



跨语言运行时的垃圾回收算法评估方法和实证研究

Evaluating Garbage Collection Performance Across Managed Language Runtimes

汪钊丞, 窦文生, 梁埏, 王毅, 王伟, 魏峻, 黄涛

IEEE/ACM 47th International Conference on Software Engineering

联系方式: 汪钊丞, wangyicheng19@otcaix.iscas.ac.cn



论文地址

研究背景

托管语言运行时通过垃圾回收 (GC) 机制实现了自动内存管理

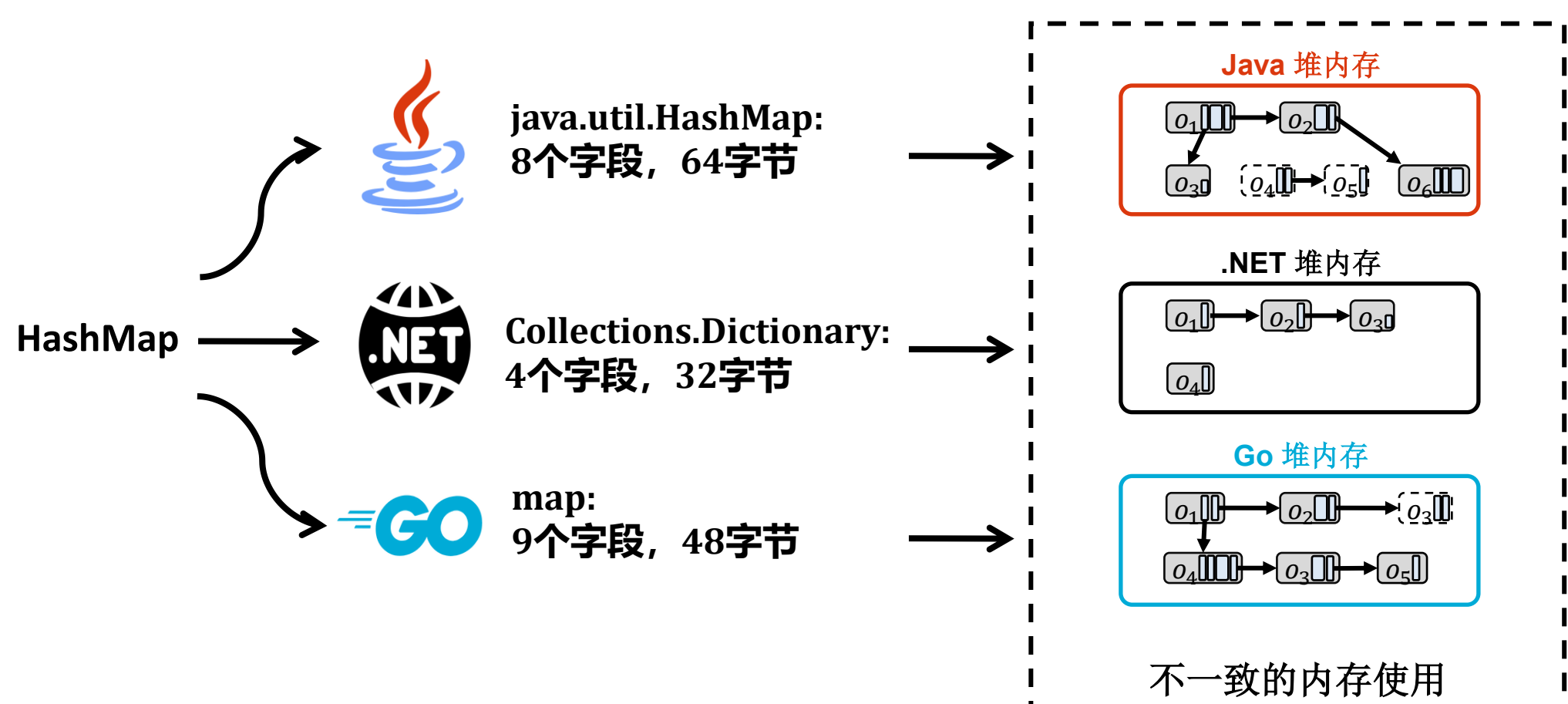
- 排名前10的编程语言, 7个使用GC机制
- 不同语言运行时的GC机制采用了不同策略
- GC实现的性能影响应用的执行效率

GC算法	压缩策略	分代策略	并发策略
Java ZGC	移动整理	非分代	全并发
Go GC	不移动	非区域分代	部分并发
C# Server GC	移动整理	分代	非并发

相关工作

使用不同语言实现相同逻辑功能

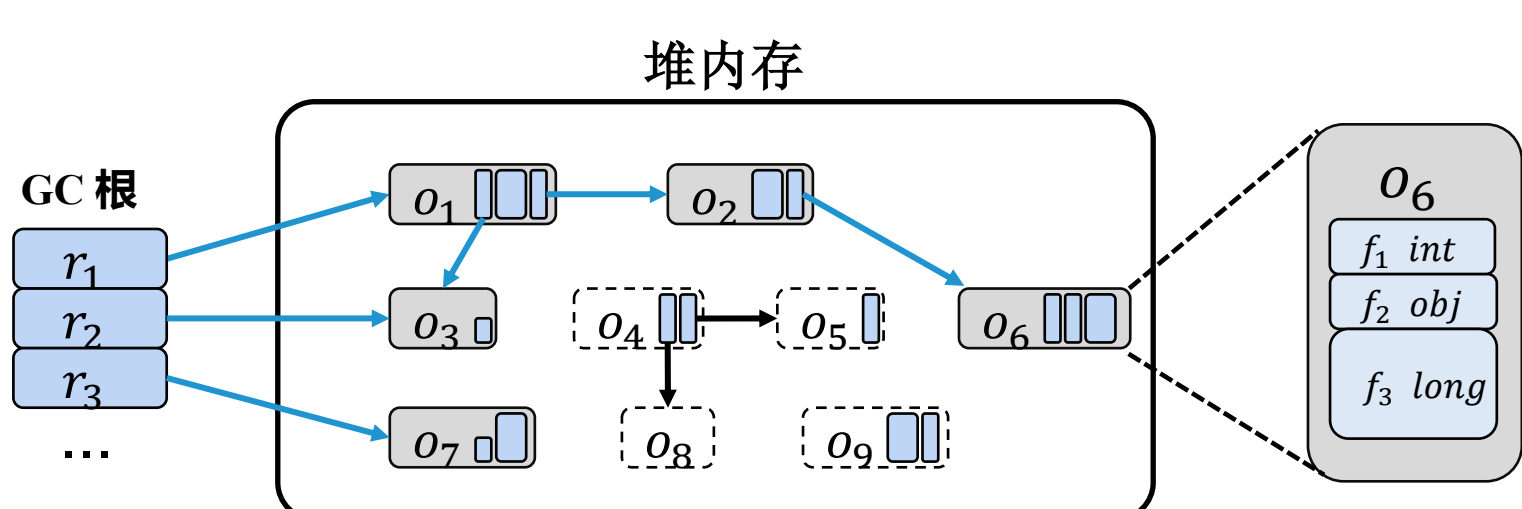
- 不能保证内存使用和GC负载一致
- 不能科学对比不同语言运行时的GC实现



研究方案

内存操作原语 (MOP) 设计

- GC影响因素: 对象大小、数量、引用关系



- MOP语句设计: 对象结构、创建、引用更新

目标语言测试程序生成

- 构建MOP到目标语言的映射, 转换MOP函数

基于真实内存操作记录生成MOP程序

- 插装主流Java运行时的源码, 在真实Java程序执行过程中记录其内存操作记录

```
public class manager {
    Object field;
    public static void main() {
        manager test = new manager();
        test.field = null;
        test.fruit();
    }
}

Add PR ID:0x04, C:[String, T:main, M:manager.main ([String])V
New ID:0x56, C:manager, T:main, M:manager.main ([String])V
Add PR ID:0x56, C:manager, T:main, M:manager.<init> ()V
Return ID:NULL, C:NULL, T:main, M:manager.<init> ()V
Add LR ID:0x56, C:manager, T:main, M:manager.main ([String])V
Put ID:0x56, C1:manager, ID2:null, C2:null, f:field, T:main,
M:manager.main ([String])V
Add PR ID:0x56, C:manager, T:main, M:manager.fruit ()V
...
Return ID:NULL, C:NULL, T:main, M:manager.fruit ()V
```

Java 应用

内存操作记录

- 内存操作记录过滤转换为MOP语句, 合并相同处理流程的方法, 重构循环操作结构

```
Add PR ID:0x04, C:[String, T:main, M:manager.main ([String])V
New ID:0x56, C:manager, T:main, M:manager.main ([String])V
Add PR ID:0x56, C:manager, T:main, M:manager.<init> ()V
Return ID:NULL, C:NULL, T:main, M:manager.<init> ()V
Add LR ID:0x56, C:manager, T:main, M:manager.main ([String])V
Put ID:0x56, C1:manager, ID2:null, C2:null, f:field, T:main,
M:manager.main ([String])V
Add PR ID:0x56, C:manager, T:main, M:manager.fruit ()V
...
Return ID:NULL, C:NULL, T:main, M:manager.fruit ()V

void manager_fruit_v0(manager manager_0) {
    ArrayList arraylist_0 = NewInstance(ArrayList);
    ArrayList_init_v0(arraylist_0);
    Loop(i: 0, 2, 3){
        String string_0 = NewInstance(String);
        root[i] = string_0;
        ArrayList_add_v0(arraylist_0, i, string_0);
    }
}
```

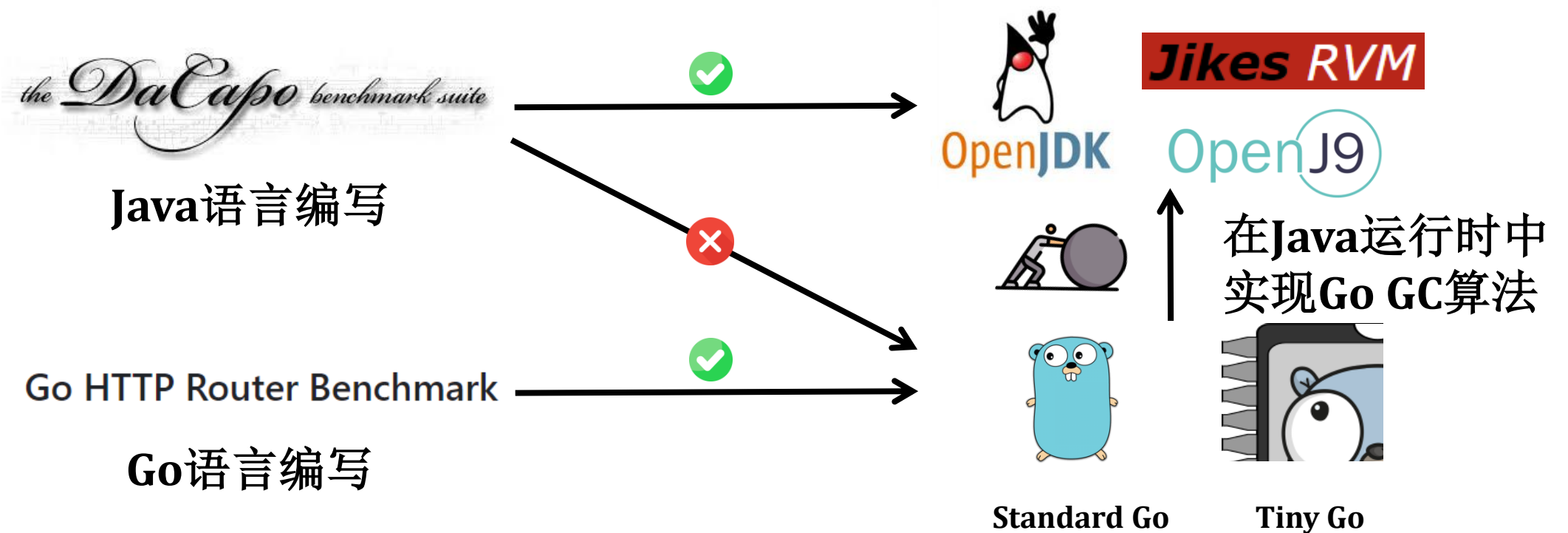
内存操作记录

简化后的MOP程序

问题描述

缺少对比不同语言运行时中GC实现的评估方法

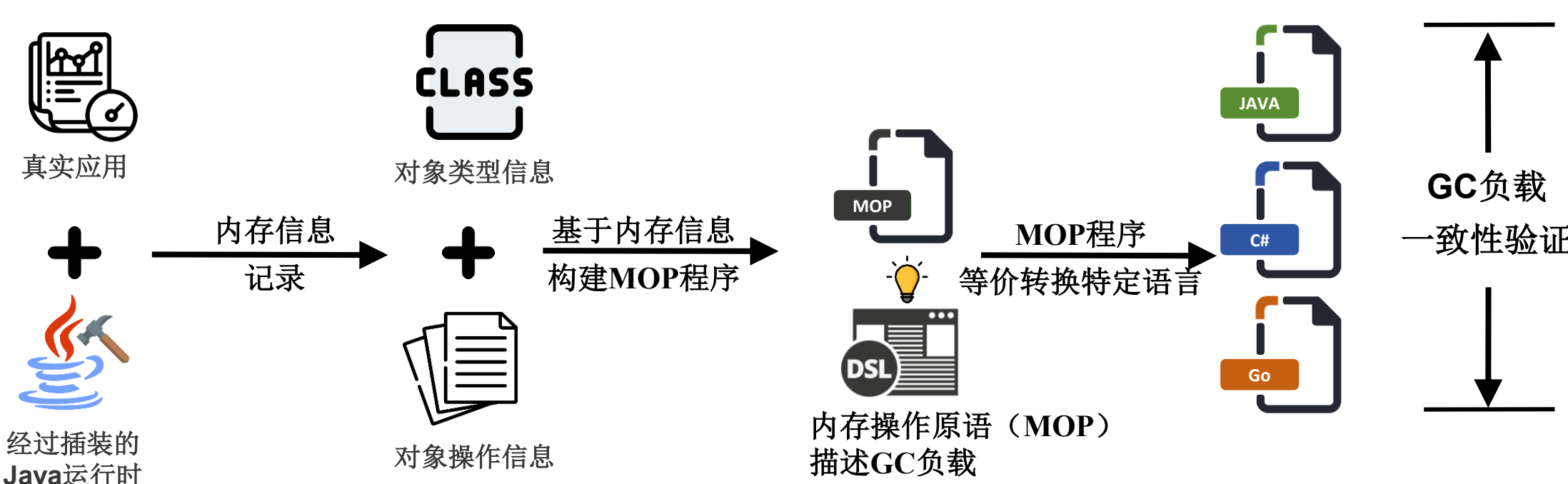
- 现有测试套件用特定语言编写
- 迁移GC实现到其他语言运行时的难度大



研究思路

GEAR: 跨语言运行时的GC评估方法

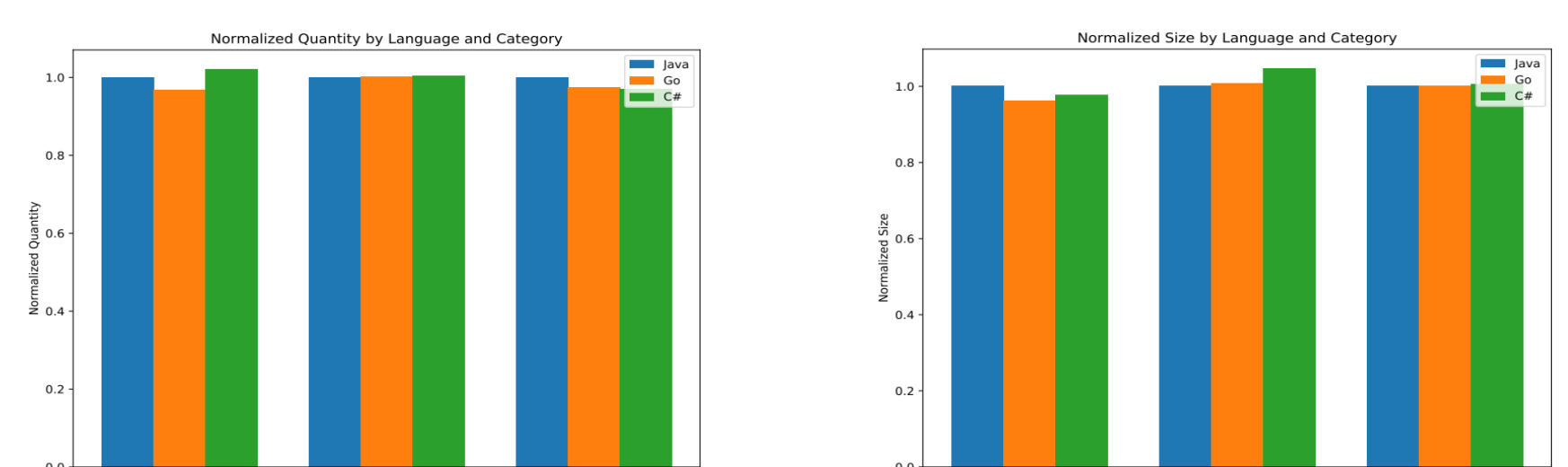
- 记录真实应用的内存操作记录
- 转换为语言运行时无关的MOP程序
- 等价转换为特定语言运行时程序



实验结果

GEAR的一致负载构建验证

- 人工编写MOP程序构建Java、Go、C#程序
- 运行时之间的对象数量和大小差异小于5%



对象数量

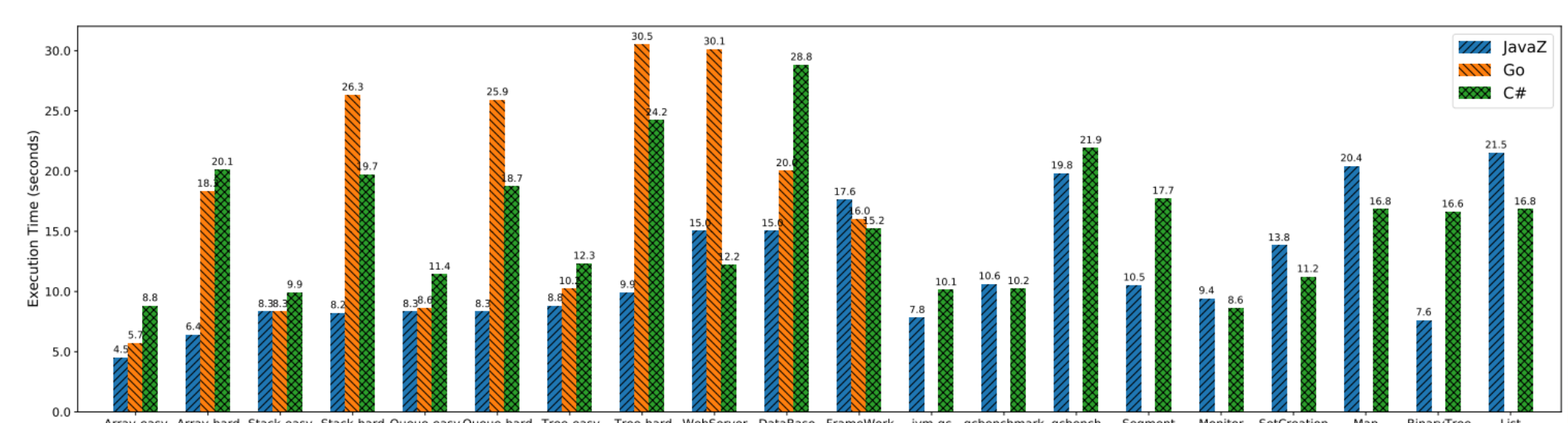
对象大小

GEAR真实GC负载构建验证

- 使用20个采样自GitHub、Guava和Rosetta的真实Java应用程序构建MOP程序
- 原始Java程序与MOP程序生成Java程序的最大对象统计差异为4.8%

跨语言运行时GC性能对比实证研究

- Java、Go、C#的代表性GC实现对比
- 3个代表性内存使用模式、9个真实GC负载、4个代表性对象引用关系的MOP程序



- 不同程序和配置下, GC算法之间的表现差异明显, 总结出6个GC性能相关的发现