目录

[1 Liunx 问题分析](#_Toc31013_WPSOffice_Level1) [3](#_Toc31013_WPSOffice_Level1)

[1.1 CPU](#_Toc21661_WPSOffice_Level2) [3](#_Toc21661_WPSOffice_Level2)

[1.2 内存](#_Toc11792_WPSOffice_Level2) [3](#_Toc11792_WPSOffice_Level2)

[1.3 IO](#_Toc5091_WPSOffice_Level2) [3](#_Toc5091_WPSOffice_Level2)

[1.4 网络](#_Toc27409_WPSOffice_Level2) [3](#_Toc27409_WPSOffice_Level2)

[1.5 综合](#_Toc28167_WPSOffice_Level2) [3](#_Toc28167_WPSOffice_Level2)

[2 分布式算法](#_Toc21661_WPSOffice_Level1) [3](#_Toc21661_WPSOffice_Level1)

[2.1 一致性算法](#_Toc9304_WPSOffice_Level2) [3](#_Toc9304_WPSOffice_Level2)

[2.2 共识算法](#_Toc31158_WPSOffice_Level2) [3](#_Toc31158_WPSOffice_Level2)

[2.3 加密算法](#_Toc11992_WPSOffice_Level2) [4](#_Toc11992_WPSOffice_Level2)

[2.3.1 hash算法](#_Toc21661_WPSOffice_Level3) [4](#_Toc21661_WPSOffice_Level3)

[2.3.2 加密算法](#_Toc11792_WPSOffice_Level3) [4](#_Toc11792_WPSOffice_Level3)

[2.3.3 加密证书](#_Toc5091_WPSOffice_Level3) [4](#_Toc5091_WPSOffice_Level3)

[2.3.4 零和证明](#_Toc27409_WPSOffice_Level3) [4](#_Toc27409_WPSOffice_Level3)

[2.4 存储算法](#_Toc21810_WPSOffice_Level2) [4](#_Toc21810_WPSOffice_Level2)

[2.4.1 merkle树](#_Toc28167_WPSOffice_Level3) [4](#_Toc28167_WPSOffice_Level3)

[3 JAVA 基础](#_Toc11792_WPSOffice_Level1) [4](#_Toc11792_WPSOffice_Level1)

[3.1 基础](#_Toc8341_WPSOffice_Level2) [4](#_Toc8341_WPSOffice_Level2)

[3.2 进阶](#_Toc5133_WPSOffice_Level2) [4](#_Toc5133_WPSOffice_Level2)

[3.3 安全](#_Toc17903_WPSOffice_Level2) [5](#_Toc17903_WPSOffice_Level2)

[3.4 性能](#_Toc9175_WPSOffice_Level2) [5](#_Toc9175_WPSOffice_Level2)

[3.5 扩展](#_Toc9764_WPSOffice_Level2) [5](#_Toc9764_WPSOffice_Level2)

[4 JAVA虚拟机](#_Toc5091_WPSOffice_Level1) [5](#_Toc5091_WPSOffice_Level1)

[4.1 基本原理](#_Toc10011_WPSOffice_Level2) [5](#_Toc10011_WPSOffice_Level2)

[4.2 高效实现](#_Toc14543_WPSOffice_Level2) [5](#_Toc14543_WPSOffice_Level2)

[4.3 代码优化](#_Toc347_WPSOffice_Level2) [5](#_Toc347_WPSOffice_Level2)

[4.4 黑科技](#_Toc15473_WPSOffice_Level2) [5](#_Toc15473_WPSOffice_Level2)

[5 架构](#_Toc27409_WPSOffice_Level1) [6](#_Toc27409_WPSOffice_Level1)

[5.1 高性能](#_Toc25587_WPSOffice_Level2) [6](#_Toc25587_WPSOffice_Level2)

[5.2 高可用](#_Toc24143_WPSOffice_Level2) [6](#_Toc24143_WPSOffice_Level2)

[5.3 可扩展性](#_Toc27400_WPSOffice_Level2) [6](#_Toc27400_WPSOffice_Level2)

[5.4 综述](#_Toc9763_WPSOffice_Level2) [6](#_Toc9763_WPSOffice_Level2)

[5.4.1 设计原则](#_Toc9304_WPSOffice_Level3) [6](#_Toc9304_WPSOffice_Level3)

[5.4.2 设计流程](#_Toc31158_WPSOffice_Level3) [6](#_Toc31158_WPSOffice_Level3)

[5.4.3 如何重构](#_Toc11992_WPSOffice_Level3) [6](#_Toc11992_WPSOffice_Level3)

[6 网络](#_Toc28167_WPSOffice_Level1) [6](#_Toc28167_WPSOffice_Level1)

[6.1 综述](#_Toc4845_WPSOffice_Level2) [6](#_Toc4845_WPSOffice_Level2)

[6.2 物理层和mac层](#_Toc27764_WPSOffice_Level2) [7](#_Toc27764_WPSOffice_Level2)

[6.3 传输层](#_Toc20894_WPSOffice_Level2) [7](#_Toc20894_WPSOffice_Level2)

[6.4 应用层](#_Toc6291_WPSOffice_Level2) [7](#_Toc6291_WPSOffice_Level2)

[6.5 数据中心](#_Toc15691_WPSOffice_Level2) [7](#_Toc15691_WPSOffice_Level2)

[6.6 云&网络](#_Toc2954_WPSOffice_Level2) [7](#_Toc2954_WPSOffice_Level2)

[6.7 容器&网络](#_Toc12298_WPSOffice_Level2) [7](#_Toc12298_WPSOffice_Level2)

[6.8 微服务&网络](#_Toc1923_WPSOffice_Level2) [7](#_Toc1923_WPSOffice_Level2)

# 1 Liunx 问题分析

## CPU

|  |
| --- |
| **CPU问题的排查思路** |
| 1 工具  top  pidstat  uptime  mpstat |
| 2 参数介绍  用户（%user）、Nice（%nice）、系统（%system） 、等待 I/O（%iowait） 、中断（%irq）以及软中断（%softirq）  •用户 CPU 和 Nice CPU 高，说明用户态进程占用了较多的 CPU，所以应该着重排查进程的性能问题。  •系统 CPU 高，说明内核态占用了较多的 CPU，所以应该着重排查内核线程或者系统调用的性能问题。  •I/O 等待 CPU 高，说明等待 I/O 的时间比较长，所以应该着重排查系统存储是不是出现了 I/O 问题。  •软中断和硬中断高，说明软中断或硬中断的处理程序占用了较多的 CPU，所以应该着重排查内核中的中断服务程序。 |
| 3 原理篇  1) CPU结构及运行基本原理  2) 上下文切换含义和原理  3) 进程上下文切换、线程上下文切换、中断上下文切换 |

|  |
| --- |
| 各个指标 |
| **1 CPU 使用率**  •用户 CPU 使用率:包括用户态 CPU 使用率（user）和低优先级用户态 CPU 使用率（nice），表示 CPU 在用户态运行的时间百分比。用户 CPU 使用率高，通常说明有应用程序比较繁忙。  •系统 CPU 使用率:表示 CPU 在内核态运行的时间百分比（不包括中断）。系统 CPU 使用率高，说明内核比较繁忙。  •等待 I/O 的 CPU 使用率:通常也称为 iowait，表示等待 I/O 的时间百分比。iowait 高，通常说明系统与硬件设备的 I/O 交互时间比较长。  •软中断、硬中断的 CPU 使用率:分别表示内核调用软中断处理程序、硬中断处理程序的时间百分比。它们的使用率高，通常说明系统发生了大量的中断。  •窃取 CPU 使用率（steal）:被其他虚拟机占用的 CPU 时间百分比  •客户 CPU 使用率（guest）:运行客户虚拟机的 CPU 时间百分比  **2 平均负载**  系统的平均活跃进程数。它反应了系统的整体负载情况，主要包括三个数值，分别指过去 1 分钟、过去 5 分钟和过去 15 分钟的平均负载。  理想情况下，平均负载等于逻辑 CPU 个数，这表示每个 CPU 都恰好被充分利用。如果平均负载大于逻辑 CPU 个数，就表示负载比较重了。  **3 上下文切换**  上下文切换，本身是保证 Linux 正常运行的一项核心功能。但过多的上下文切换，会将原本运行进程的 CPU 时间，消耗在寄存器、内核栈以及虚拟内存等数据的保存和恢复上，缩短进程真正运行的时间，成为性能瓶颈。  **4 CPU缓存命中率**  CPU 缓存的命中率。由于 CPU 发展的速度远快于内存的发展，CPU 的处理速度就比内存的访问速度快得多。这样，CPU 在访问内存的时候，免不了要等待内存的响应。为了协调这两者巨大的性能差距，CPU 缓存（通常是多级缓存）就出现了。 |

|  |
| --- |
| CPU性能工具 |
|  |
|  |

|  |
| --- |
| CPU性能排查思路 |
|  |

|  |
| --- |
| **CPU优化思路** |
| **性能优化方法论**  1 怎么评估性能优化的效果？  --检测量化  2 多个性能问题同时存在，要怎么选择？  --不是所有指标都要优化，二八原则  3 有多种优化方法时，要如何选择?  --平衡效果与投入成本 |
| **应用程序优化**  **1 编译器优化**  很多编译器都会提供优化选项，适当开启它们，在编译阶段你就可以获得编译器的帮助，来提升性能。比如， gcc 就提供了优化选项 -O2，开启后会自动对应用程序的代码进行优化。  **2 算法优化**  使用复杂度更低的算法，可以显著加快处理速度。比如，在数据比较大的情况下，可以用 O(nlogn) 的排序算法（如快排、归并排序等），代替 O(n^2) 的排序算法（如冒泡、插入排序等）。  **3 异步处理**  使用异步处理，可以避免程序因为等待某个资源而一直阻塞，从而提升程序的并发处理能力。比如，把轮询替换为事件通知，就可以避免轮询耗费 CPU 的问题。  **4 多线程代替多进程**  前面讲过，相对于进程的上下文切换，线程的上下文切换并不切换进程地址空间，因此可以降低上下文切换的成本。  **5 善用缓存**  经常访问的数据或者计算过程中的步骤，可以放到内存中缓存起来，这样在下次用时就能直接从内存中获取，加快程序的处理速度。 |
| **系统优化**  **1 CPU 绑定**  把进程绑定到一个或者多个 CPU 上，可以提高 CPU 缓存的命中率，减少跨 CPU 调度带来的上下文切换问题。  **2 CPU 独占**  跟 CPU 绑定类似，进一步将 CPU 分组，并通过 CPU 亲和性机制为其分配进程。这样，这些 CPU 就由指定的进程独占，换句话说，不允许其他进程再来使用这些 CPU。  **3 优先级调整**  使用 nice 调整进程的优先级，正值调低优先级，负值调高优先级。优先级的数值含义前面我们提到过，忘了的话及时复习一下。在这里，适当降低非核心应用的优先级，增高核心应用的优先级，可以确保核心应用得到优先处理。  **4 为进程设置资源限制**  使用 Linux cgroups 来设置进程的 CPU 使用上限，可以防止由于某个应用自身的问题，而耗尽系统资源。  **5 NUMA（Non-Uniform Memory Access）优化**  支持 NUMA 的处理器会被划分为多个 node，每个 node 都有自己的本地内存空间。NUMA 优化，其实就是让 CPU 尽可能只访问本地内存。  **6 中断负载均衡**  无论是软中断还是硬中断，它们的中断处理程序都可能会耗费大量的 CPU。开启 irqbalance 服务或者配置 smp\_affinity，就可以把中断处理过程自动负载均衡到多个 CPU 上。 |
| **优化避免**  **1 过早优化是万恶之源**  **2 不要过度优化** |

## 内存

## IO

## 网络

## 综合

# 2 分布式算法

## 2.1 一致性算法

## 2.2 共识算法

## 2.3 加密算法

### 2.3.1 hash算法

### 2.3.2 加密算法

### 2.3.3 加密证书

### 2.3.4 零和证明

## 2.4 存储算法

### 2.4.1 merkle树

# 3 JAVA 基础

## 3.1 基础

## 3.2 进阶

## 3.3 安全

## 3.4 性能

## 3.5 扩展

# 4 JAVA虚拟机

## 4.1 基本原理

## 4.2 高效实现

## 4.3 代码优化

## 4.4 黑科技

# 5 架构

## 5.1 高性能

## 5.2 高可用

## 5.3 可扩展性

## 5.4 综述

### 5.4.1 设计原则

### 5.4.2 设计流程

### 5.4.3 如何重构

# 6 网络

## 6.1 综述

## 6.2 物理层和mac层

## 6.3 传输层

## 6.4 应用层

## 6.5 数据中心

## 6.6 云&网络

## 6.7 容器&网络

## 6.8 微服务&网络