阿里云 | 奥运会全球指定云服务商

Production Ready ZGC for Java11
Alibaba Dragonwell11 上规模化实践

Alibaba JVM团队
唐浩

2021-6-19

GreenTea JUG Meetup

Alibaba Dragonwell

Alibaba Dragonwell 简介

Alibaba Dragonwell = OpenJDK+ 阿里巴巴云原生特性

可靠替代

OpenJDK下游

生产就绪

百万实例生产验证

云原生

多项云原生特性

公有云(ECS，EDAS)/开源用户

阿里巴巴业务

Alibaba Dragonwell 8 & 11

AJDK patch

OpenJDK

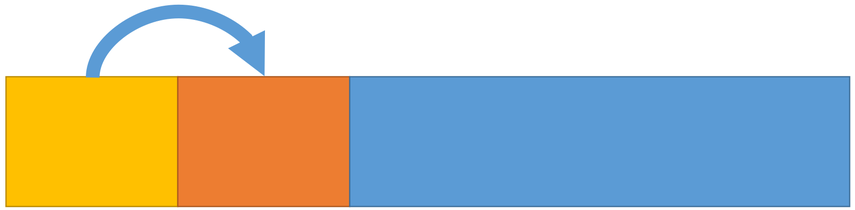
GC影响服务的SLA

- 长尾请求
  - GC暂停、磁盘抖动、网络延迟……
- SLA指标：RT P99 （99%响应时间）
- GC暂停因素
  - 每10s周期，一次100ms的GC暂停
  - RT P99 >= 100ms

OpenJDK11中的GC

- >100ms级别 GC
  - ParallelGC, CMS, G1
  - 暂停时转移对象
  - 难以扩展到大堆

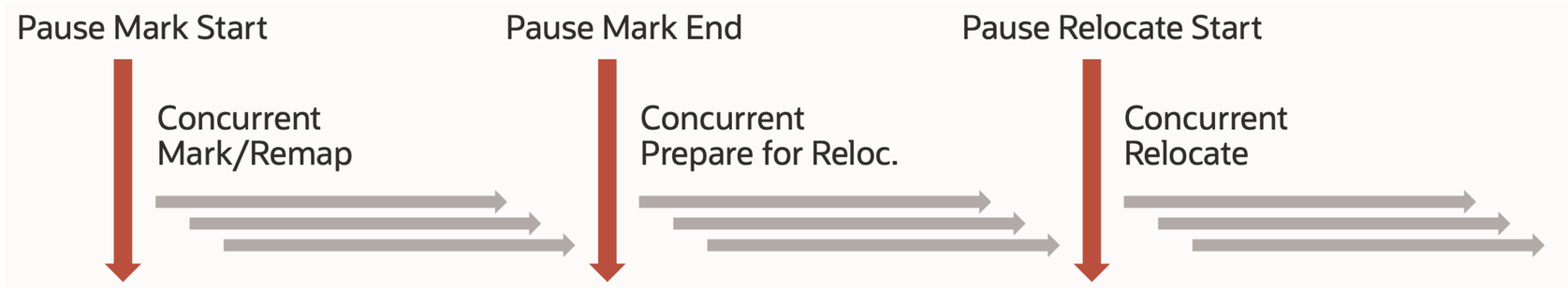
- ms级别（Pauseless GC）
  - **ZGC**, Shenandoah
    - Shenandoah: JDK12发布，移植回JDK11
  - 并发压缩
  - 大堆可扩展性



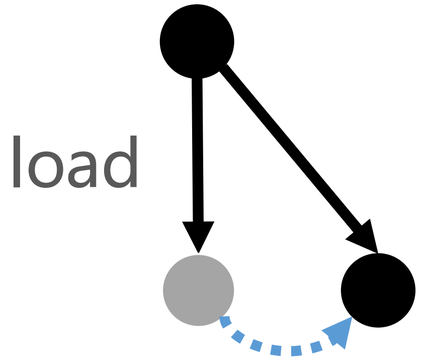
GC：转移对象
（整理、压缩）

ZGC简介

- 并发压缩
  - GC线程在后台运行
    - 标记、转移活跃对象
  - 解决碎片化问题



- 核心技术
  - 读屏障（load barrier）：修正、标记、转移
    - 原因：Java/GC线程相互影响
    - 目标：维持“弱三色不变式”
  - 染色指针（colored pointer）



- 业务价值
  - ms级暂停
  - 超大堆
  - 调优简单

JDK11的ZGC

- OpenJDK11 & Dragonwell11
  - Experimental ZGC

- ZGC发展现状
  - OpenJDK15： Production Ready ZGC
    - http://openjdk.java.net/jeps/377
    - 不是长期支持LTS的版本

阿里业务和云上用户对ZGC的需求

数据库　　搜索引擎　　消息队列　　风控　　推荐系统

提升用户体验

止损/盈利

降低机器
部署数量

阿里业务和云上用户对ZGC的需求

- SLA: RT P99 (P999, 最大值, 平均值)
  - 暂停时间越短越好

- 堆的规格
  - 大堆（100GB级别以上）
  - 多规格（100MB ~ 1TB）

- GC调优
  - 节约心智成本
  - 减少代码改造

Experimental ZGC 落地的问题

- 稳定性
  - Load barrier与Load分离
    - 表现1：两个指向同一个对象的指针不相等
    - 表现2：Crash
    - ……

load

load barrier

暂停 safepoint poll

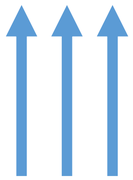
- RT未达到预期
  - 吞吐不足：回收跟不上分配
    - Allocation Stall / OOM
  - 受到ZGC非暂停时间的影响
    - Page Cache Flush

分配

回收

- 功能不完善
  - Unloading, Uncommit, Pre-touching，JFR

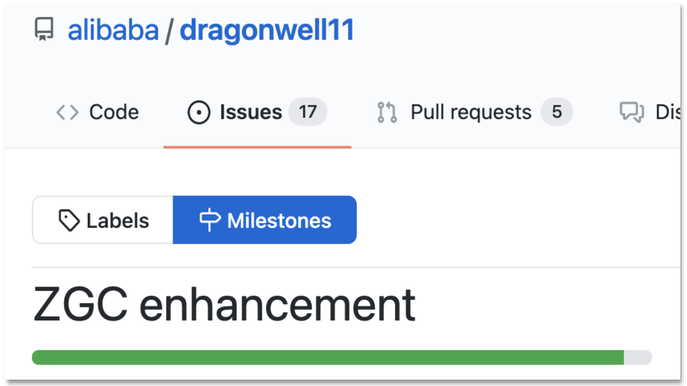
Dragonwell11 ZGC升级

ZGC 

- Production ready ZGC代码移植
  - JDK15的大部分ZGC相关代码
    - 修复重大Bug
    - 更多功能
    - 更多平台
- 解决Production ready ZGC实践中存在的问题
  - 阿里业务与云上客户中实践发现

Dragonwell11 ZGC升级

- ZGC升级目标
  - 解决业务问题
  - 保持JDK11 LTS质量的**长期稳定性**
- 内部AJDK ZGC升级
  - 验证多个业务的大量实例
- GitHub开源
  - Milestone
- Nightly build



Dragonwell11 ZGC升级：代码风险控制

- 保证JDK11 LTS 长期稳定性：不影响JDK11其他部分
  - 参考：Shenandoah GC Backport to JDK11
  - 公共代码的宏隔离和if隔离

- 编译时检查（**宏隔离**）
  - #if INCLUDE_ZGC ... #endif
  - ZGC_ONLY(...)

- 运行时检查（**if隔离**）
  - if (UseZGC) { ... }

```
//------------------------------LoadStoreNode---------------------------
// Note: is_Mem() method returns 'true' for this class.
class LoadStoreNode : public Node {
private:
  const Type* const _type;      // What kind of value is loaded?
  const TypePtr* _adr_type;     // What kind of memory is being addressed?
  ZGC_ONLY( uint8_t _barrier; ) // Bit field with barrier information
  virtual uint size_of() const; // Size is bigger
```

```
#if INCLUDE_ZGC
  if (UseZGC && FLAG_IS_DEFAULT(SoftMaxHeapSize)) {
    FLAG_SET_ERGO(size_t, SoftMaxHeapSize, MaxHeapSize);
  }
#endif
```

Dragonwell11 ZGC升级的好处

- Production Ready的稳定性
  - 重构 C2 load barrier
- Production Ready的功能
  - Unloading
  - Uncommit / SoftMaxHeapSize
  - JFR事件
  - …
- 更多通用优化
  - 避免OOM
  - 设定触发阈值
  - 对象分配阈值

ZGC稳定性：重构 C2 Load Barrier

- JIT编译器
  - C2: Sea of nodes

load

load barrier

暂停 safepoint poll

- 问题：
  - 原因：Load与Load Barrier被safepoint poll隔开
    - 无法维持“弱三色不变式”
  - 发现：Object相等判断错误
    - assert(a == b); // 此处a与b的hash值相等

```
//------------------------------LoadStoreNode---------------------------
// Note: is_Mem() method returns 'true' for this class.
class LoadStoreNode : public Node {
private:
  const Type* const _type;      // What kind of value is loaded?
  const TypePtr* _adr_type;     // What kind of memory is being addressed?
  ZGC_ONLY( uint8_t _barrier; ) // Bit field with barrier information
  virtual uint size_of() const; // Size is bigger
```

- 改造Load相关的C2 node
  - 增加barrier node data
  - 推迟到机器码展开阶段（不会被C2优化打乱）

ZGC多平台支持

- Dragonwell ZGC 新增支持 AArch64/Linux
  - 业务需求驱动
  - Windows & MacOS（Dragonwell 暂无计划）

ZGC类卸载 (Class Unloading)

- Concurrent Class Unloading是OpenJDK12引入的
  - OpenJDK12对公共数据结构进行并发化改造
  - 直接移植到Dragonwell11
    - 难以控制代码风险
    - 后续与上游同步的成本高

Dragonwell11
是阿里长期支持的版本

- 实现了暂停的Class Unloading
  - 参考Shenandoah移植到JDK11的策略

- -XX:+ClassUnloading

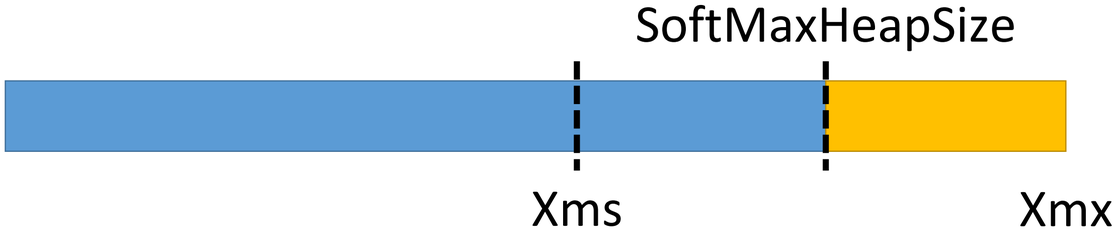
ZGC内存使用的优化

- 内存规格扩展
  - 超大堆：16TB
  - 超小堆：8MB
- 并行Pre-touching
- 归还物理内存
  - -XX:+ZUncommit
  - Xmx, Xms, SoftMaxHeapSize
  - ZUncommitDelay

同一个业务
不同规格的机器

大堆应用启动的
响应速度

同一台机器部署
多个实例



ZGC响应时间优化

- **非暂停因素**影响RT
  - ZGC 的 Page Cache Flush

- ZPage (Region)
  - Small (2MB), Medium (32MB), Large (2*N MB)
  - 例子：缺少Medium时候就要从Small/Large转化
  - Page Cache Flush
    - 耗时长：多次系统调用mmap
    - 影响面大：锁住ZPage分配全局锁

ZGC响应时间优化

- 移植“提升ZPage分配并发度”的特性
  - 减少使用ZPage分配全局锁
  - 异步执行mmap
  - 效果：大幅减少Page Cache Flush引发的线程阻塞

- 调整Medium ZPage的对象大小阈值
  - 原来的范围：256KB~4MB
  - 新增：ZMediumObjectUpperBound
    - -XX:ZMediumObjectUpperBound=10MB（调整后的范围：256KB~10MB)

ZGC吞吐问题

- 现象1: Allocation Stall
  - 回收速度跟不上分配速度
  - 增加 堆大小(Xmx) 或 GC线程数量(ConcGCThreads)
  - **GC触发时机**
    - ZAllocationSpikeTolerance：处理分配速率毛刺
    - ZHighUsagePercent：超过ZHighUsagePercent%时触发ZGC

线上监控
堆水位高报警

- 现象2: OOM
  - 预留固定的空间：作为对象转移的分配区域
    - 预留空间不足产生OOM：转移速度过快
  - ZRelocationReservePercent
    - 堆的ZRelocationReservePercent%作为预留空间
    - 参考Shenandoah的解决方案

ZGC监控升级

- 日志细节更新
  - 错误活跃对象信息更正
  - 不同规格ZPage的统计信息
  - ……

- JFR事件
  - ZAllocationStall
  - ZPageAllocation
  - ZRelocationSet & ZRelocationSetGroup
  - ZUncommit
  - ZUnmap

ZGC升级小结

- 维持Dragonwell11的稳定性

- Production Ready
  - Bug修复
  - 多平台
  - 新功能

- Alibaba新增通用特性
  - ZHighUsagePercent
  - ZRelocationReservePercent
  - ZMediumObjectUpperBound

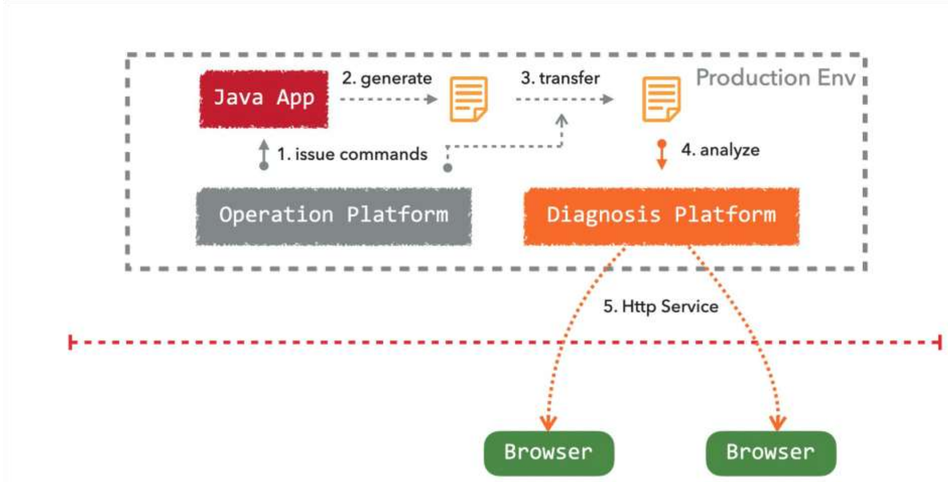
展望

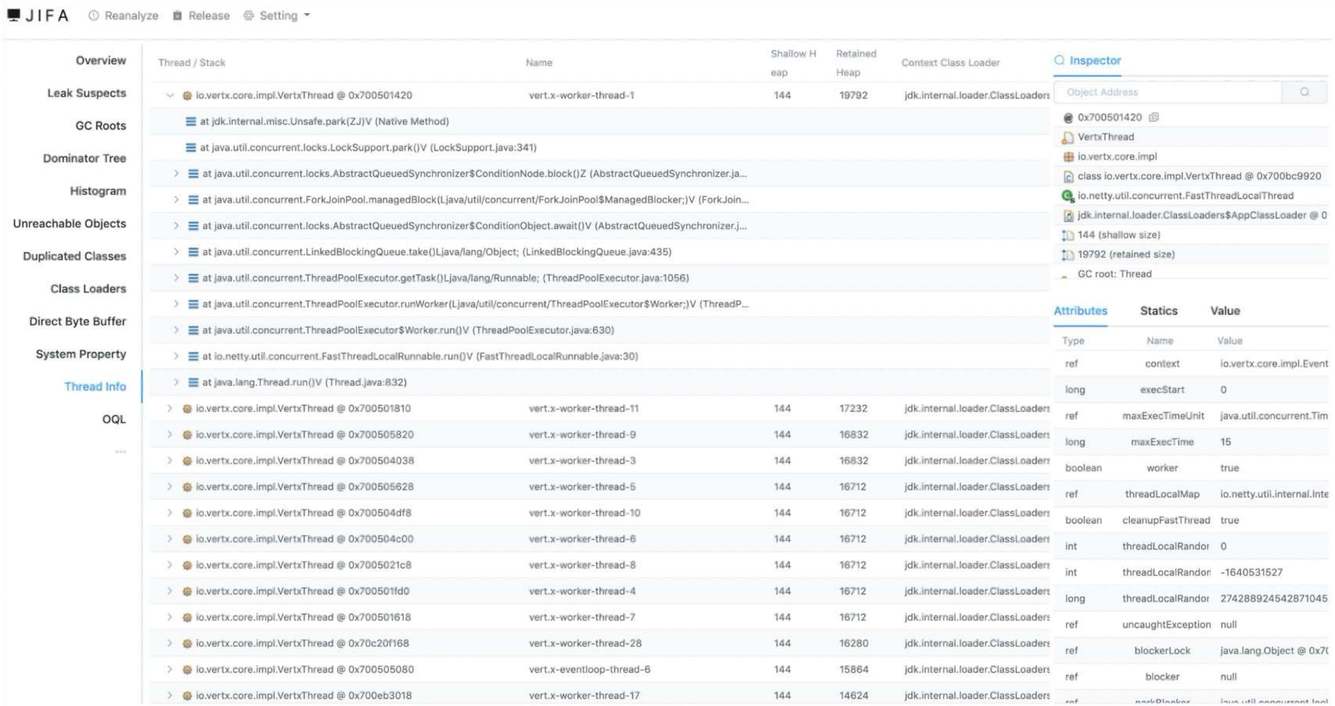
- 替代品
  - Shenandoah GC
    - 32GB以内效果更好 (Compressed oops)

- 未来方向
  - Compressed class pointers
    - 内部实验跑分提升明显
  - Class data sharing
  - 多平台支持
  - 彻底解决OOM问题：原地对象转移（JDK16）
  - 亚毫秒级别暂停：并发线程栈处理（JDK16）
  - 吞吐量提升：分代ZGC（最近代码公开）

开源项目

- Eclipse JIFA (Incubator)
  - Java 应用在线诊断平台
  - 解决本地资源不足，难以分析生产环境大堆文件等问题
  - https://github.com/eclipse/jifa

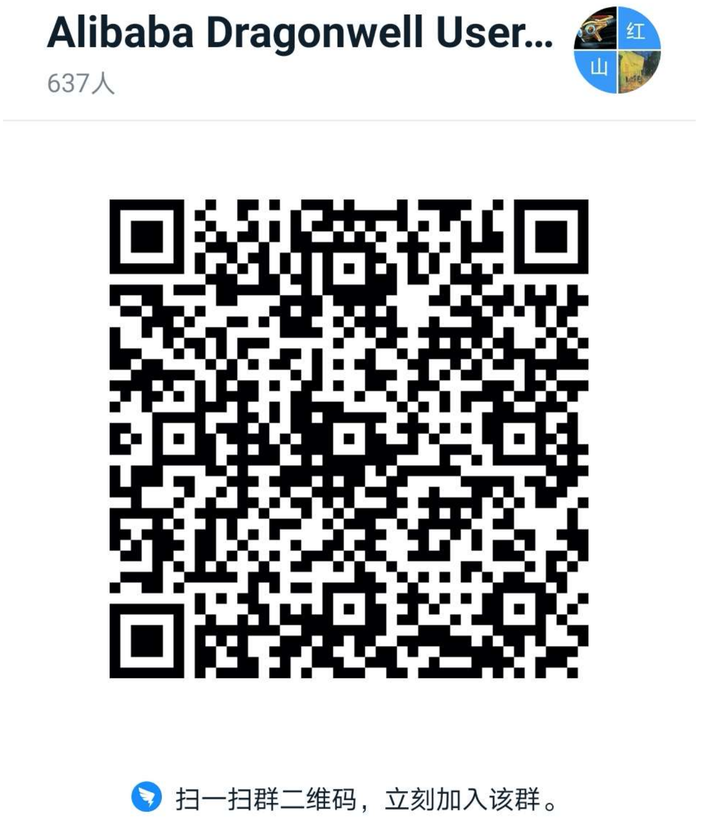




Dragonwell 获取支持

- https://github.com/alibaba/dragonwell11

- 钉钉群



奥运会全球指定云服务商