凌晨四点，线上CPU告警，绩效没了……

dbaplus社群 2025年04月07日 07:15 广东

前言

凌晨4点，我被一阵刺耳的手机铃声惊醒。迷迷糊糊地摸索着手机，屏幕上赫然显示着"线上CPU告警"的字样。瞬间，我的困意全无，取而代之的是一阵冷汗和心跳加速。作为公司核心系统的负责人，我深知这意味着什么——用户体验受损、可能的数据丢失，更糟糕的是，我的年终绩效可能就此化为泡影。

我迅速起身，开始了一场与时间赛跑的故障排查之旅。

一、初步诊断：快速定位问题

首先，我登录了服务器，使用top命令查看系统资源使用情况：

```
top
```

输出显示CPU使用率接近100%，load average远超服务器核心数。这确实是一个严重的问题。

接下来，我使用htop命令获取更详细的进程信息：

```
$ htop
```

我发现有几个Java进程占用了大量CPU资源。这些进程正是我们的核心服务。

二、JVM层面分析：寻找热点方法

确定了问题出在Java应用上，我开始进行JVM层面的分析。首先使用jstat命令查看GC情况：

```
$ jstat -gcutil [PID] 1000 10
```

输出显示Full GC频繁发生，这可能是导致CPU使用率高的原因之一。

接着，我使用jstack命令生成线程转储，查看线程状态：

```
$ jstack [PID] > thread_dump.txt
```

分析thread dump文件，我发现大量线程处于RUNNABLE状态，执行着相似的方法调用。

为了进一步定位热点方法，我使用了async-profiler工具：

```
$ ./profiler.sh -d 30 -f cpu_profile.svg [PID]
```

生成的火焰图清晰地显示了一个自定义的排序算法占用了大量CPU时间。

三、应用层面优化：重构算法

找到了罪魁祸首，我立即查看了相关代码。这是一个用于大量数据的自定义排序算法，原本设计用于小规模数据，但随着业务增长，它的性能问题暴露无遗。

我迅速重构了算法，使用Java 8的并行流进行优化：

```
List<Data> sortedData = data.parallelStream()
    .sorted(Comparator.comparing(Data::getKey))
    .collect(Collectors.toList());
```

同时，我添加了缓存机制，避免重复计算：

```
@Cacheable("sortedData")
public List<Data> getSortedData() {
```

```
    // 优化后的排序逻辑
}
```

四、数据库优化：索引与查询改进

在排查过程中，我还发现了一些低效的数据库查询。使用explain命令分析SQL语句：

```
EXPLAIN SELECT * FROM large_table WHERE status = 'ACTIVE';
```

结果显示这个查询导致了全表扫描。我立即添加了合适的索引：

```
CREATE INDEX idx_status ON large_table(status);
```

并重写了部分ORM查询，使用更高效的原生SQL：

```
@Query(value = "SELECT * FROM large_table WHERE status = :status", nat
List<LargeTable> findByStatus(@Param("status") String status);
```

五、部署优化：资源隔离

为了防止单个服务影响整个系统，我决定使用Docker进行资源隔离。创建了如下的Dockerfile：

```
FROM openjdk:11-jre-slim
COPY target/myapp.jar app.jar
ENTRYPOINT ["java", "-Xmx2g", "-jar", "/app.jar"]
```

并使用Docker Compose进行服务编排，限制了CPU和内存使用：

```
version: '3'
services:
  myapp:
```

```
    build: .
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 512M
```

六、监控告警：防患未然

最后，为了避免类似问题再次发生，我升级了监控系统。使用Prometheus和Grafana搭建了全面的监控平台，并设置了更加智能的告警规则：

```
- alert: HighCPUUsage
  expr: 100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idl
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "High CPU usage detected"
    description: "CPU usage is above 80% for more than 5 minutes"
```

结语：危机与成长

经过近4小时的奋战，系统终于恢复了正常。CPU使用率降到了30%以下，服务响应时间也恢复到了毫秒级。

这次经历让我深刻意识到，在追求业务快速发展的同时，我们不能忽视技术债务的累积。定期的代码审查、性能测试和压力测试是必不可少的。同时，建立完善的监控和告警机制，能够帮助我们更快地发现和解决问题。

虽然这次事件可能会影响我的年终绩效，但它带给我的经验和教训是无价的。持续学习和改进永远是我们的必修课。

凌晨的阳台上，我望着渐亮的天空，心中暗自庆幸：又一次化险为夷。但我知道，明天将是新的挑战，我们还有很长的路要走。

作者 | JustinNeil

来源丨网址：https://juejin.cn/post/7424522247791247394

dbaplus社群欢迎广大技术人员投稿，投稿邮箱：editor@dbaplus.cn