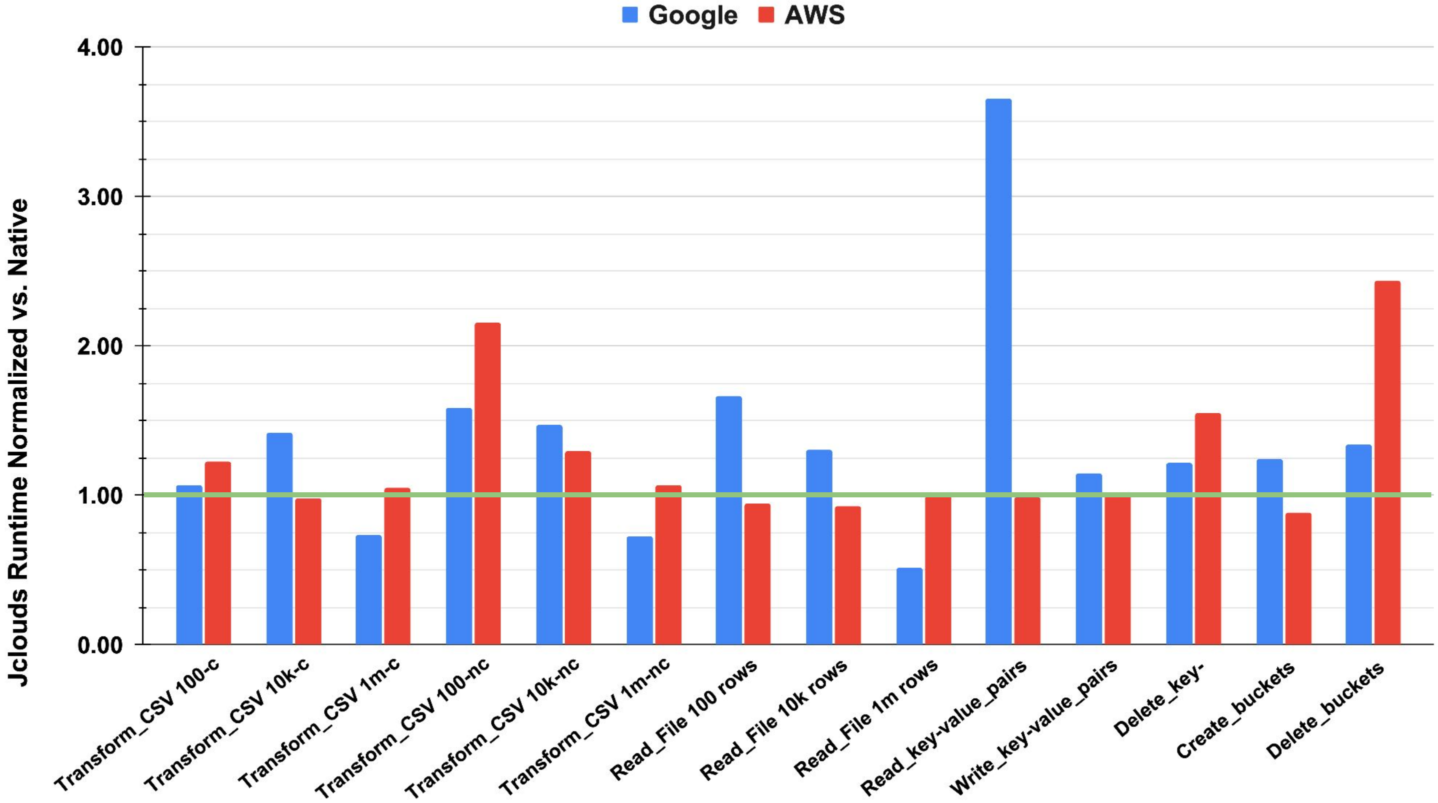
Experiment 1a

Abstraction Library FaaS Performance

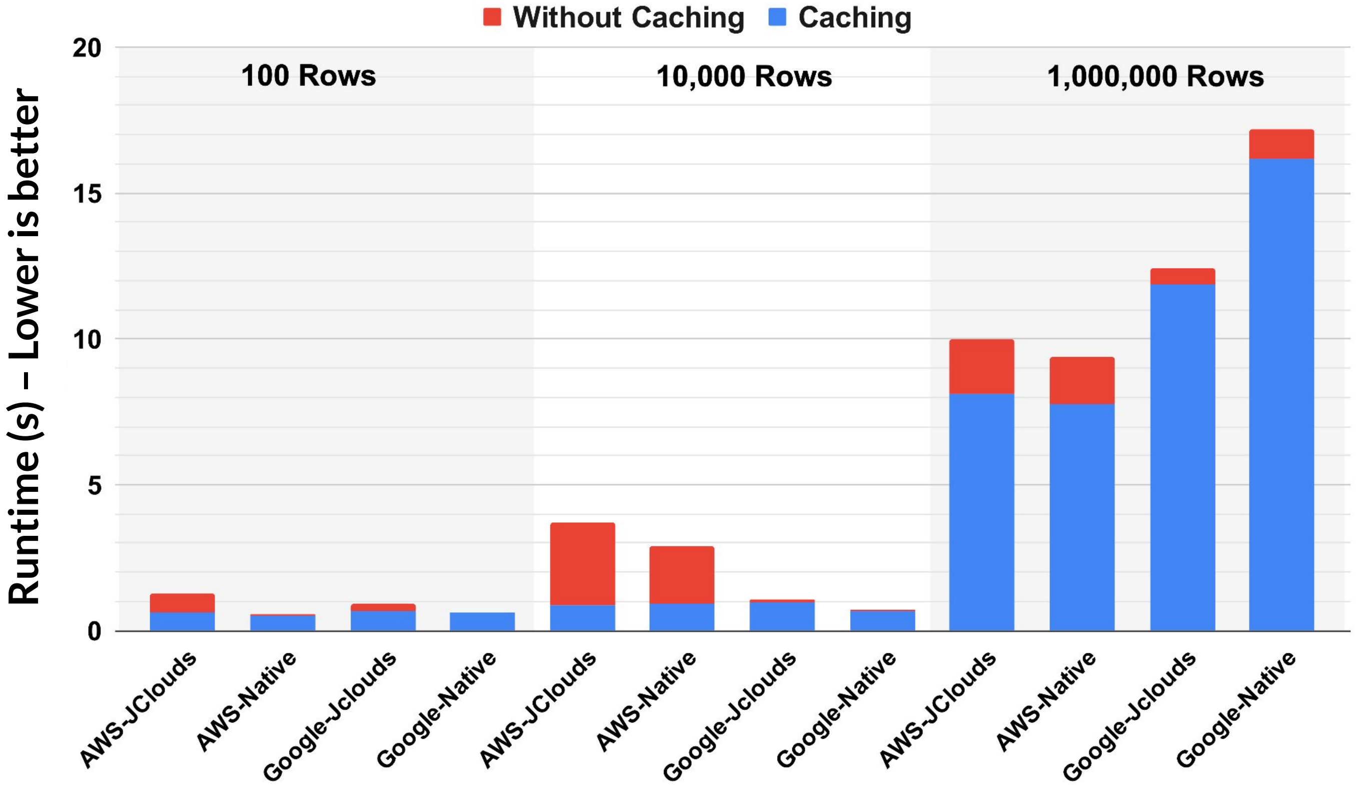
RQ-1: Abstraction Overhead

- Across all tests, functions using jclouds were 25% slower on AWS compared to native libraries for accessing object storage
- On Google, jclouds were 36% slower compared to native
- Jclouds performed better when reading and writing a large files vs transactional operations with many key-pairs or buckets

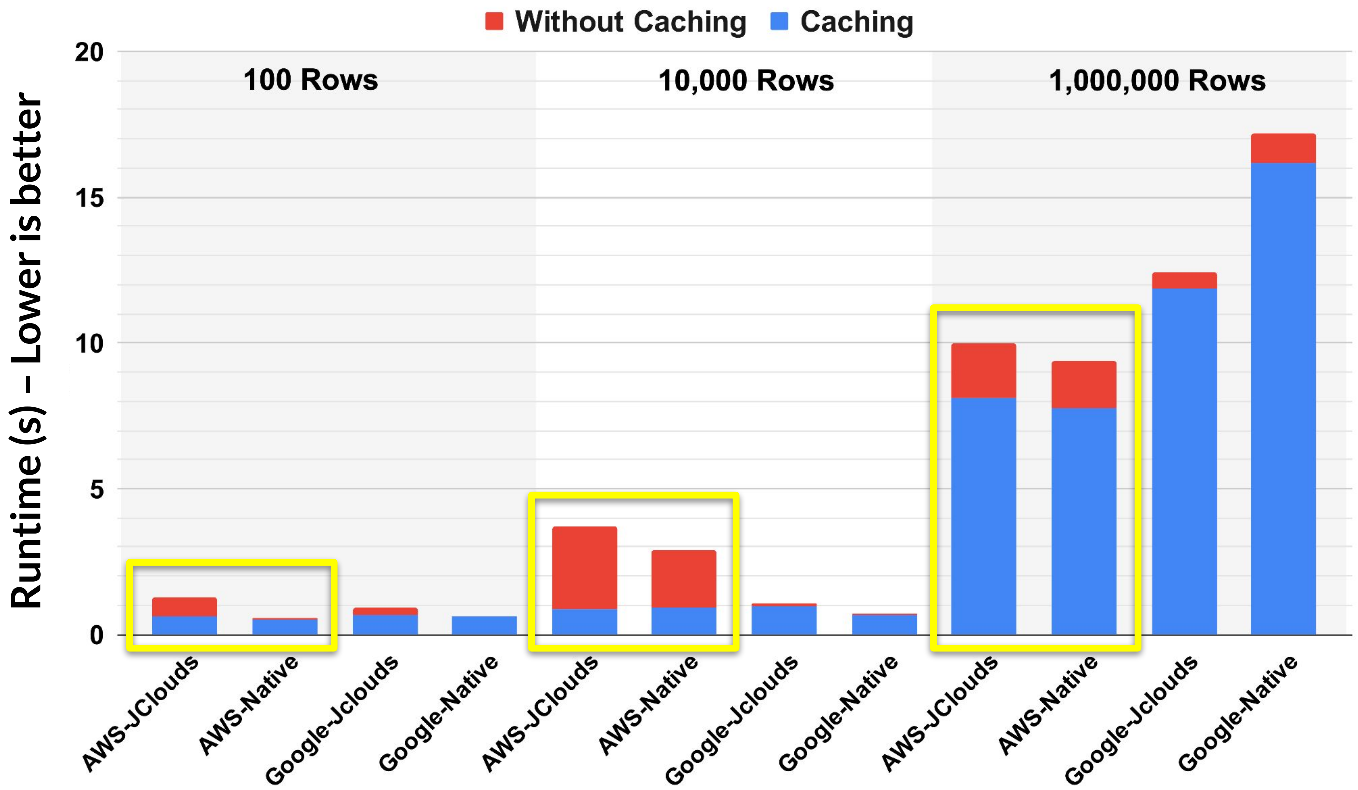
Function Runtime Comparison: jclouds vs native



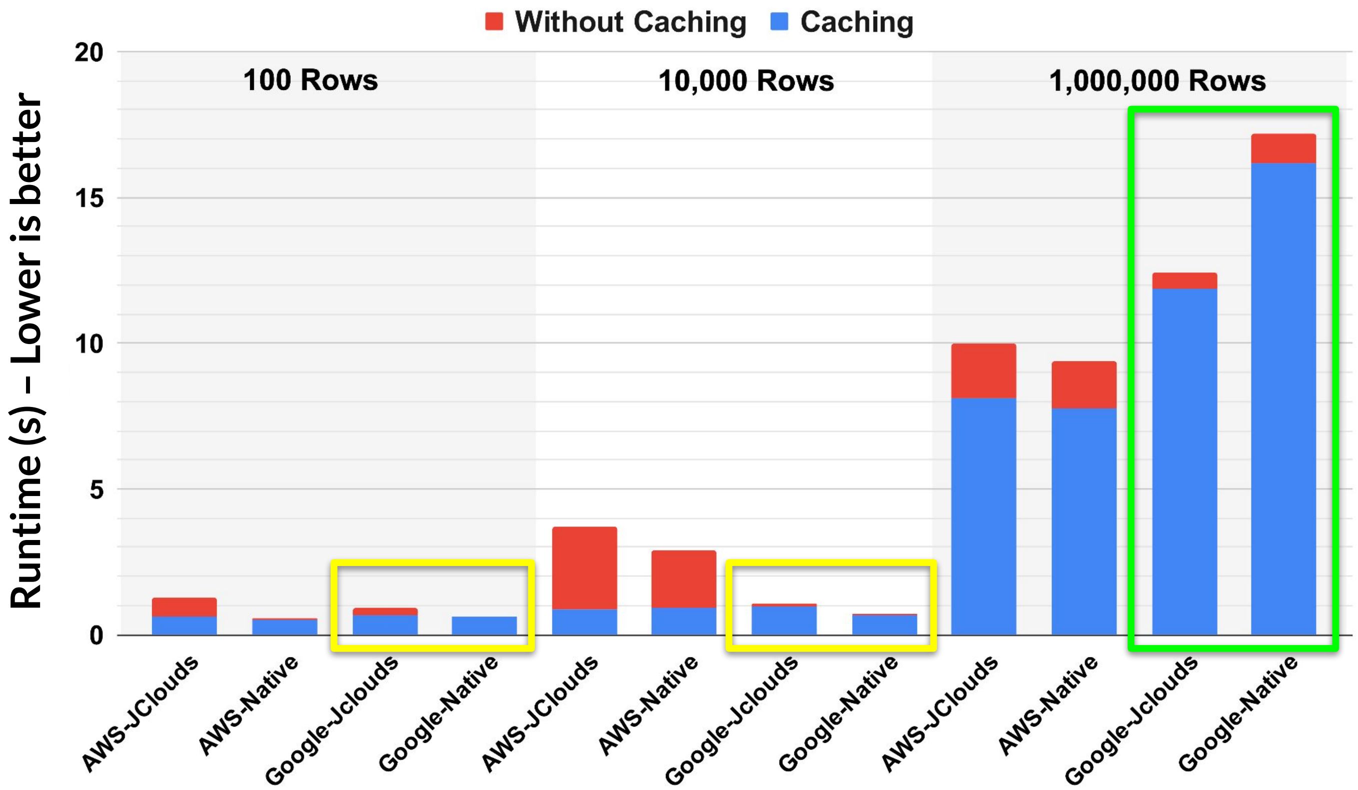
Transform_CSV Function Average Runtime



Transform_CSV Function Average Runtime



Transform_CSV Function Average Runtime



Experiment 1b

Abstraction Library FaaS Code Quality

	AWS Native	GCP Native	Jclouds
Jar File Size (MBs)	10	10.1	17.7
Source Files	7	7	7
Third-Party Elements	117	120	133
LOC	283	294	308
LOC Refactored	(N/A, baseline)	34	63
Average CC	2.66	2.65	2.56

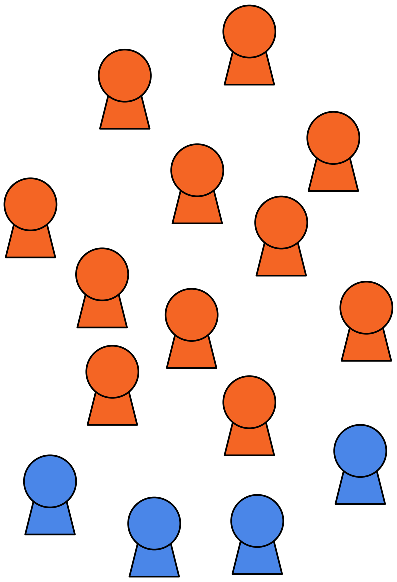
Refactored Code for Read_File – Quality Metrics

RQ-2: Abstraction Library Code Quality

- Migrating to jclouds involved nearly twice as many lines of code refactored (63 LOCR) compared to migrating to native GCP (34 LOCR)
- Jclouds exhibited slightly reduced code complexity (CC of 2.56) vs 2.66 on AWS and 2.65 on GCP

Experiment 2

Code Portability Empirical Study



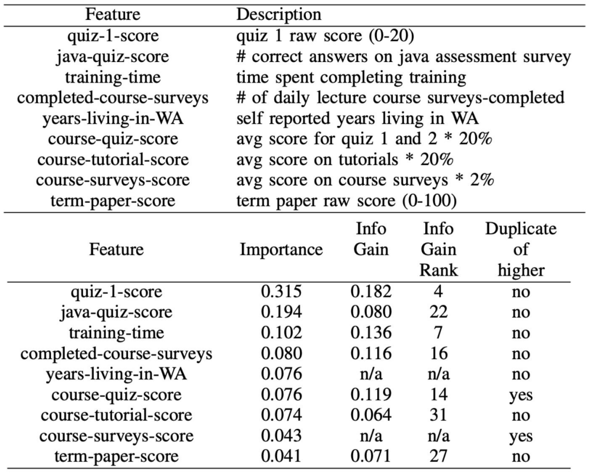
RQ-3: Code Portability

- Of the 42 participants, 15 were able to successfully migrate their function from AWS to GCP
- Of those 15, 11 successes were in Group-Jclouds while only 4 were in Group-Native
  - Using jclouds increased success of function migration by 30% (statistically significant – two proportion z-test: z=-2.0265, p=0.04236)
- Using Jclouds increased the average migration time by 14.3 minutes (from 93.3 mins to 104.6 mins)

—

Survey Results

Survey Feature Importance

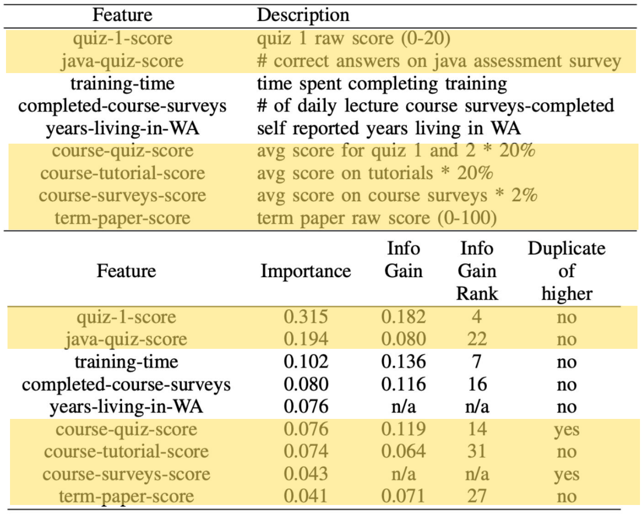


Feature	Description
quiz-1-score	quiz 1 raw score (0-20)
java-quiz-score	# correct answers on java assessment survey
training-time	time spent completing training
completed-course-surveys	# of daily lecture course surveys-completed
years-living-in-WA	self reported years living in WA
course-quiz-score	avg score for quiz 1 and 2 * 20%
course-tutorial-score	avg score on tutorials * 20%
course-surveys-score	avg score on course surveys * 2%
term-paper-score	term paper raw score (0-100)

Feature	Importance	Info Gain	Info Gain Rank	Duplicate of higher
quiz-1-score	0.315	0.182	4	no
java-quiz-score	0.194	0.080	22	no
training-time	0.102	0.136	7	no
completed-course-surveys	0.080	0.116	16	no
years-living-in-WA	0.076	n/a	n/a	no
course-quiz-score	0.076	0.119	14	yes
course-tutorial-score	0.074	0.064	31	no
course-surveys-score	0.043	n/a	n/a	yes
term-paper-score	0.041	0.071	27	no

- Utilized random forest modeling to analyze features that could most accurately predict successful outcomes
- With the survey and class graded we evaluated over 100 features to build our models
- We wanted to know what lead to successful outcomes

Experiment 2: Survey Results



Feature	Description
quiz-1-score	quiz 1 raw score (0-20)
java-quiz-score	# correct answers on java assessment survey
training-time	time spent completing training
completed-course-surveys	# of daily lecture course surveys-completed
years-living-in-WA	self reported years living in WA
course-quiz-score	avg score for quiz 1 and 2 * 20%
course-tutorial-score	avg score on tutorials * 20%
course-surveys-score	avg score on course surveys * 2%
term-paper-score	term paper raw score (0-100)

Feature	Importance	Info Gain	Info Gain Rank	Duplicate of higher
quiz-1-score	0.315	0.182	4	no
java-quiz-score	0.194	0.080	22	no
training-time	0.102	0.136	7	no
completed-course-surveys	0.080	0.116	16	no
years-living-in-WA	0.076	n/a	n/a	no
course-quiz-score	0.076	0.119	14	yes
course-tutorial-score	0.074	0.064	31	no
course-surveys-score	0.043	n/a	n/a	yes
term-paper-score	0.041	0.071	27	no

- A students quiz score was the most important feature for determining successful migration
- 6/9 features that contributed to successful migration where course grade components

Outline

- Background and Motivation
- Research Questions
- Methodology
- Experimental Results
- Conclusions

Conclusions

- **(RQ-1: Abstraction Overhead)** jclouds increased function runtime by 25% on AWS and 36% on GCP
- **(RQ-2: Code Quality)** jclouds increased overall code size by 8% and reduced cyclomatic complexity by 4%
- **(RQ-3: Portability)** jclouds improved serverless function migration outcomes by 30% with Java competency and course grades helped predict success

Thank You!