2023 · 上海
SkyWalking Summit
演讲主题

陆家靖

收钱吧框架工具团队负责人、SkyWalking PMC Member

“基于SkyWalking Agent的持续性能剖析与交互式诊断”
目录

CONTENTS

01. 持续性能剖析
    Continuous Profiling

02. 交互式诊断
    Interactive Diagnosis
01
Continuous Profiling
Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers

Google-Wide Profiling (GWP), a continuous profiling infrastructure for data centers, provides performance insights for cloud applications. With negligible overhead, GWP provides stable, accurate profiles and a datacenter-scale tool for traditional performance analyses. Furthermore, GWP introduces novel applications of its profiles, such as application-platform affinity measurements and identification of platform-specific, microarchitectural peculiarities.

FIG. GWP published by Google in 2010: low overhead, stable, accurate, scalable
FIG. Since GWP, many major vendors have joined “Continuous Profiling”:
Pyroscope is an open-source solution, acquired by Grafana Lab on 2023-03-15.
FIG. Architecture of the Grafana Pyroscope
Java: How to collect? Java Flight Recorder

- Capture both JVM and application data
  - GC
  - Synchronization
  - Compiler
  - CPU usage
  - Exceptions
  - I/O
- Sampling-based profiler
  - Very low overhead: 2-3%
- Buffers
  - Thread Buffer
  - Global Buffer
  - Repository (Disk chunk)

Behind the Scenes: Event Data Flow

![Diagram of event data flow]

1 with event streaming

FIG. How JFR works in the background: API events and JVM events as sources. https://www.infoq.com/presentsinations/monitoring-jdk-jfr
Java: How to collect? Java Flight Recorder

```java
ejdk.ExecutionSample {
    startTime = 2023-02-13T05:53:01.646060063Z
    sampledThread = "http-nio-8080-exec-482" (javaThreadId = 12559)
    state = "STATE_RUNNABLE"
    contextId = 0
    stackTrace = [
        java.util.LinkedHashMap.entrySet() line: 635
        java.util.HashMap.putMapEntries(Map, boolean) line: 513
        java.util.HashMap.<init>(Map) line: 491
        io.netty.bootstrap.AbstractBootstrap.copiedMap(Map) line: 429
        io.netty.bootstrap.AbstractBootstrap.options() line: 417
        ...
    ]
}
```

FIG. The anatomy of a JFR event and a typical example
This project is a low overhead sampling profiler for Java that does not suffer from Safepoint bias problem. It features HotSpot-specific APIs to collect stack traces and to track memory allocations. The profiler works with OpenJDK, Oracle JDK and other Java runtimes based on the HotSpot JVM.

async-profiler can trace the following kinds of events:

- CPU cycles
- Hardware and Software performance counters like cache misses, branch misses, page faults, context switches etc.
- Allocations in Java Heap
- Contented lock attempts, including both Java object monitors and ReentrantLocks
Java: How to analyze? FlameGraph

https://github.com/brendangregg/FlameGraph

FIG. A typical flamegraph
Java: How to analyze? JDK Mission Control

Arch Overview

FIG. Overview of the system design
package org.example;

import jdk.jfr.consumer.RecordedEvent;
import jdk.jfr.consumer.RecordingFile;

import java.nio.file.Paths;
import java.util.List;

public class App {
    public static void main(String[] args) throws Exception {
        List<RecordedEvent> events = RecordingFile.readAllEvents(Paths.get("/path/to/jfr"));
        for (RecordedEvent event : events) {
            // process...
        }
    }
}
JFR Reader: build call stack

FIG. Build the call stack (Tree with treeNode as children)
FIG. Memory issue: large heap size while building the call stack with millions of (allocation) events
FIG. Memory issue: large heap size while building the call stack with millions of (allocation) events
JFR Reader: Iterator pattern

FIG. Process RecordEvent one by one
FIG. Memory issue: large heap size while building the call stack with millions of (allocation) events
FIG. Performance issue: most time spent on building frame names
JFR Reader: use raw references

FIG. use references instead of materialized stack trace

```java
public class StackTrace {
    // 方法ID
    public final long[] methods;
    // 每个byte表示对应的方法类型，有INTERPRETED, JIT компilled等
    public final byte[] types;
    // 每个int表示方法所在的行号和bci
    public final int[] locations;
    // ...
}
```
JFR Reader: use raw references

Through JDK native way reading, 240 million events share 30,000 stacktraces.

Through asyncio profiler reading, 240 million events share 30,000 stacktraces.

FIG. 2,000,000 alloc events share 30,000 stacktraces
FIG. Another performance issue: too many binary searches during insertion even if binary search has $O(\log N)$ complexity
JFR Reader: aggregate first

Event1的stackTraceld

Event2的stackTraceld

Event3的stackTraceld

Event4的stackTraceld

Event5的stackTraceld

Map存放中间结果stackTraceld-value(5)

根据id获取

stackTrace

List<String>stackStrings

插入,value=5

Tree
Final result: use <100M heap, and finish parsing <1 second
JFR Reader: What about large JFR file?
FIG. ChunkSize can be controlled by parameter
JFR Reader: What about large JFR file?

Support read JFR file chunk by chunk #718

lujiajing1126 wants to merge 2 commits into async-profiler:master from lujiajing1126:master

closed

lujiajing1126 commented on Feb 23 · edited

If the JFR file is large, for example, alloc event is enabled, it may cost large heap space to process millions of events.

This PR intends to amortize memory consumption by allowing users to read a single chunk once.

API:

As is used by readAllEvents in the JfrReader.java.

```java
public <E extends Event> List<E> readAllEvents(Class<E> cls) {
    Chunks<E> chunks = readChunks(cls);
    ArrayList<E> events = new ArrayList<>();
    for (final Chunk<E> chunk : chunks) {
        for (final E event : chunk) {
            events.add(event);
        }
    }
    Collections.sort(events);
    return events;
}
```

Still questions: (Excuse for my poor understanding of the JFR spec)

In the current impl, I noticed only types and `@typesByName` are cleared. However, according to the file format, chunk should be self-contained. Does it mean that we can clear all intermediate states, e.g. classes, symbols, methods when we start to read a new Chunk?
One more thing: correlation

Context ID functionality #576

Adding two operations to Java API:

- `setContextId(long contextId)`
- `clearContextId()`

Use case:
There are Java applications that want to achieve better performance by distributing the single request work into multiple threads. In such cases it is hard to find in profiler results where the time is wasted, since you have no information which thread executed which request. To make it traceable I would like to use external correlation id, so it is generated by client before passing work to other threads, then the worker thread would do:

```java
asyncProfiler.setContextId(correlationId);
actualWork();
asyncProfiler.clearContextId();
```

The context id is passed to custom field on execution sample, so we can post-filer it.

Other use cases I see is reactive programming, and in the future, loom project, distributed systems...

This PR is not finished, I just want to know, what do you think @spangin? If you like that functionality I can add this field to other profiling event.
交互式诊断
Interactive Diag.
How to diag. a CPU spike

有一个节点 cpu 和 young gc 次数 遥遥领先，看起来很奇怪
How to diag. a CPU spike: Arthas

支持一键展示当前最忙的前 N 个线程并打印堆栈:

```
$ thread -n 3
```

```
"CL CompilerThread0" [Internal] cpuUsage=0.11% deltaTime=0ms time=401ms RUNNABLE
  at java.management@11.0.7/sun.management.ThreadImpl.dumpThreads0(Native Method)
  at java.management@11.0.7/sun.management.ThreadImpl.getThreadInfo(ThreadImpl.j)
  at com.taobao.arthas.core.command.monitor.ThreadCommand.processTopBusyThread
  at com.taobao.arthas.core.command.monitor.ThreadCommand.<init>
```

```
"arathas-command-execute" Id=23 cpuUsage=0.11% deltaTime=0ms time=401ms RUNNABLE
  at java.management@11.0.7/sun.management.ThreadImpl.dumpThreads0(Native Method)
  at java.management@11.0.7/sun.management.ThreadImpl.getThreadInfo(ThreadImpl.j)
  at com.taobao.arthas.core.command.monitor.ThreadCommand.processTopBusyThread
  at com.taobao.arthas.core.command.monitor.ThreadCommand.<init>
```

```
"VM Periodic Task Thread" [Internal] cpuUsage=0.07% deltaTime=0ms time=584ms
  at java.base@11.0.7/java.lang.Thread.run(Thread.java:834)
```

How to integrate SkyWalking with Arthas

FIG. 将 Apache SkyWalking 与 Arthas 集成 By 魏翔
https://skywalking.apache.org/zh/2023-09-17-integrating-skywalking-with-arthas/
How to integrate SkyWalking with Arthas

- For those commands that does not need bytecode retransform
  - thread
  - ...
- Bypass Storage: latency sensitive
  - ElasticSearch flush interval
  - Agents poll commands from OAP: scheduled per 20 seconds

1. Connect via WebSocket

Diagram:

- OAP server
- SW Java Agent
- RemoteCommand Executor
- Container
- JVM
- Bidi gRPC
- 1. Connect via WebSocket
- 4. request
- 5. response
Protocol Design: bidi over unary

```java
service ProfileTask {

    // query all sniffer need to execute profile task commands
    rpc getProfileTaskCommands (ProfileTaskCommandQuery) returns (common.v1.Commands) {
    }

    // collect dumped thread snapshot
    rpc collectSnapshot (stream ThreadSnapshot) returns (common.v1.Commands) {
    }

    // report profiling task finished
    rpc reportTaskFinish (ProfileTaskFinishReport) returns (common.v1.Commands) {
    }

}

service RemoteCommandTask {

    // collect remote command result
    rpc executeRemoteCommand (stream RemoteCommandRequest) returns (stream RemoteCommandResponse) {
    }
}
```
What about distributed OAP?
What about retransform?


**Problem**

1. When using the SkyWalking agent, some other agents, such as Arthas, can’t work properly. #4858

2. The retransform classes in the Java agent conflict with the SkyWalking agent, as illustrated in this demo

**Cause**

The SkyWalking agent uses ByteBuddy to transform classes when the Java application starts. ByteBuddy generates auxiliary classes with different random names every time.

When another Java agent retransforms the same class, it triggers the SkyWalking agent to enhance the class again. Since the bytecode has been regenerated by ByteBuddy, the fields and imported class names have been modified, and the JVM verifications on class bytecode have failed, the retransform classes would therefore be unsuccessful.

**Resolution**

1. Enable the class cache feature

Add JVM parameters:

```
-Dskywalking.agent.is_cache_enhanced_class=true -Dskywalking.agent.class_cache_mode=MEMORY
```
What about retransform? Changes in 9.0.3

• For those commands that does need bytecode retransform,
  • watch: observe method exec (parameter, result, exception...)
  • trace: trace method exec path
  • monitor: stat method exec (not real time)

• Main idea
  • For TypeDescription: always prefer bytecode from TypePool to reflection API
  • For aux. fields/methods: use stable prefix/suffix instead of random ones
Changes in 9.0: perf issue (resolved)

FIG. Using POOL_FIRST TypeDescription strategy in SW Java 9.0 caused almost double application launch time and larger heap size. Resolved in PR #637.
欢迎您提问交流
（仅限2位提问）
2023 • SkyWalking Summit

感谢您的观看