Google
持续集成和构建系统简介

叶航军

日程

- Google的开发模式，现状和挑战
- 构建系统简介
- Build in Cloud
- 持续集成系统

现状（2010年）

- 规模庞大，开发活跃
  - 单一的代码库，多种编程语言
  - 超过1亿行源代码，12万个测试文件
  - 6000+工程师，40+研发中心
  - 1500+活跃开发的项目
  - 日均6.5万次构建，5亿次构建动作 (action)
  - 日均运行750万个测试文件，1.2亿个测试用例
  - 平均每分钟20+代码提交
  - 50%的代码每月至少有一次修改和提交

开发模式

- head/trunk上开发
  - 代码提交对相关项目立即可见
  - 敏捷，没有分支合并问题
  - 容易构建/测试失败

- 源代码依赖、构建和发布
  - 确定性，简单
  - 减少兼容性问题和冲突
  - 要访问和构建所有相关代码，耗时

挑战

- 规模和开发模式的冲突
  - 构建缓慢
  - 错误扩散
- 期望
  - 快速发现，精确定位有缺陷的代码提交，持续集成
- 效率！
  - 花多少时间checkout/sync代码？
  - 等待构建和测试结果？
  - 定位构建和测试问题？

解决方案

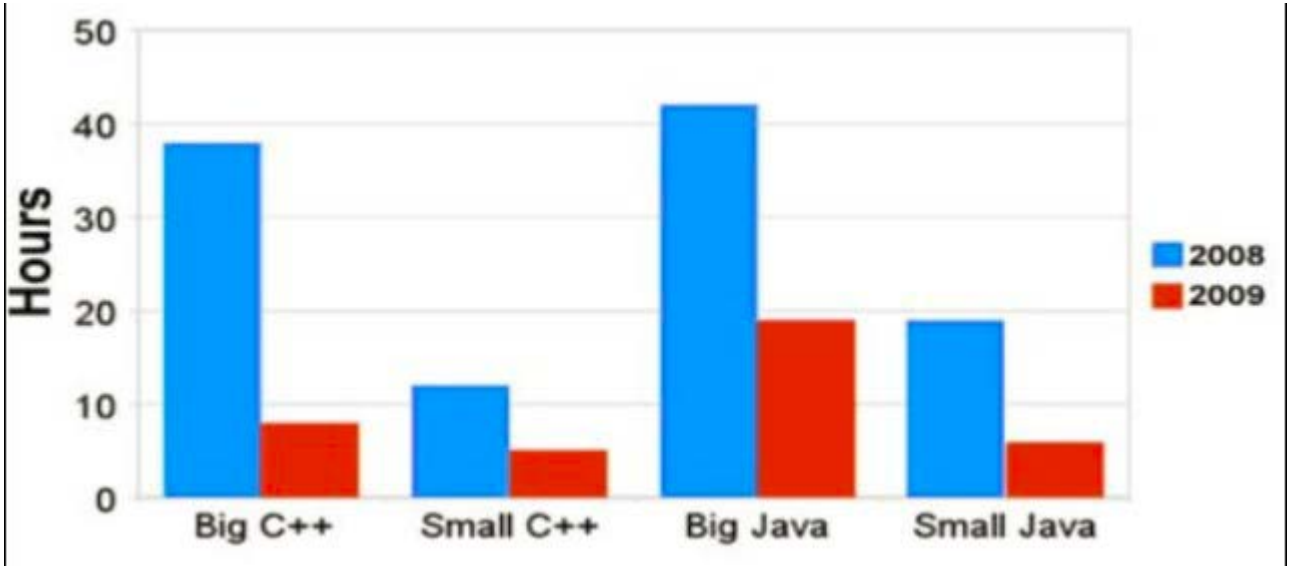
- 强大的构建系统
  - 工程师/开发者
  - 自动构建/持续集成

工程师

- 每月构建相关的活动

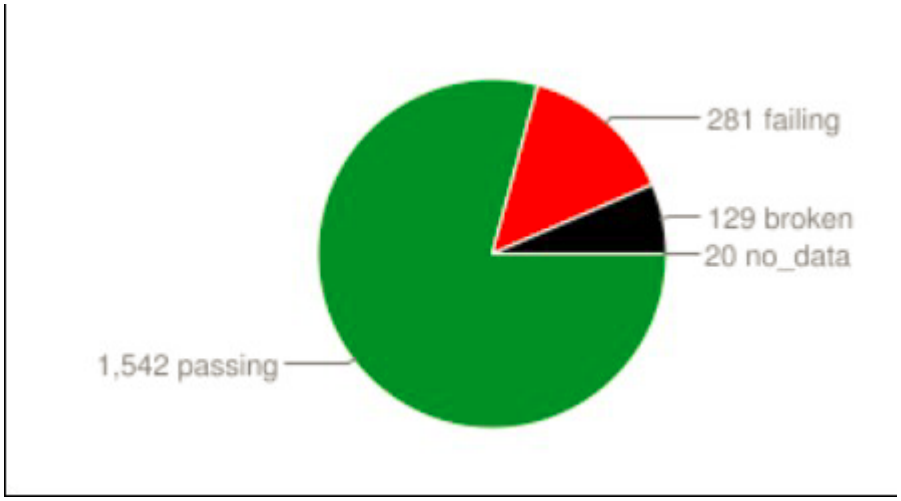
ACTIVITY	INITIAL CHECK-OUT	CLEAN BUILD	BUILD AFTER EDIT	BUILD AFTER SYNC	RUN TESTS
FREQUENCY	2	4	160	20	60

- 等待时间



自动构建

- 自动，快速，精确定位有缺陷的代码提交
  - 每次代码提交后运行所有（受影响）的测试
  - 2/3的构建动作 (action) 来自动构建
  - 1800+完整的持续集成构建（回归测试）



具体方案

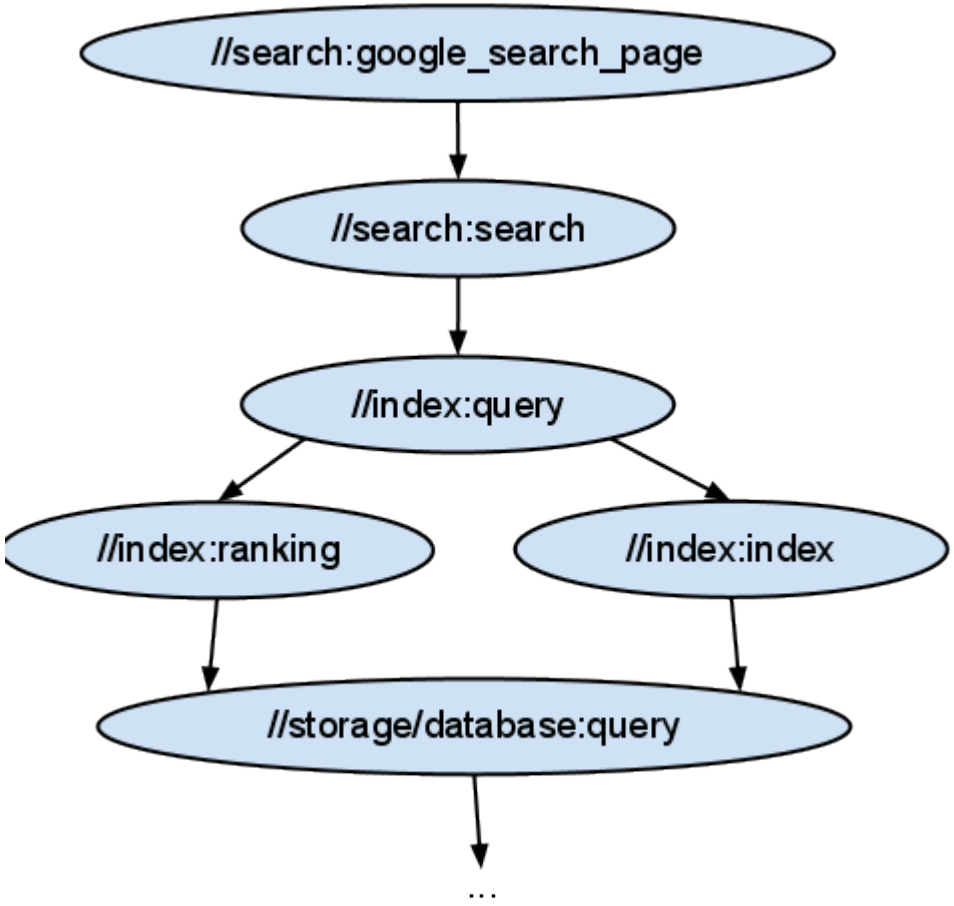
- 构建描述
- 依赖分析
- 增量构建
- Build in Cloud

构建描述 (rules)

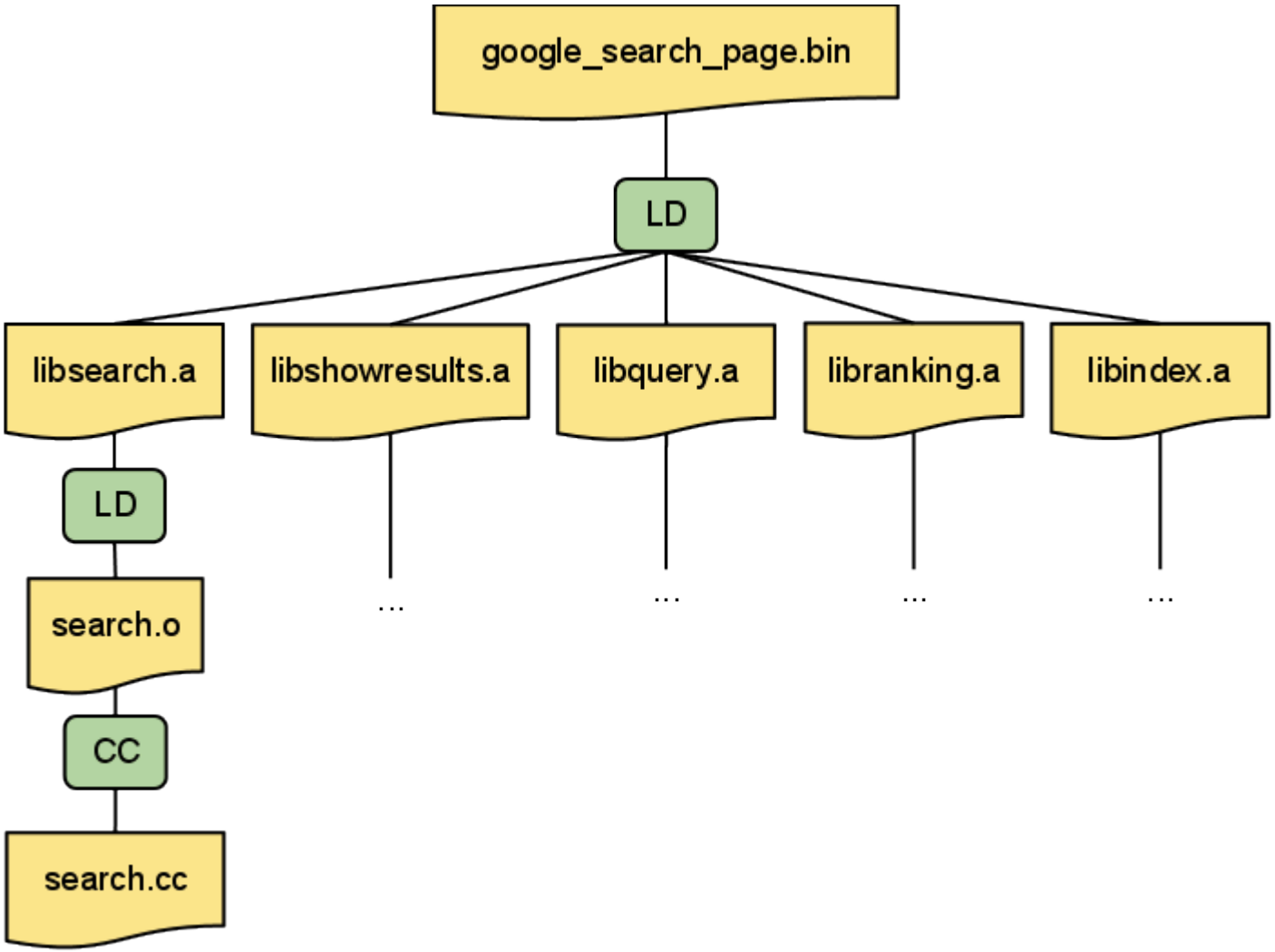
- /search/BUILD:

```
cc_binary(name = ‘google_search_page’,
          deps = [ ‘:search’,
                   ‘:show_results’])

cc_library(name = ‘search’,
          srcs = [ ‘search.h’,‘search.cc’],
          deps = [‘//index:query’])

/index/BUILD:
cc_library(name = ‘query’,
          srcs = [ ‘query.h’, ‘query.cc’, ‘query_util.cc’],
          deps = [‘:ranking’,
                   ‘:index’])

cc_library(name = ‘ranking’,
          srcs = [‘ranking.h’, ‘ranking.cc’],
          deps = [‘:index’,
                   ‘//storage/database:query’])

cc_library(name = ‘index’,
          srcs = [‘index.h’, ‘index.cc’],
          deps = [‘//storage/database:query’])
```

依赖关系 - DAG


//search:google_search_page
//search:search
//index:query
//index:ranking
//index:index
//storage/database:query
...

构建动作 (action)


google_search_page.bin
LD
libsearch.a
libshowresults.a
libquery.a
libranking.a
libindex.a
LD
search.o
CC
search.cc
...
...
...
...

Why reinvent the wheel?

- 确定性 (deterministic, hermetic)
  - 同样的输入和动作，一定要产生同样的输出
  - 各种优化才有可能
    - 增量构建，并行构建，缓存
- 和Makefile相比：
  - 构建描述分布到每个package
  - 每个rule描述的是输入文件和输出类型，输出文件和构建动作是隐含的（推导出来的）
  - 做什么，而不是怎么做（汇编和高级语言的区别）
  - Content-based, not timestamp-based

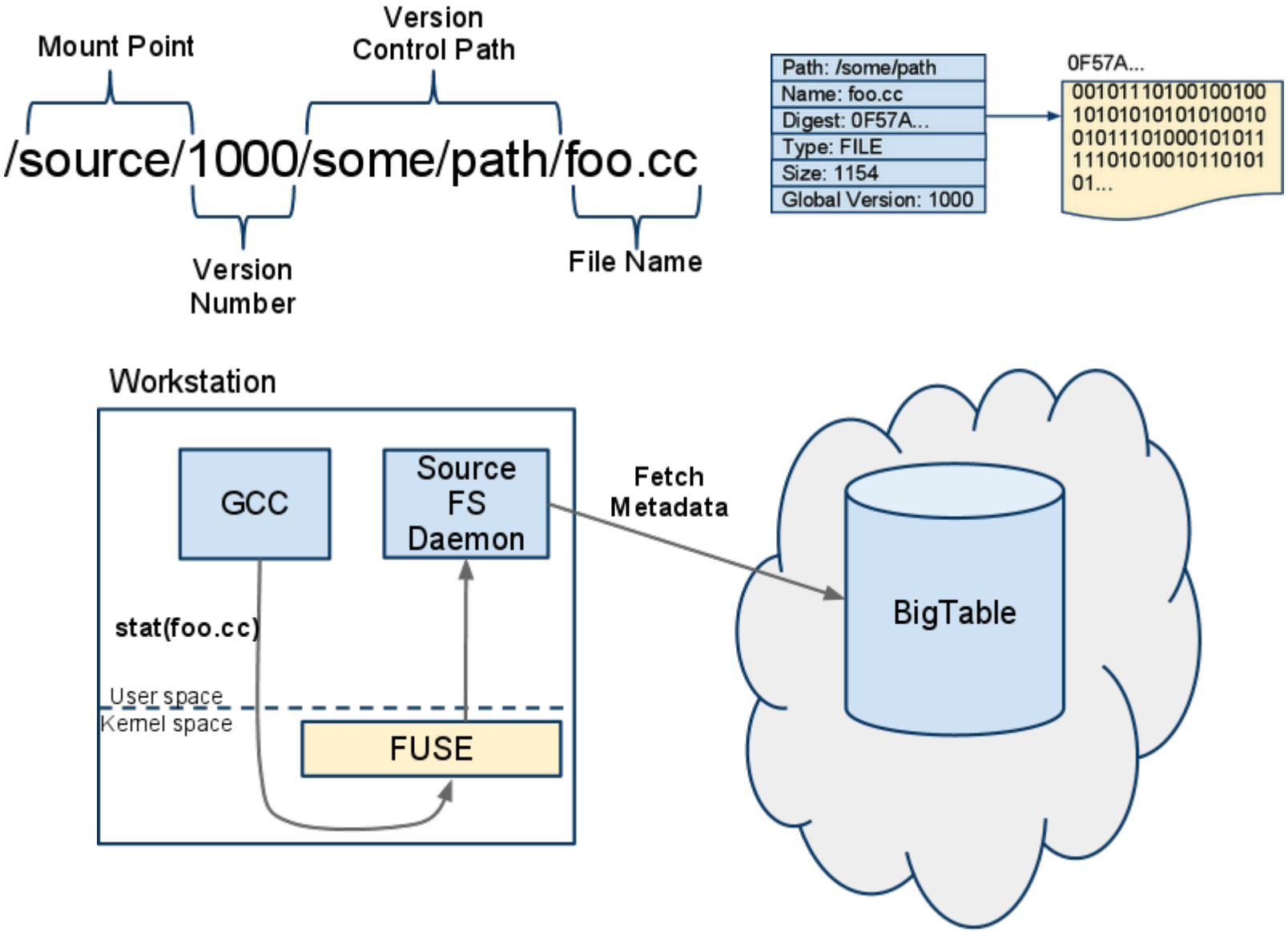
Build in Cloud

- 源文件文件系统
- 目标文件系统
- 并行构建

源文件系统

- checkout/sync代码比较耗时
- 需要checkout所有构建所需代码吗？
  - 只需要checkout自己要修改的代码！
  - 其它代码按需获取，本地缓存

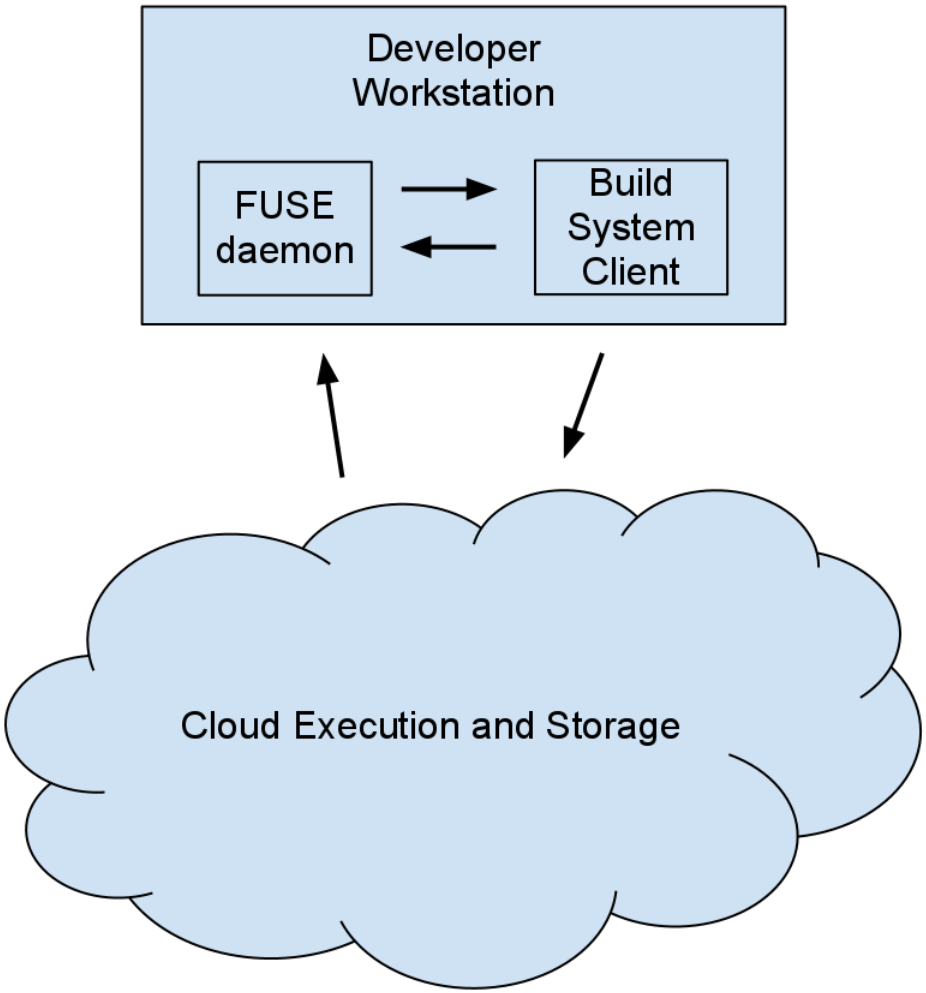
源文件系统 - FUSE


Mount Point
Version Control Path
/source/1000/some/path/foo.cc
Version Number
File Name
Path: /some/path
Name: foo.cc
Digest: 0F57A...
Type: FILE
Size: 1154
Global Version: 1000
0F57A...
00101110100100100
10101010101010010
01011101000101011
11101010010110101
01...
Workstation
GCC
Source FS Daemon
stat(foo.cc)
User space
Kernel space
FUSE
Fetch Metadata
BigTable

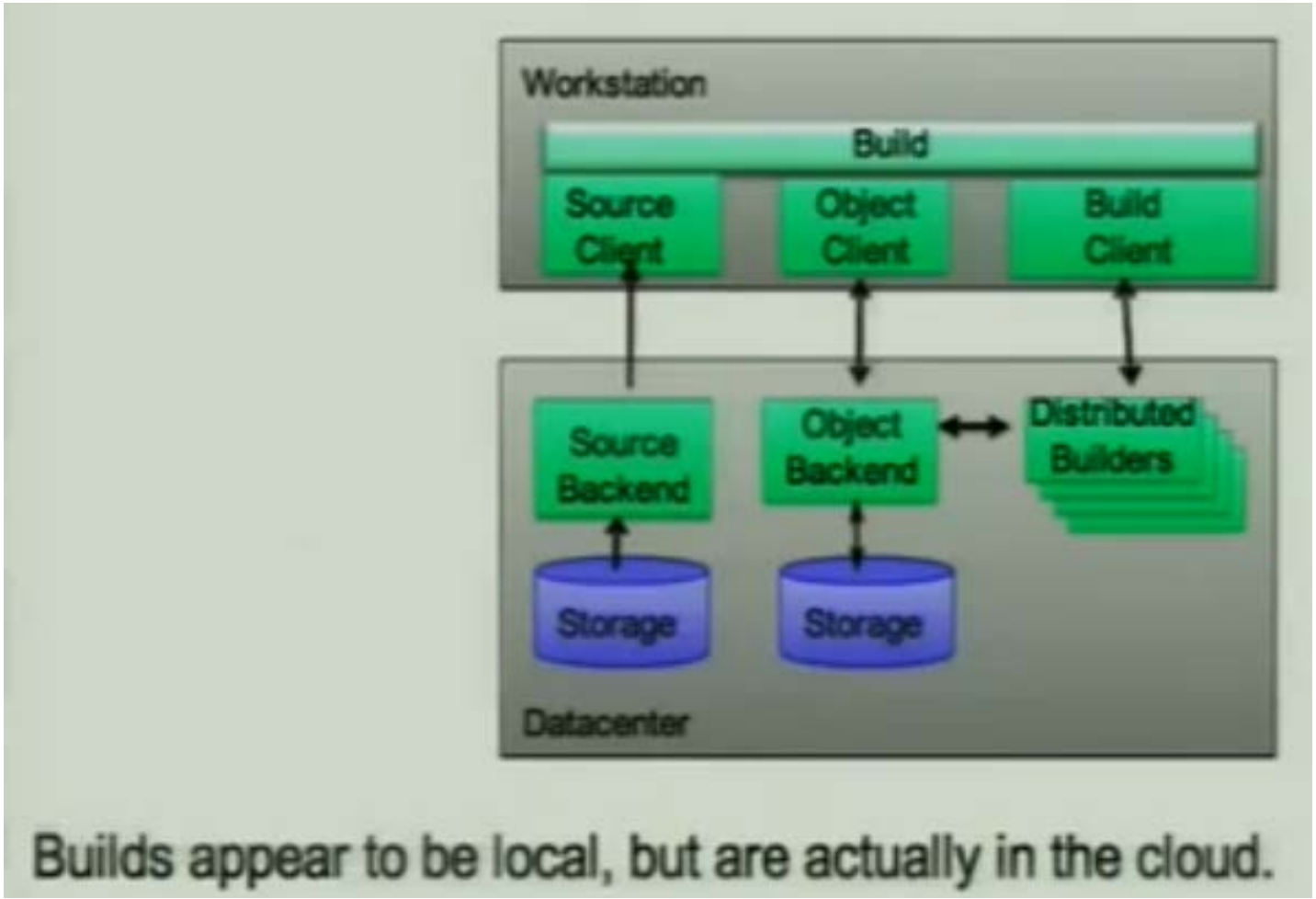
目标文件系统 - FUSE

- 构建产生大量的输出文件
- 需要访问所有输出文件吗？
  - 大部分文件用户不需要访问
    - obj, library文件
  - 少部分文件用户可能需要访问
    - binary, test文件
  - 甚至完全不需要访问输出文件
    - 用户只是要验证构建/测试是否通过
  - 按需获取，本地缓存

并行构建 – Build in Cloud

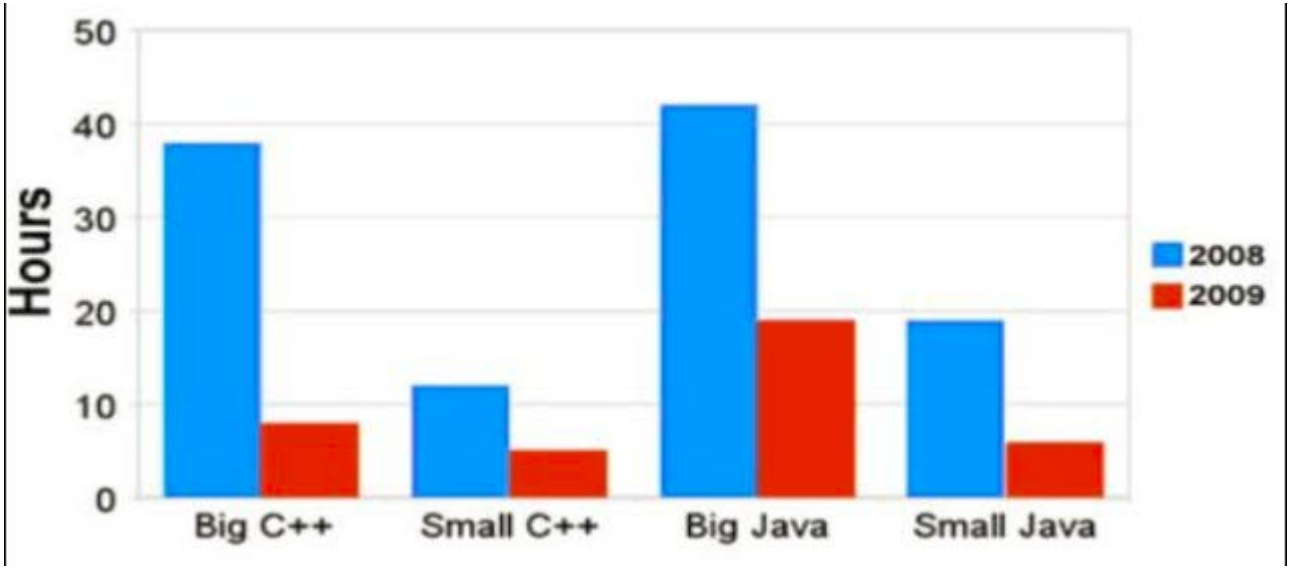


More details



And more…

- 10000+ cores, 50TB+内存
- 目标文件缓存7天，1PB存储
- 5亿次构建动作 (action)，2/3来自动构建
- 94%的构建动作命中缓存
- 效果：

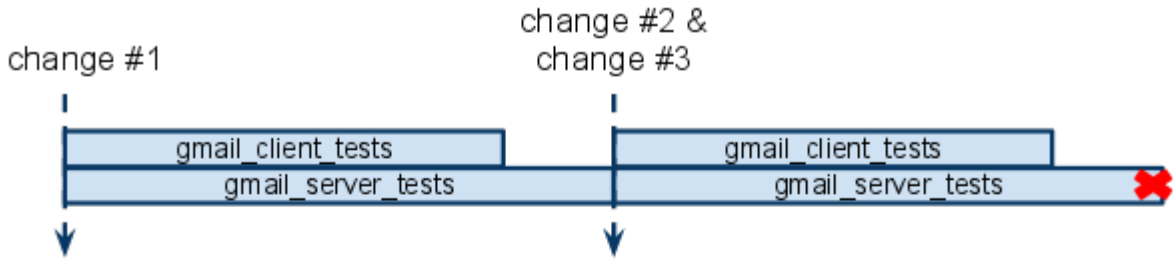


自动构建/持续集成

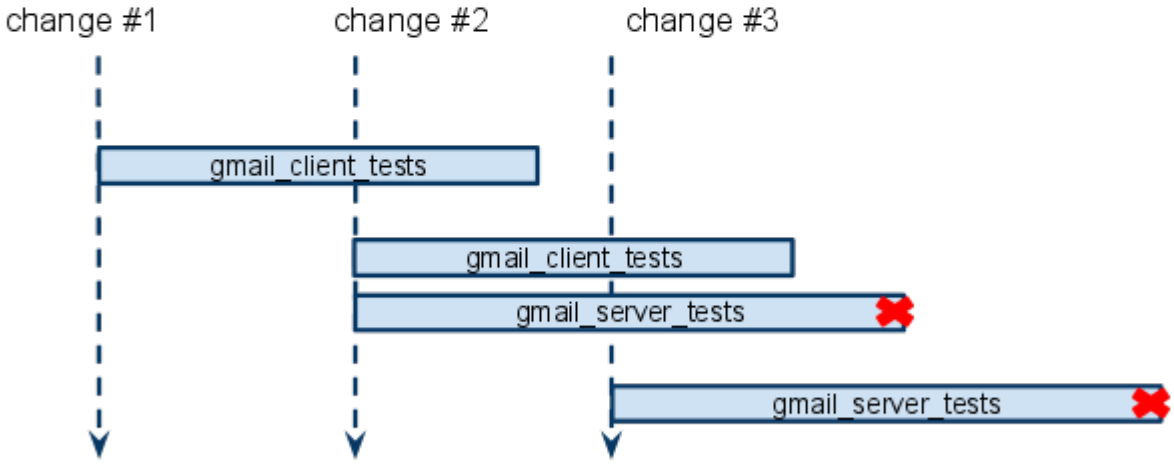
- 目标：
  - 自动，快速，精确定位有缺陷的代码提交
  - 每次代码提交后运行所有（受影响）的测试
- 并行构建
  - Build in Cloud
- 并发构建
  - 一次代码提交立即触发一次构建，无须等待上次构建/测试周期完成
- 依赖分析
  - 仅需要构建/测试受本次代码提交的测试用例

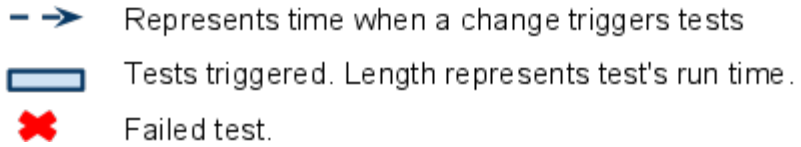
并发构建

Typical continuous integration system



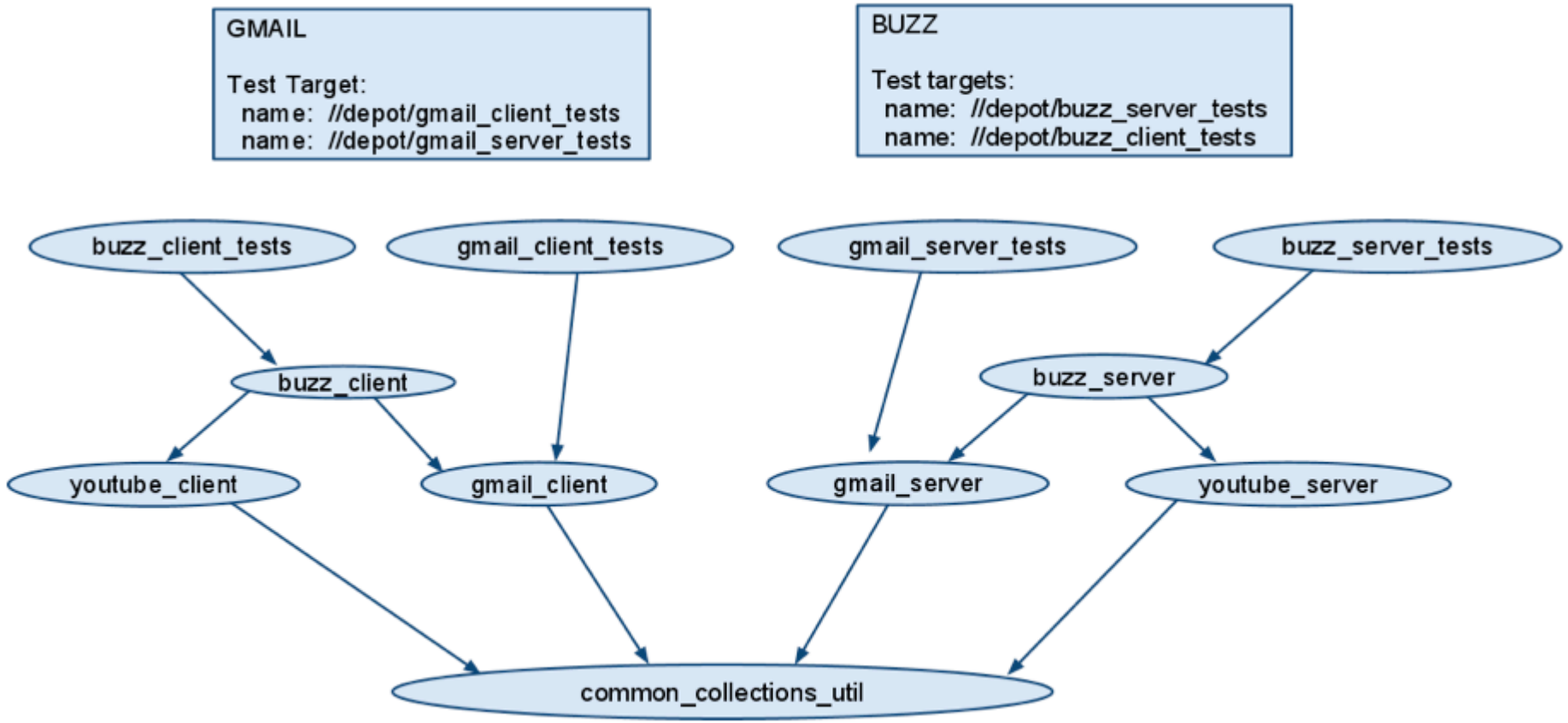
Continuous integration system with dependency analysis



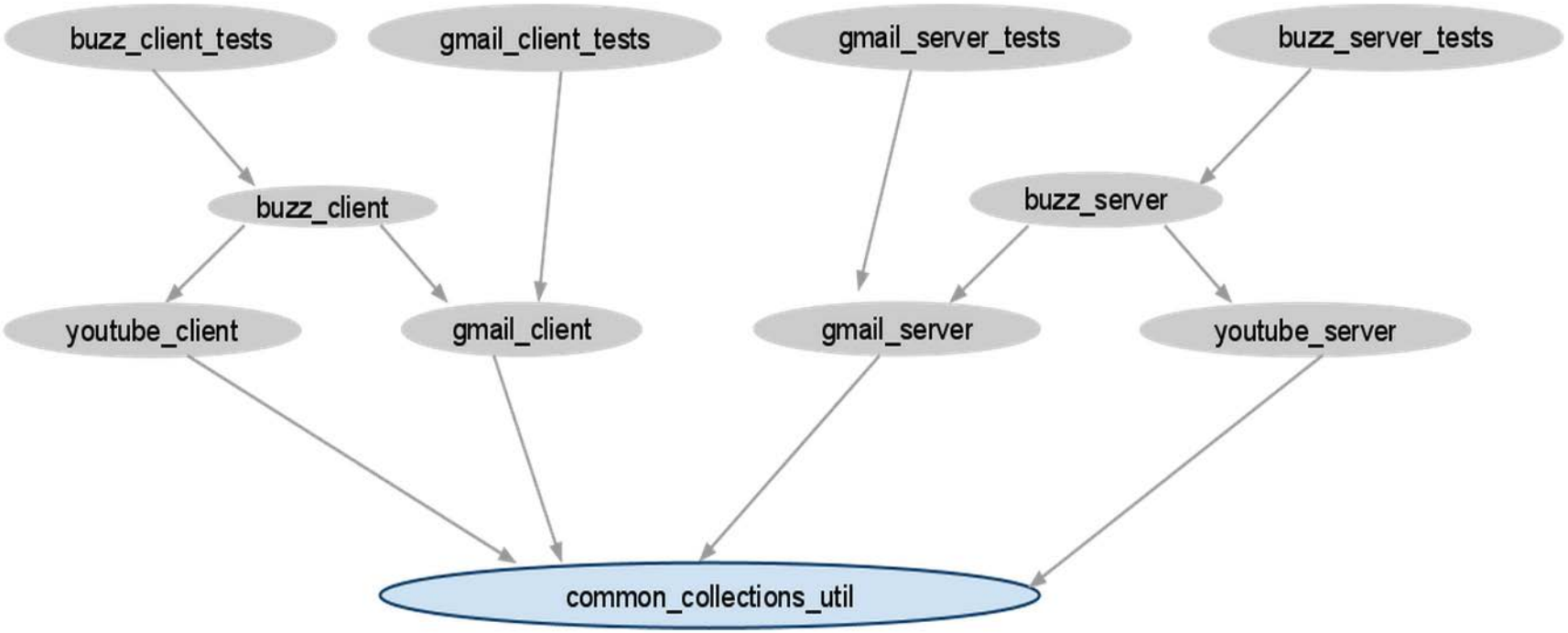


依赖分析

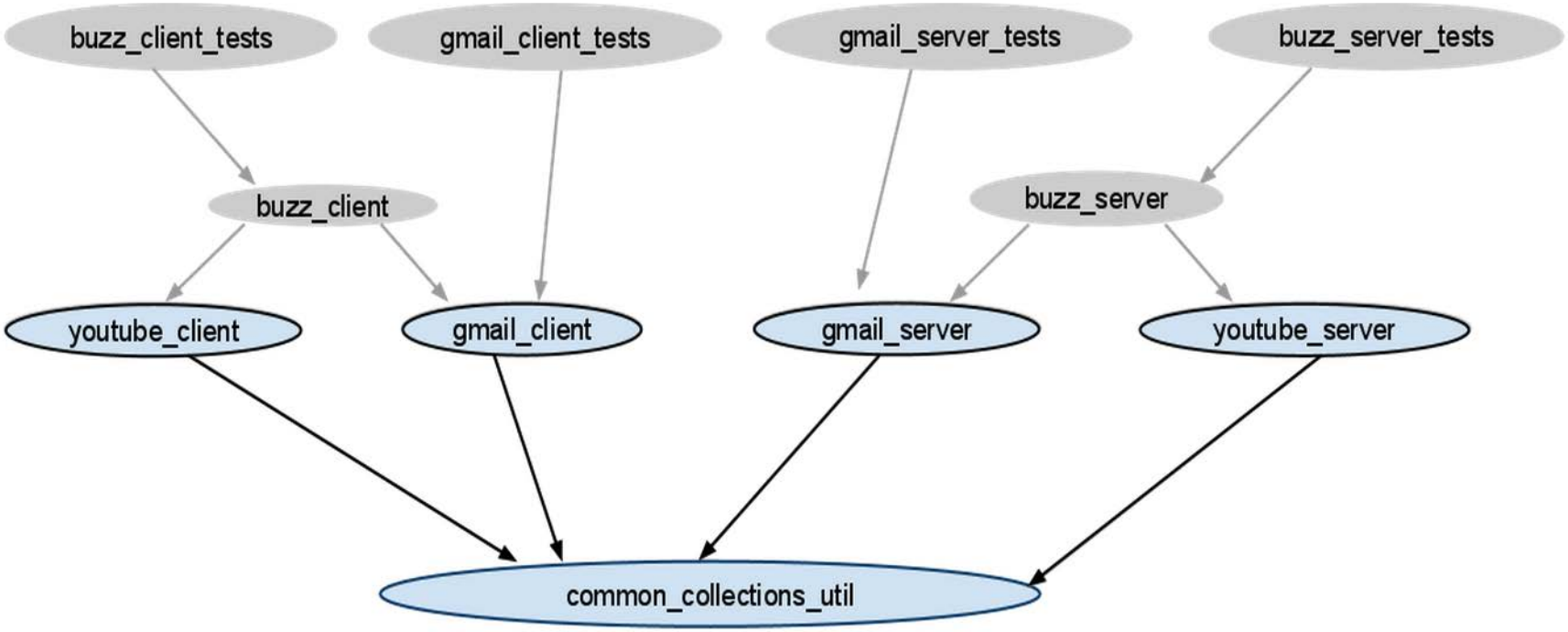
- 内存中维持所有测试用例和构建规则的依赖图
- 每次提交后广度优先搜索所有受影响的测例



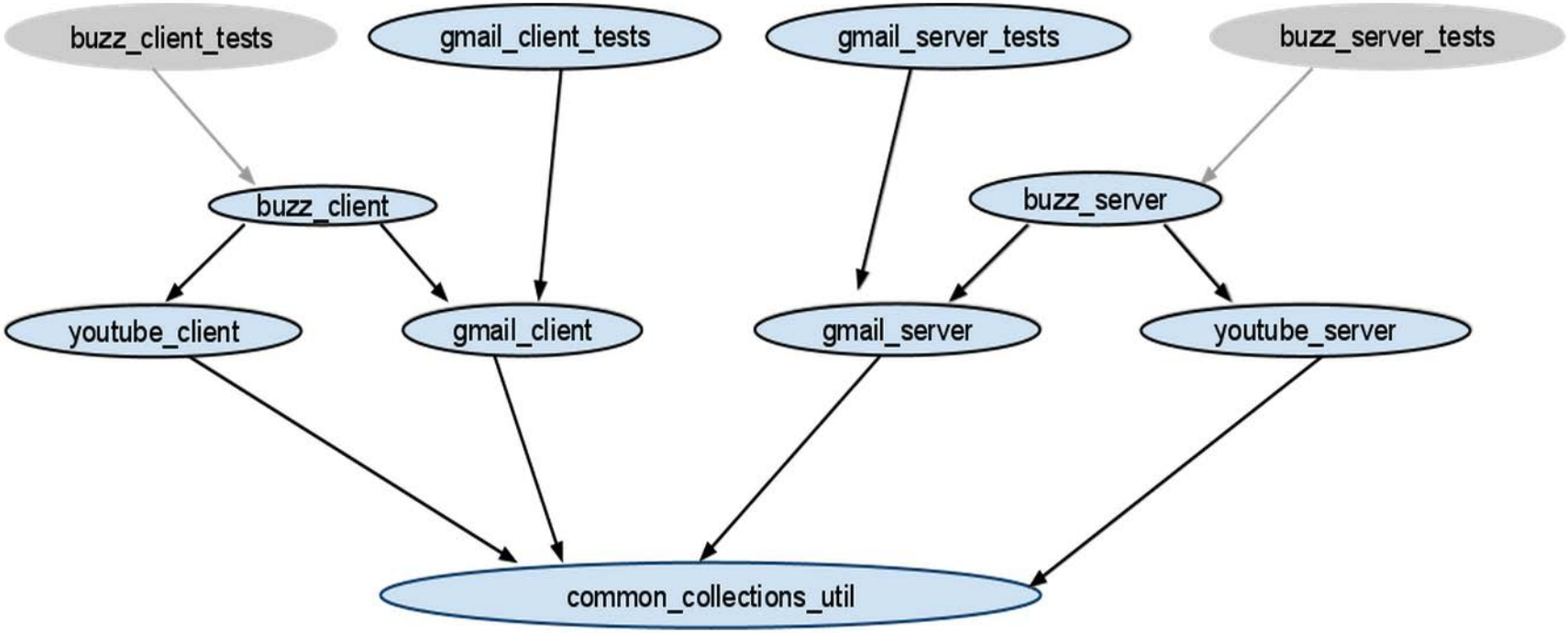
依赖分析—基础库变化


buzz_client_tests
gmail_client_tests
gmail_server_tests
buzz_server_tests
buzz_client
buzz_server
youtube_client
gmail_client
gmail_server
youtube_server
common_collections_util

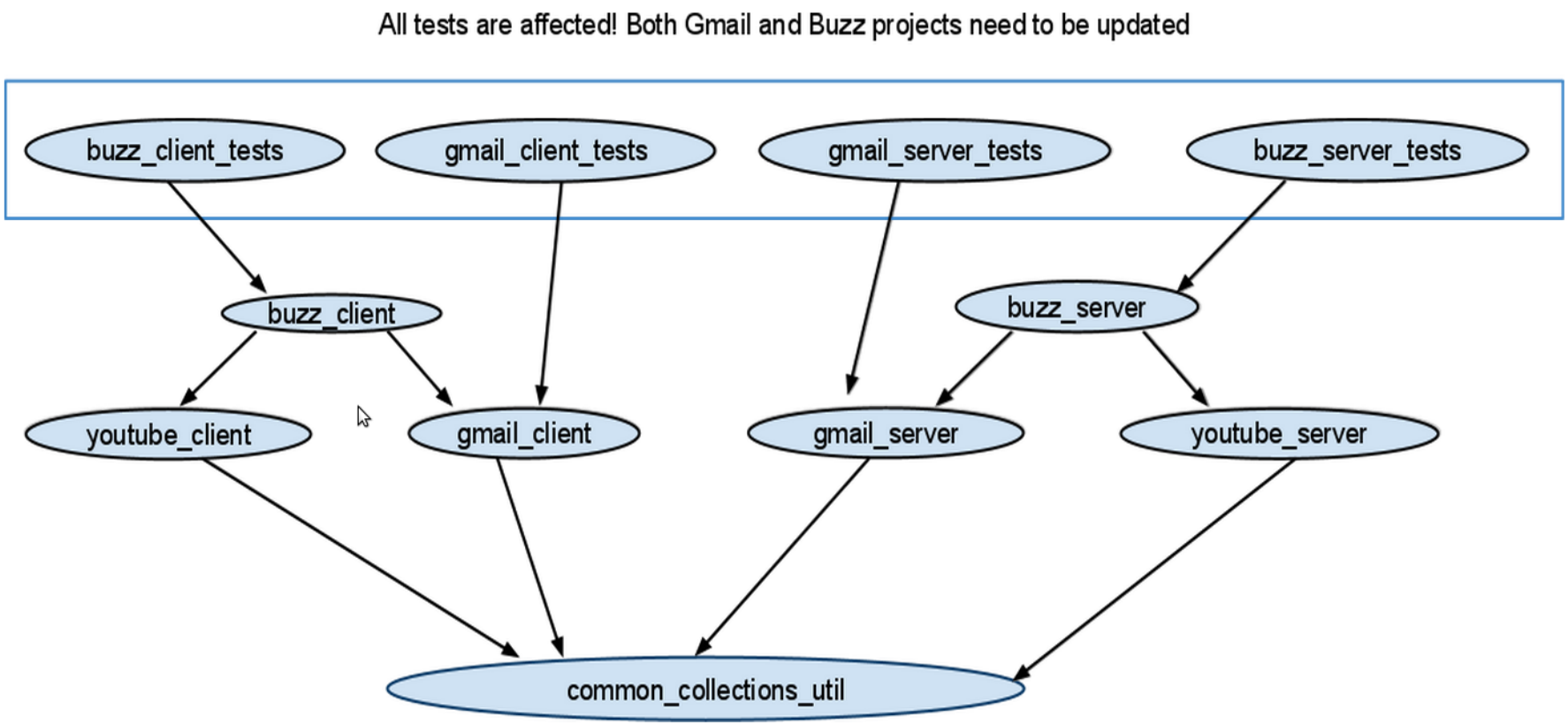
依赖分析—基础库变化


buzz_client_tests
gmail_client_tests
gmail_server_tests
buzz_server_tests
buzz_client
buzz_server
youtube_client
gmail_client
gmail_server
youtube_server
common_collections_util

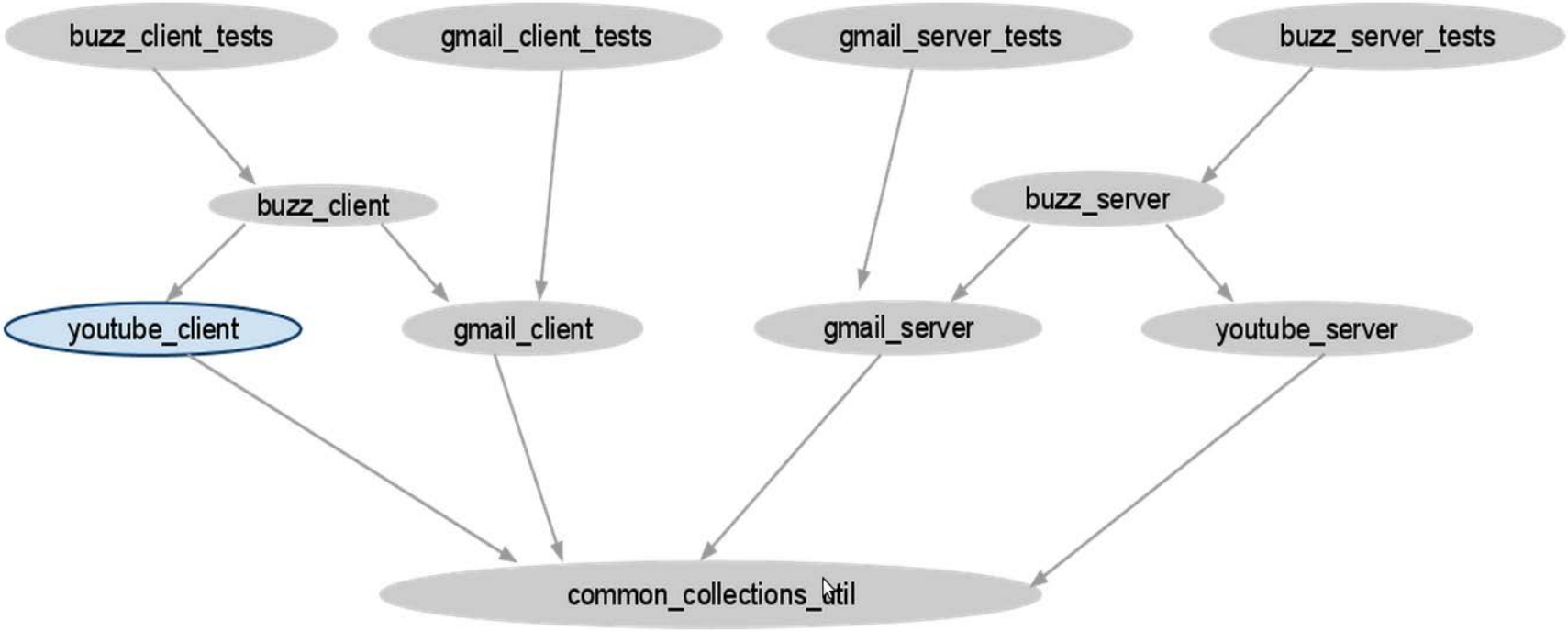
依赖分析—基础库变化


buzz_client_tests
gmail_client_tests
gmail_server_tests
buzz_server_tests
buzz_client
buzz_server
youtube_client
gmail_client
gmail_server
youtube_server
common_collections_util

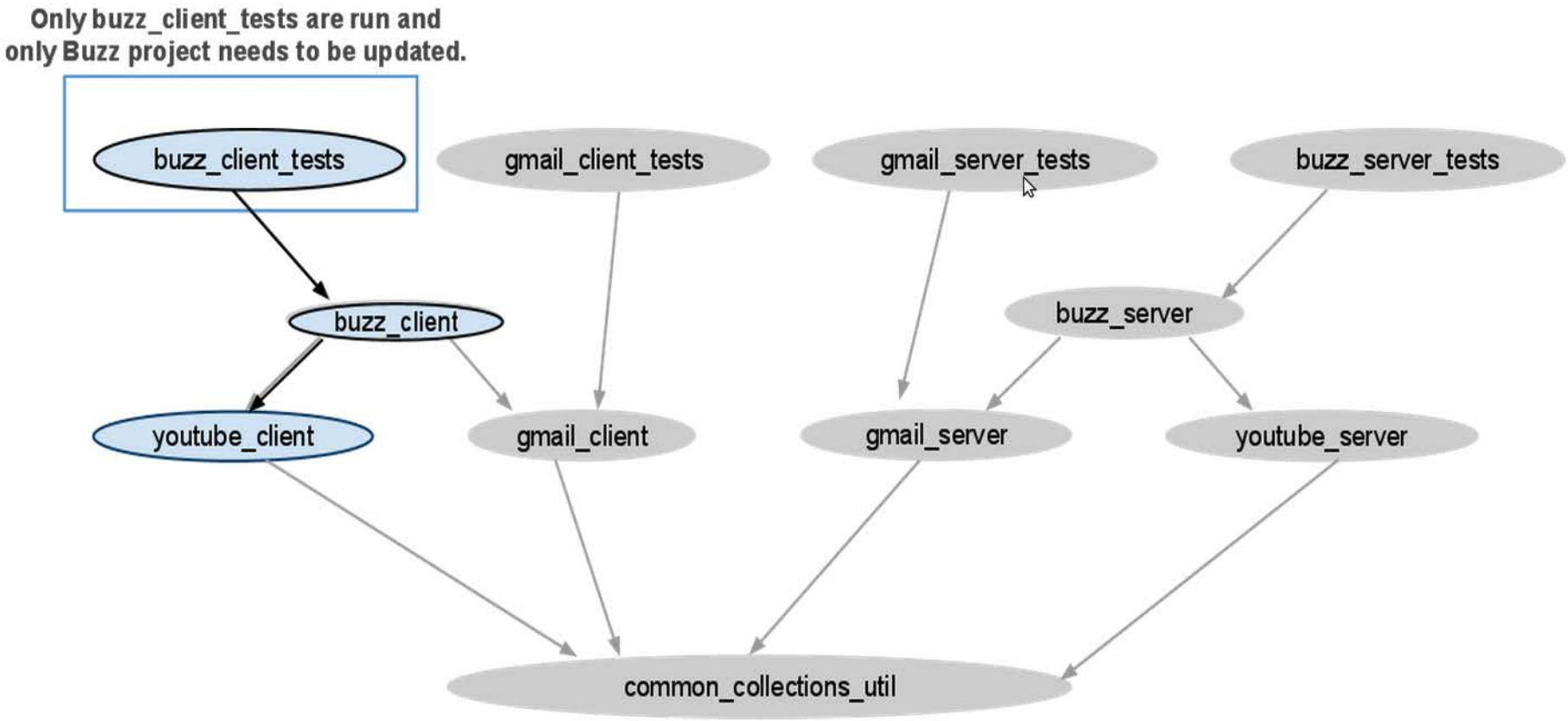
依赖分析—基础库变化


All tests are affected! Both Gmail and Buzz projects need to be updated
buzz_client_tests
gmail_client_tests
gmail_server_tests
buzz_server_tests
buzz_client
buzz_server
youtube_client
gmail_client
gmail_server
youtube_server
common_collections_util

依赖分析—依赖项目变化


buzz_client_tests
gmail_client_tests
gmail_server_tests
buzz_server_tests
buzz_client
buzz_server
youtube_client
gmail_client
gmail_server
youtube_server
common_collections_util

依赖分析—依赖项目变化


Only buzz_client_tests are run and
only Buzz project needs to be updated.
buzz_client_tests
gmail_client_tests
gmail_server_tests
buzz_server_tests
buzz_client
buzz_server
youtube_client
gmail_client
gmail_server
youtube_server
common_collections_util

结束语

- 工欲善其事，必先利其器
- 罗马不是一天建成的

参考资料

- Development at the Speed and Scale of Google
  - http://www.infoq.com/presentations/Development-at-Google
  - http://qconsf.com/dl/qcon-sanfran-2010/slides/AshishKumar_DevelopingProductsattheSpeedandScaleofGoogle.pdf
- Tools for Continuous Integration at Google Scale
  - http://www.youtube.com/watch?v=b52aXZ2yi08
- Testing at the speed and scale of Google
  - http://google-engtools.blogspot.com/2011/06/testing-at-speed-and-scale-of-google.html
- Google's Innovation Factory (and how testing adapts)
  - http://googletesting.blogspot.com/2010/04/googles-innovation-factory-and-how.html
- Google Engineering Tools Blog
  - http://google-engtools.blogspot.com/

Thanks!