CFGO全流程反馈优化技术分享

Update on GCC for openEuler　　王如锋

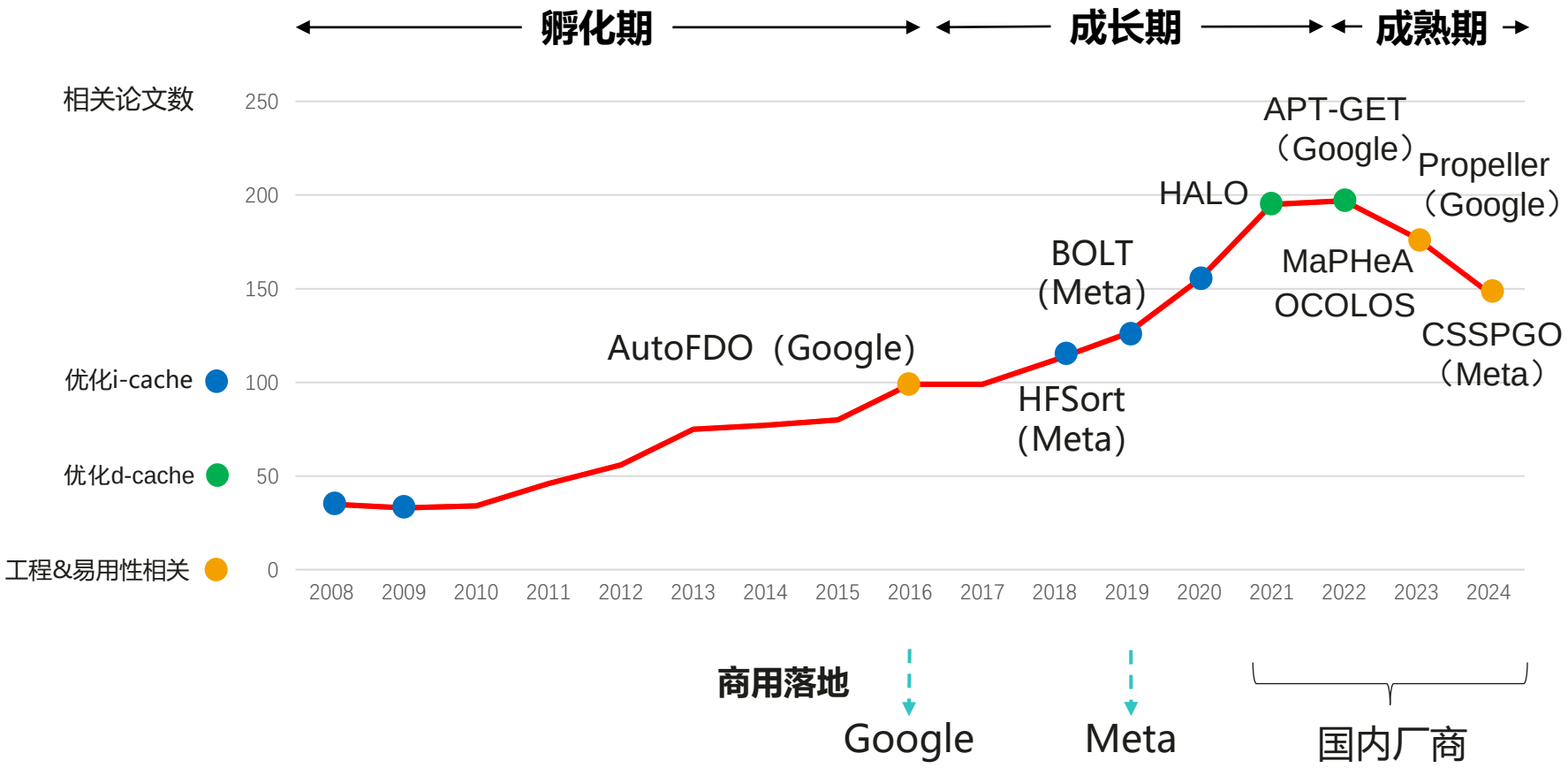
目录

反馈优化技术和CFGO框架

反馈优化原理和业界发展

基本概念：反馈优化是一种编译器优化技术，在保证程序功能不变的前提下，通过插桩/采样收集程序运行时信息，指导优化决策。

应用场景：适用于整机性能瓶颈在CPU上，且CPU瓶颈为frontend-bound型的C/C++应用，如数据库、分布式存储等场景。



技术趋势：

- 工业界关注采样路线，聚焦提升性能和改善易用性；学术界关注减少插桩开销
- 开始探索d-cache优化
- 从静态优化转向动态优化

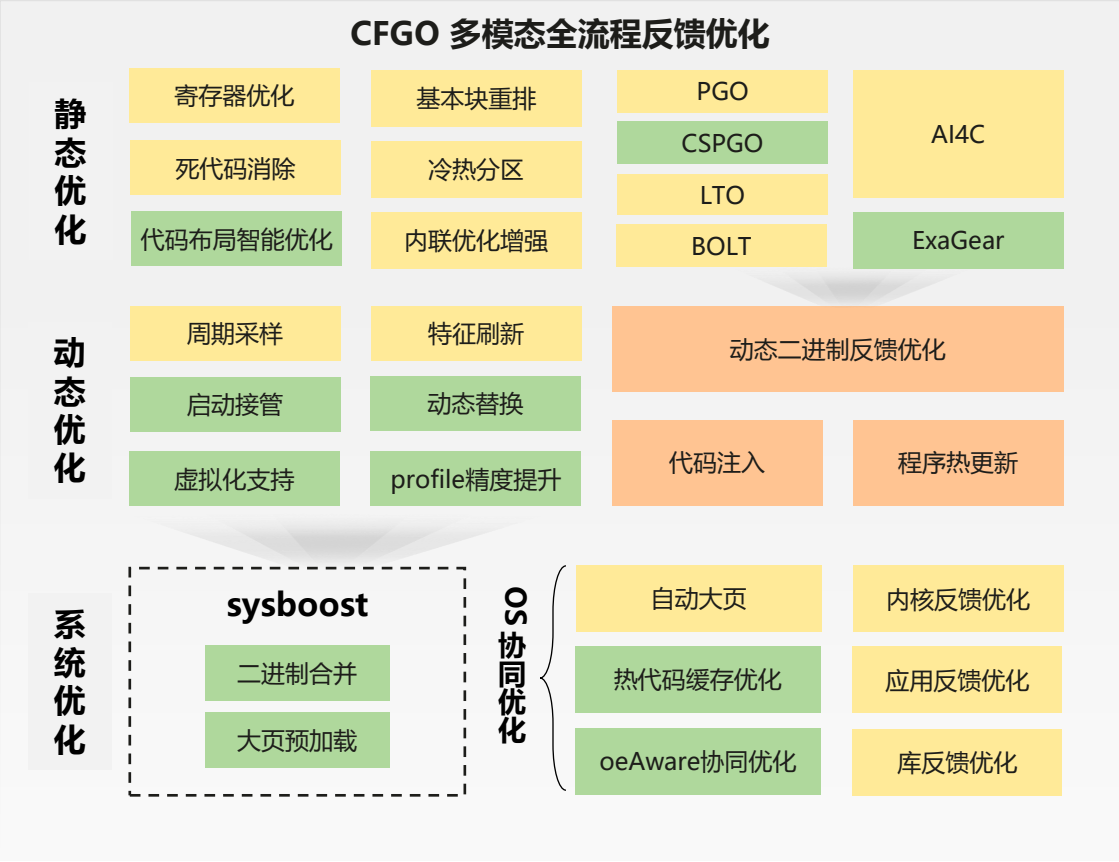
商用趋势：

- 基于开源版本增强可靠性和易用性后，实现内部商用落地

友商1：
应用技术：AutoFDO+BOLT
收益：性能收益5~10%
友商2：
应用技术：AutoFDO
收益：性能+5%
友商3：
应用技术： AutoFDO +BOLT+LTO
收益：时延降低5%+

CFGO反馈优化框架 (Continuous Feature Guided Optimization)

CFGO是GCC for openEuler中多模态（源代码、汇编码、二进制）、全生命周期（编译、链接、后链接）的持续优化手段



稳定性

功能稳定性高：专项测试加固，持续拓展应用范围，修复开源社区特性问题

性能泛化性好：具备短期/长期成熟方案，快速应对极端场景下的性能问题

易用性

流程优化：代码布局智能优化利用AI模型解耦采样流程，显著提高构建效率

工具整合：A-FOT整合常用反馈优化采样和编译流程，大大降低用户使用门槛

高性能

性能领先：极致优化场景下已超开源5-10%，持续探索软硬协同，算法优化中

全栈使能：内核/应用/依赖库协同反馈优化，实现场景性能进一步提升

MySQL +15% openGauss +5% Nginx +15% Ceph +7% Redis +5% kernel +5%

传统静态反馈优化的局限性

传统静态反馈优化的局限性

用户需要侵入式修改构建系统，使能步骤繁杂

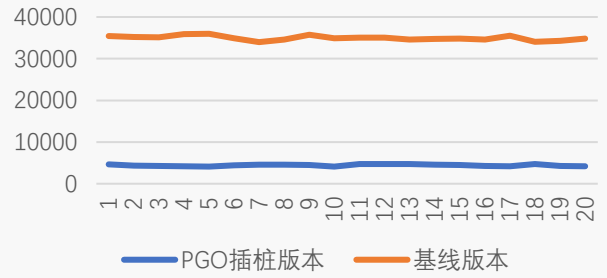
不同构建工具/不同应用适配方式不尽相同

使能反馈优化也有不同步骤

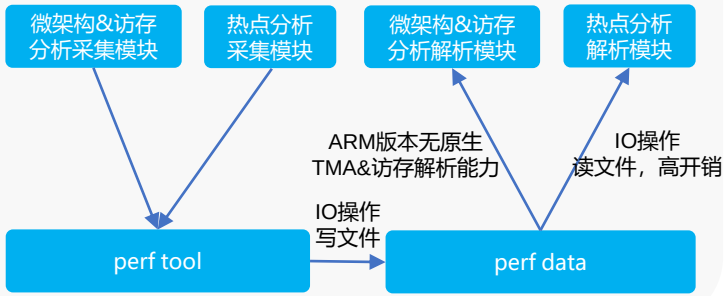
PGO	插桩编译	采样	二次编译
AutoFDO	编译	硬件采样	二次编译
BOLT	编译	硬件采样	二进制优化
A-FOT	填写编译采样配置	执行A-FOT	

优化过程收集/处理数据开销大，影响系统性能

PGO插桩版本性能影响大

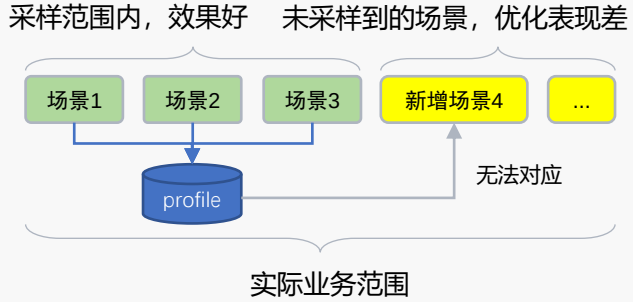


常用采样工具perf采样开销大，兼容性差

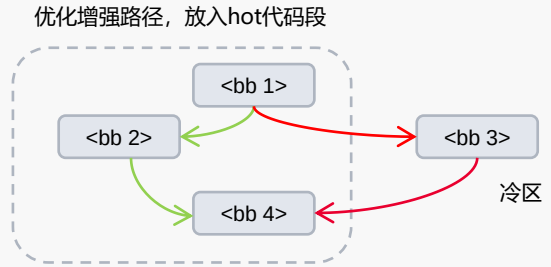


运行场景发生剧烈变化时，无法及时调整和更新

应用场景复杂，数据往往采集不完整



单次优化后的冷区代码，也有可能成为热点



业务变化导致出现1-3-4热点时可能有劣化

CFGO竞争力方向：静态极致优化 / 动态优化

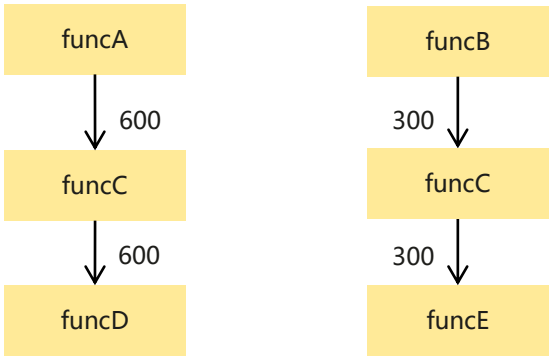
静态极致优化

极致优化路径：选项/参数调优 + PGO + CSPGO + Inst-BOLT

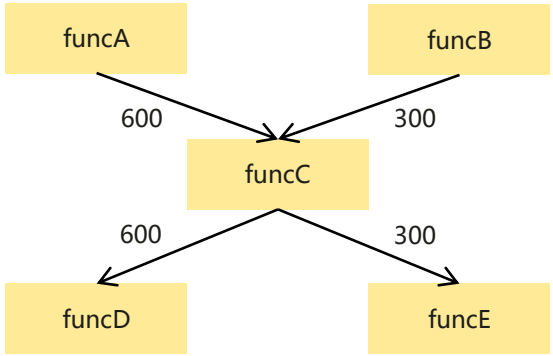
CSPGO：当前PGO的profile不是上下文敏感的，影响CFG准确性。通过增加内联后的CSPGO，可有效提升profile准确性，从而进一步提升性能

传统PGO

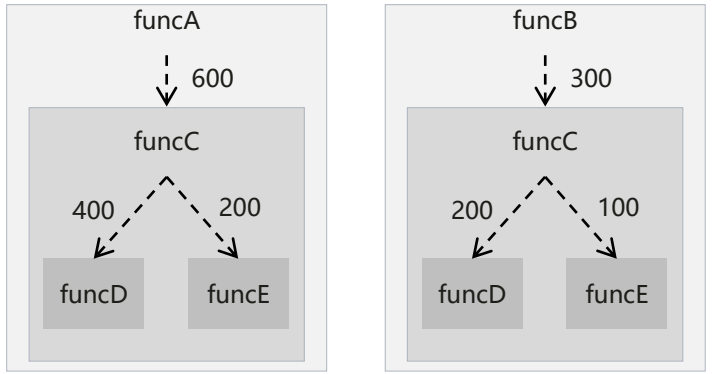
假设函数调用路径为A->C->D和B->C->E



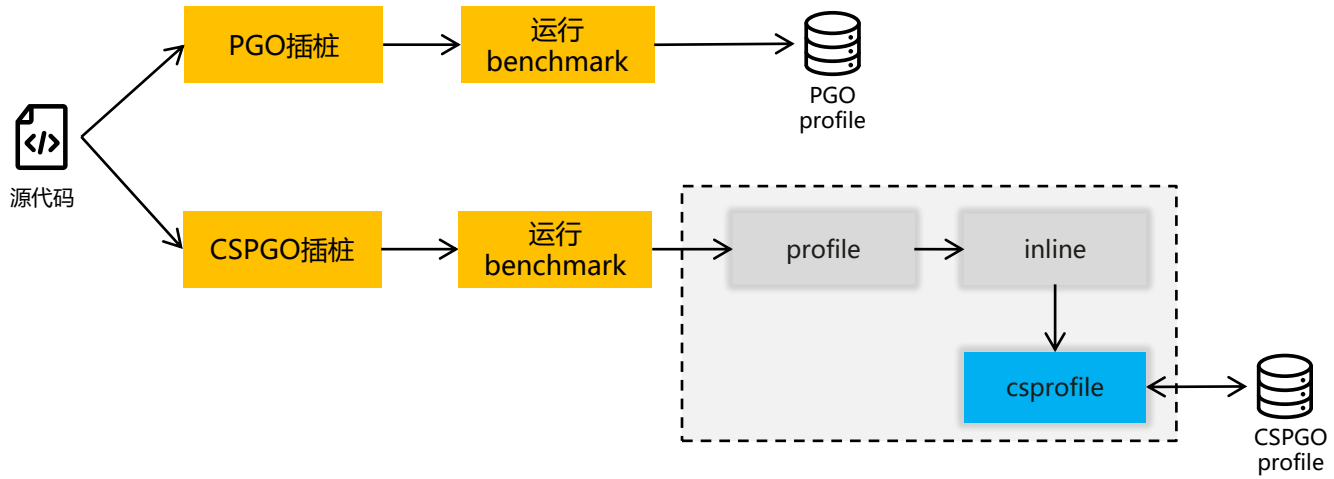
PGO后的控制流图示意如下，由于不区分上下文，函数C会被注释600次调用D，300次调用E

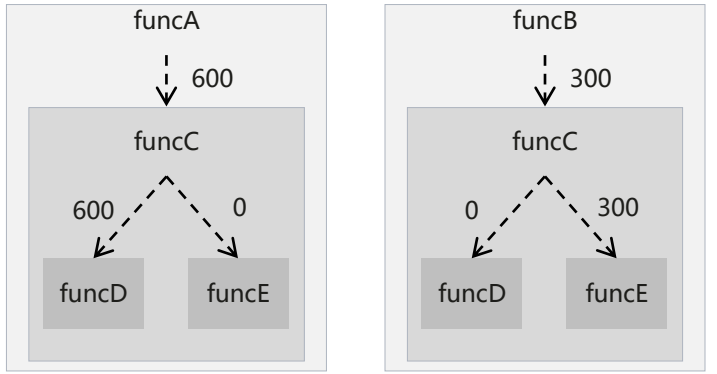


调用次数会导致内联优化后错误的分支概率，影响寄存器分配等后续编译优化遍，最终影响性能



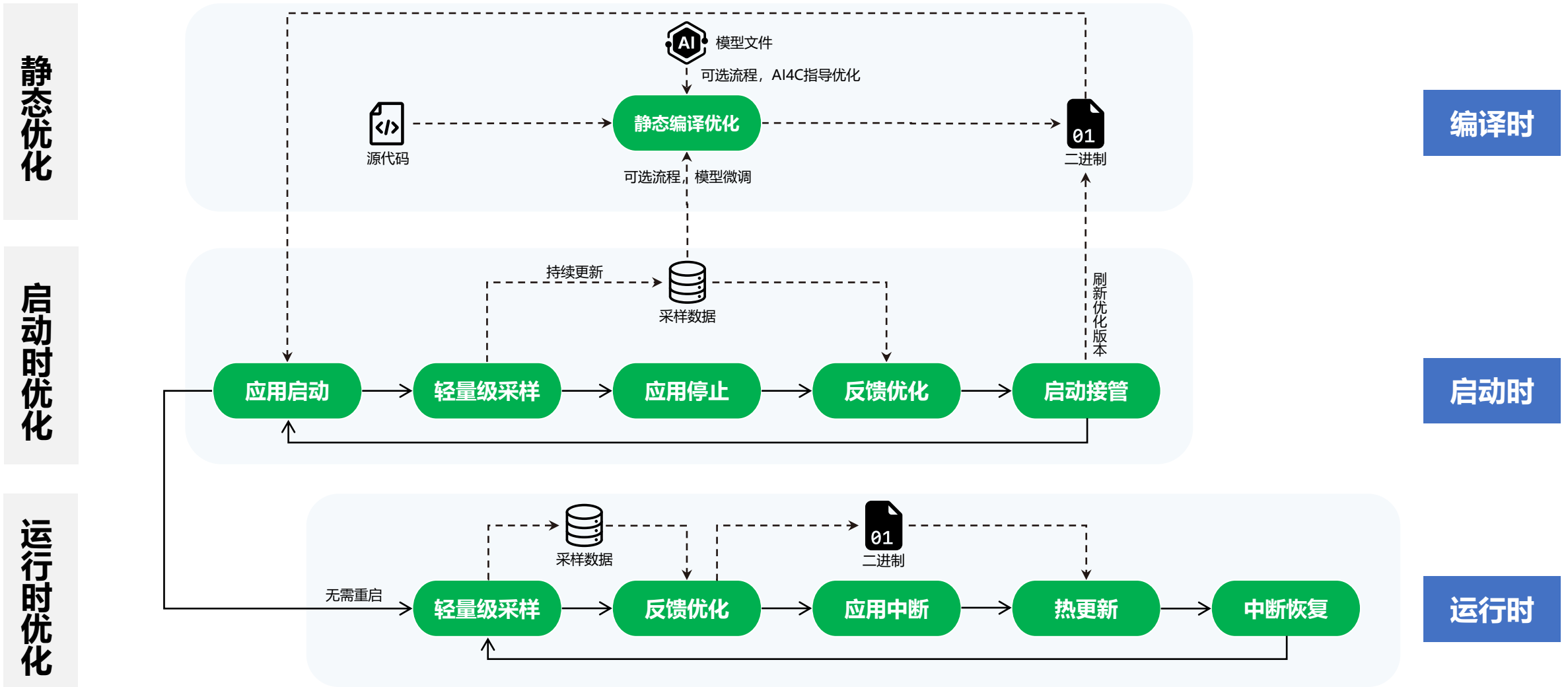
CSPGO





动态优化：启动时优化 / 运行时优化

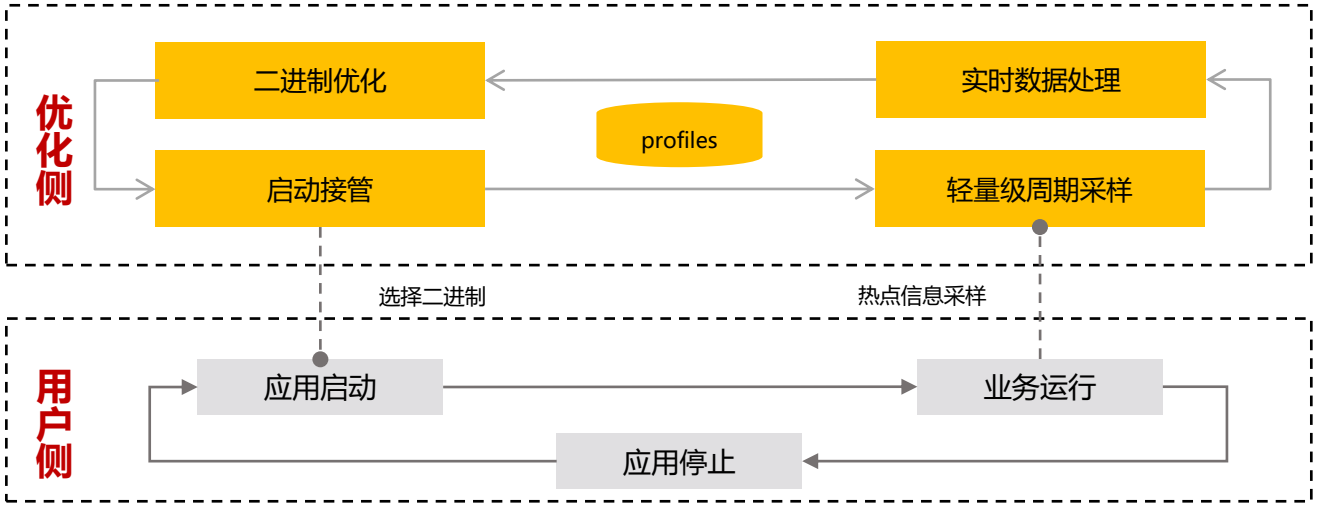
动态优化：将反馈优化从 编译链接时 延伸至 启动时/运行时， 解决易用性问题和性能泛化性问题



动态优化第一阶段：启动时优化

启动时优化：应用运行时自动执行采样和反馈优化，完成优化后自动接管下一次启动，重启后拉起优化版本。

启动时反馈优化框架



oeAware
业务在线无感调优框架

libkperf
轻量级全内存化采集工具

sysboost
微架构优化工具

llvm-bolt
二进制优化器

使用简单可控 (Simple)

- 一键启动优化服务
- 优化流程底层细节可控

轻量级适配和采样 (Light-Weight)

- 构建流程无侵入
- 轻量级采样和数据处理

实时优化 (Realtime)

- 实时采样，优化紧跟场景
- 启动接管，自动择优启动

安全可靠 (Safety and Reliability)

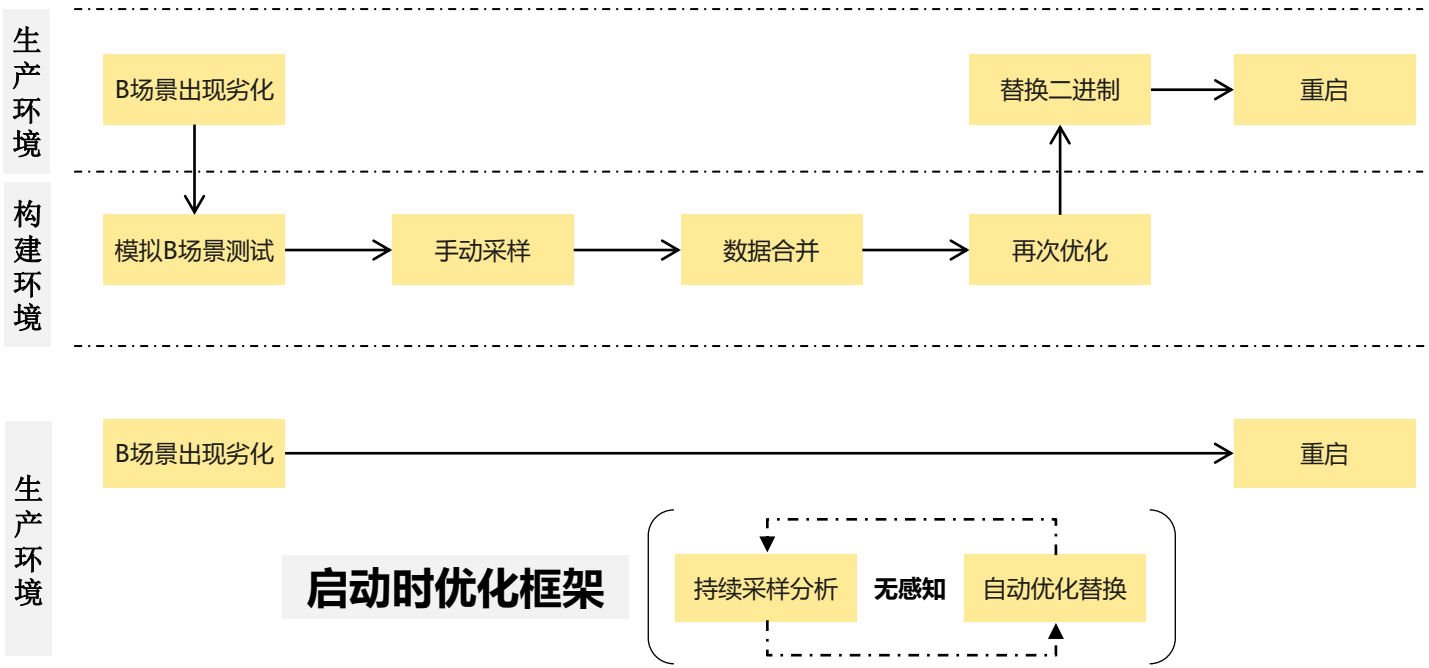
- 异常监控，自动检错回滚
- 详细现场记录，快速定位

启动时优化应用实例

场景热点特征变化，动态优化可以及时获取新的热点数据，快速生成新的优化版本

测试：基于sysbench只写场景采样优化的应用，测试TPCH Q1用例（复杂查询）

测试场景	Sysbench oltp_write_only	TPC-H Q1（不在采样范围内）
静态反馈优化效果	+18.7%	-2.7%



用例说明

两个场景热点差异大，在没有B场景的数据情况下，针对A场景的优化会有一定几率将B的热点函数/分支放入冷区，导致出现劣化

静态优化场景手动解决泛化性问题

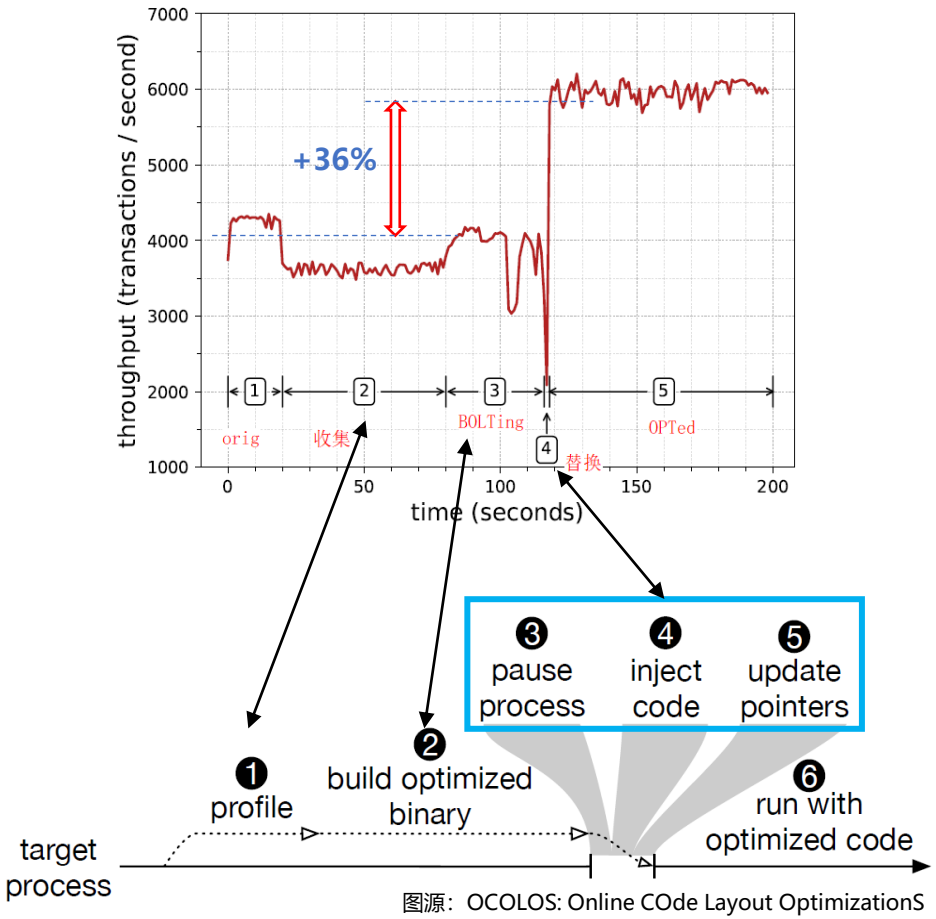
通过在构建环境中采样B场景、合并采样数据、重新优化后的二进制可以解决上述问题，但过程繁琐复杂，无法做到及时调整

动态优化自动更新应用

启动时优化场景下，出现劣化后仅需重启，即可自动应用最新优化版本；后续运行时优化场景下无需用户干预，可真正做到无感知自动更新

动态优化第二阶段：运行时优化

运行时优化：应用运行时自动执行采样和反馈优化，不需要重启，仅少量中断后即可使能优化版本。



图源：OCOLOS: Online COde Layout OptimizationS

核心技术点

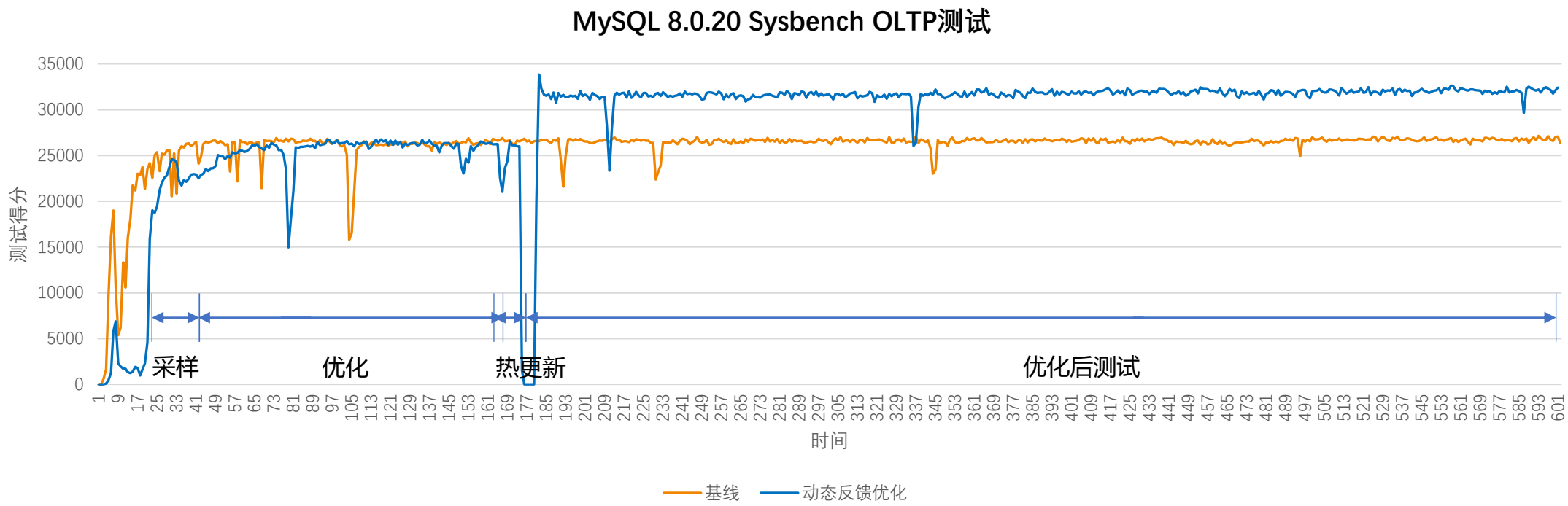
程序中断 | 利用ptrace等机制暂停应用进程，并保存上下文信息

代码注入 | mmap分配内存，注入优化后二进制信息，并配置内存权限

热更新 | 映射优化前后代码段，更新跳转指针，下一次执行优化后代码

适用场景 | 对实时性要求不高，允许短暂中断，不适合重启的应用

运行时优化应用实例



未来工作和展望

下一步计划

未来工作

易用性持续优化

（减少优化耗时；减少中断时间；自动决策调优对象）

持续探索新的优化机会

（CSSPGO；函数/BB块重排改进; 预取优化）

协同AI4C框架增强调优

（提高优化精度；增强优化Pass；模型替代采样）

挖掘底层芯片潜力

（提升执行单元利用率；结构体布局/堆内存优化）

THANKS