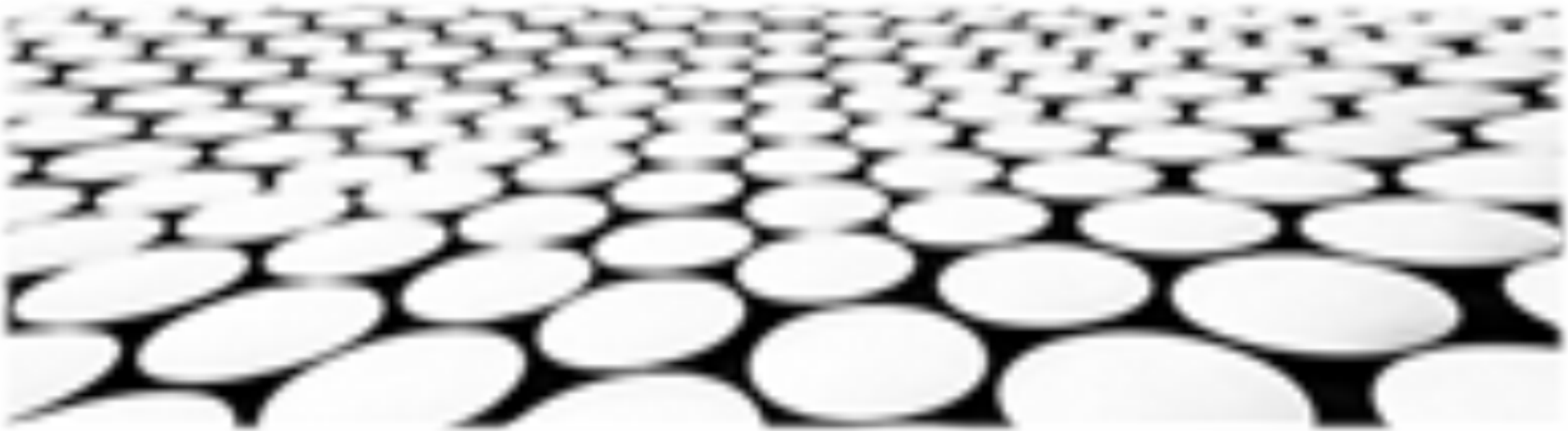
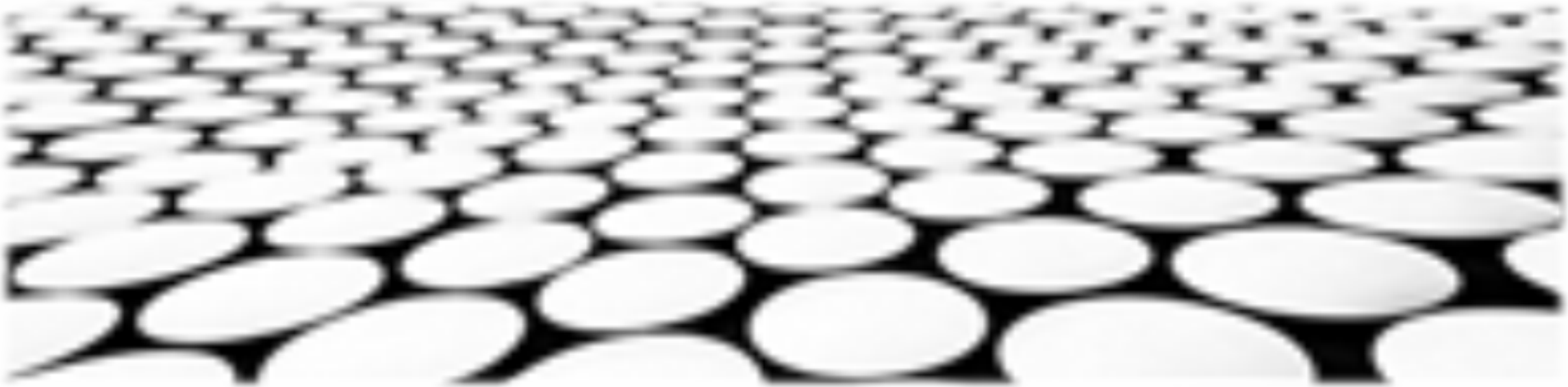
多核架构的线程调度优化



目录页

Contents Page

抢占式调度与非抢占式调度对比



抢占式调度与非抢占式调度对比

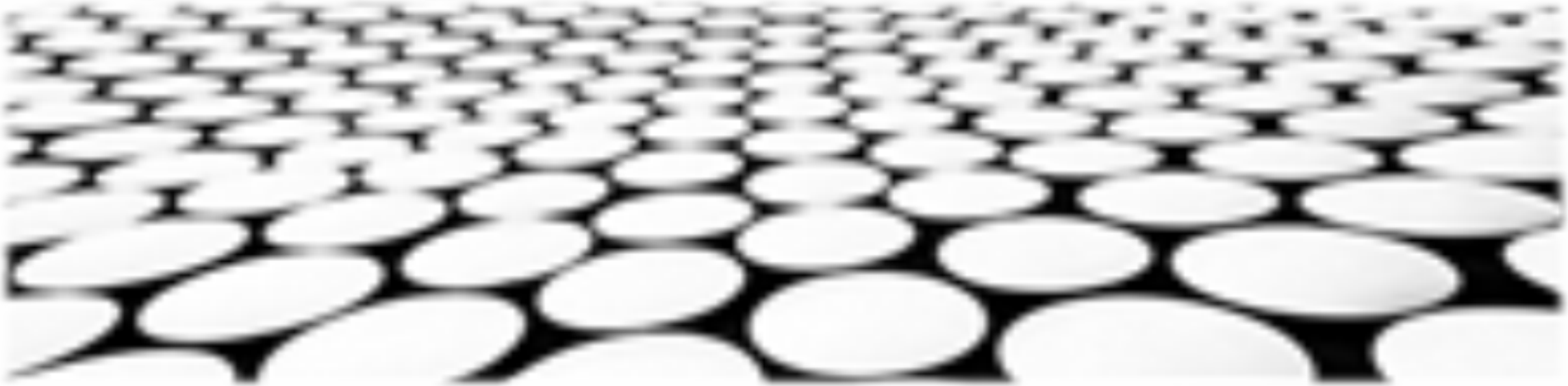
抢占式调度

1. 允许新到达的高优先级线程立即抢占运行中的低优先级线程，提升了系统响应速度。
2. 抢占发生时，需要保存当前线程的上下文信息，并加载新线程的上下文信息，导致额外的开销。
3. 在实时系统中广泛应用，可确保关键任务及时执行。

非抢占式调度

1. 当前线程持续执行直至完成或主动放弃 CPU，不会被新到达的线程抢占。
2. 避免了抢占式调度的开销，适用于开销敏感的系统。
3. 在某些情况下，非抢占式调度可能导致优先级反转，即低优先级线程长期占用 CPU。

线程优先级设置与管理



线程优先级设置与管理

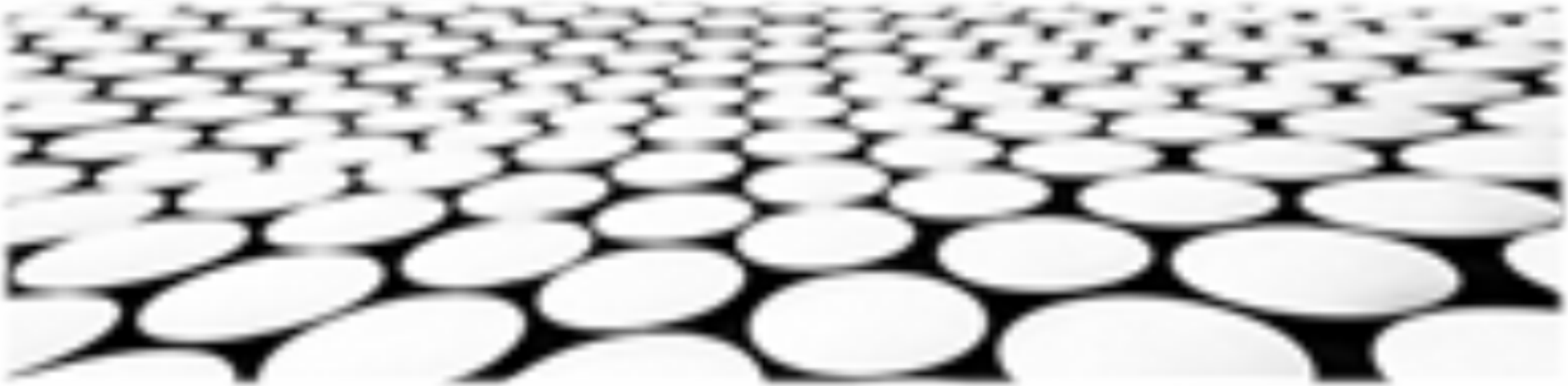
线程优先级设置

- 优先级等级划分：线程调度算法通常将优先级划分为多个等级，例如低、中、高或自定义范围。不同优先级的线程具有不同响应时间，高优先级线程会在较短时间内被调度执行。
- 优先级继承：当一个高优先级线程调用低优先级线程时，低优先级线程会继承高优先级，以确保高优先级线程可以及时完成任务。
- 优先级动态调整：随着系统负载和应用需求的变化，线程的优先级可能会动态调整。调度器可以根据某些规则或事件触发器调整优先级，以优化系统性能。

线程优先级管理

- 优先级反转预防：当低优先级线程持有高优先级线程所需的资源时，可能会发生优先级反转。调度器必须采用机制，例如优先级继承或锁定优先级继承，以防止这种反转。
- 优先级饥饿避免：调度器应确保所有线程在一段时间内都有机会执行，避免低优先级线程长期处于饥饿状态。可以采用轮询调度或优先级队列等技术来避免饥饿。
- 优先级老化：随着时间推移，低优先级线程可能会被高优先级线程持续阻塞。调度器可以通过优先级老化机制，逐渐提升长时间未执行线程的优先级，以防止老化线程长期处于阻塞状态。

亲和度调度与负载均衡



亲和度调度与负载均衡

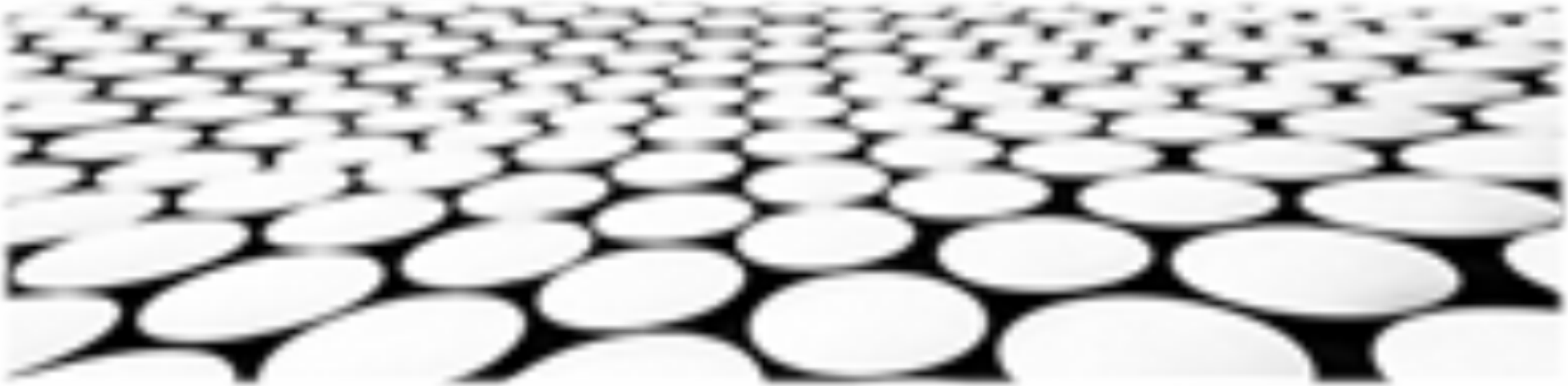
亲和度调度

1. 将线程分配到与其数据或计算需求最相近的处理核上，从而减少缓存未命中和内存访问延迟。
2. 可通过跟踪线程对特定核心的使用情况或通过将线程分组到基于共享数据的不同核心里来实现。
3. 对于高度局部化和数据密集型应用程序特别有效，因为它们可以最小化线程之间对共享资源的竞争。

负载均衡

1. 确保所有处理器核心的工作负载均匀分布，从而最大化系统资源利用率并最小化等待时间。
2. 可通过测量每个内核的利用率并相应地重新分配线程来实现。

NUMA感知的线程调度



NUMA感知的线程调度

NUMA感知的线程调度

1. 处理器亲和性：任务被分配到与正在访问的数据位于同一NUMA节点的处理器上，减少跨节点数据访问造成的延迟。
2. 内存亲和性：线程优先调度到最近访问的内存区域所在NUMA节点的处理器上，优化内存访问速度。
3. NUMA感知算法：优化调度算法，考虑NUMA拓扑结构，如First-Touch原则和局部优先调度策略。

负载平衡的优化

1. 局部负载平衡：在单个NUMA节点内平衡线程负载，避免热点问题和资源争用。
2. 全局负载平衡：在不同NUMA节点之间平衡负载，优化资源利用率和系统吞吐量。
3. 动态负载调节：根据系统运行时状态，动态调整线程分配和调度策略，适应不断变化的负载情况。