基于功能复用构建高性价比分布式系统

——以分布式数据复制为例

陈榕

并行与分布式系统研究所 · 上海交通大学

2020.12 华为·松山湖

Joint work with Haibo, Xingda, Sijie and other members in IPADS

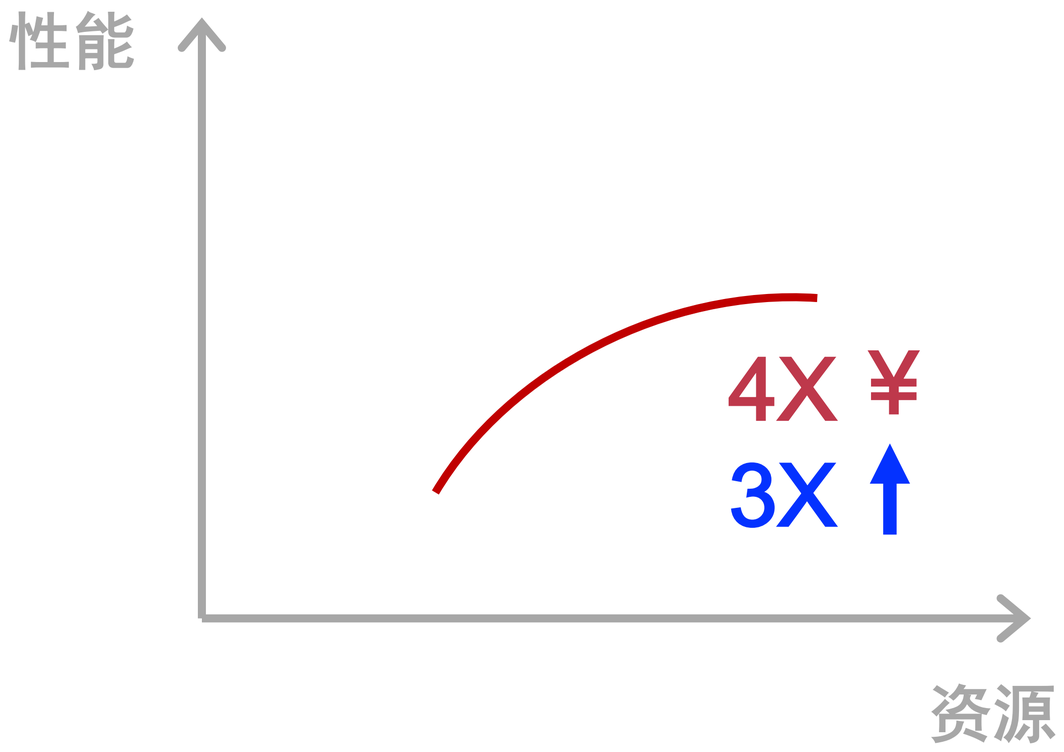
提升系统“性能”的通常方法

简单堆积更多的硬件资源

但性价比可能反而下降

例如：最新TPC-C世界记录

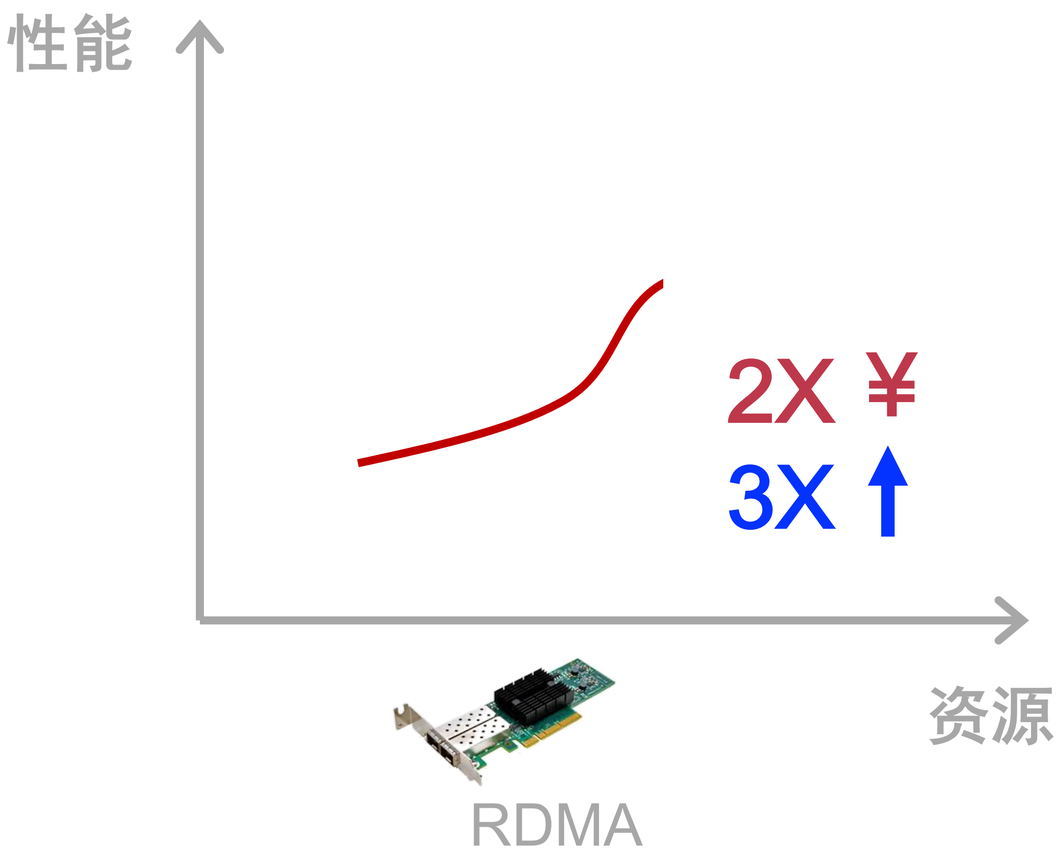
- ✓ 峰值吞吐量达到1,179万事务每秒
- ✓ 1557台虚拟机（65,394个虚拟核）
- ✓ 系统售价28亿人名币



提升系统“性价比”的方法：兑现硬件红利

新硬件在某一方面的性能/功能上带来巨大突破

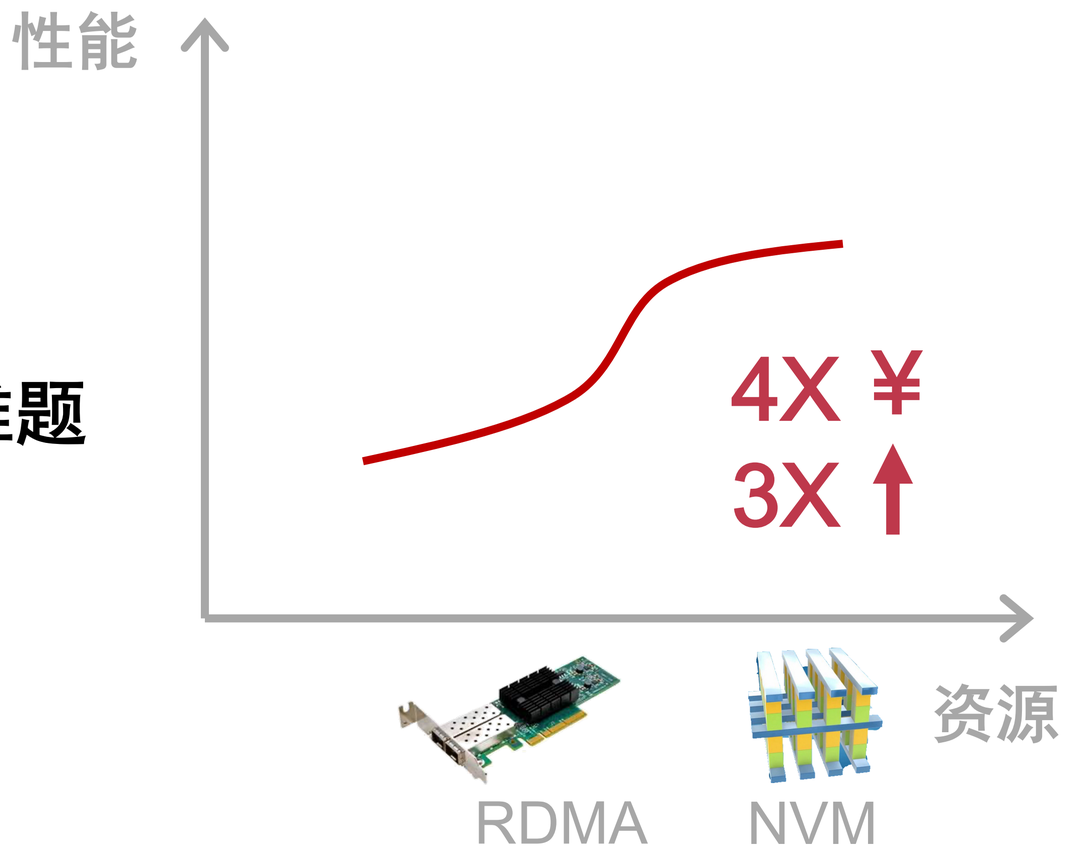
- RDMA网络：**微秒级**跨节点访存能力
- NVM存储：**内存级**持久化存储能力



提升系统“性价比”的方法：兑现硬件红利

新硬件在某一方面的性能/功能上带来巨大突破

- RDMA网络：**微秒级**跨节点访存能力
- NVM存储：**内存级**持久化存储能力



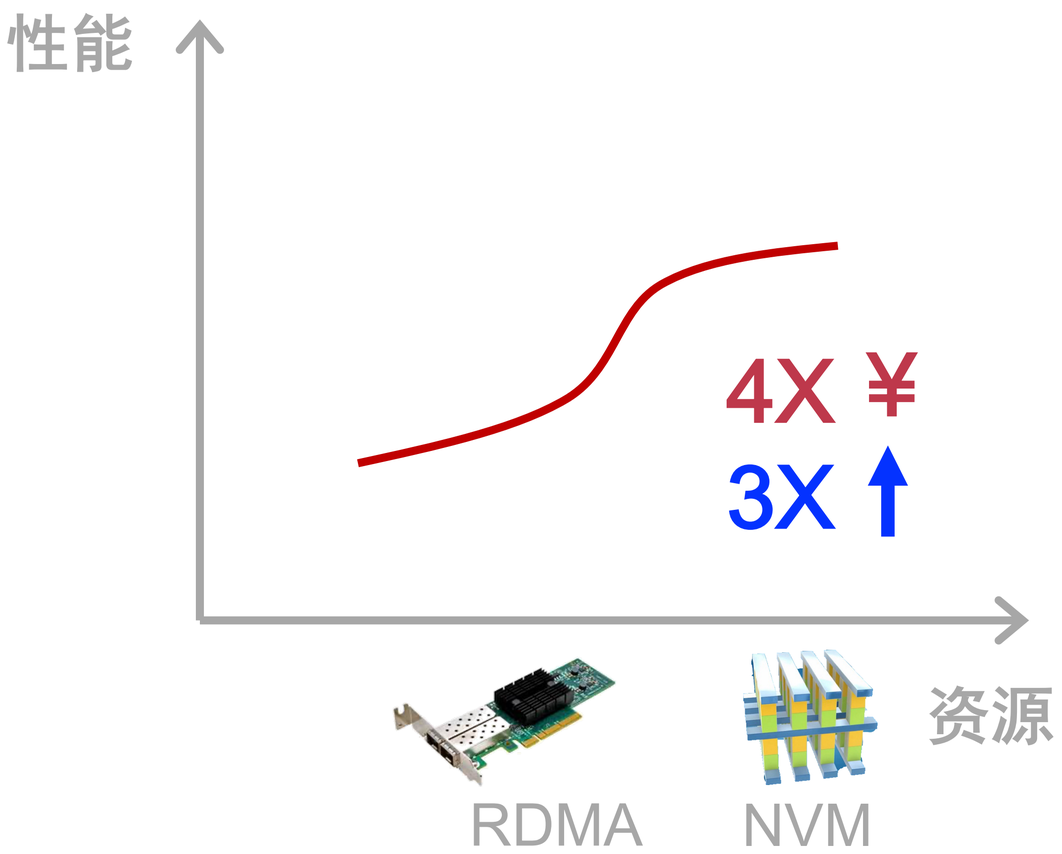
但同时利用好多种新硬件资源是一个难题

- 普遍存储“1+1<2”现象，甚至不兼容
  比如：RDMA、NVM、HTM

提升系统“性价比”的方法：兑现硬件红利

现有设计无法完全兑现新硬件红利

- ✗ 现有设计未考虑新特性
- ✗ 额外的系统级支撑开销
- ✗ 硬件功能单一，使用受限



提升系统“性价比”的方法：兑现硬件红利

现有设计无法完全兑现新硬件红利

✗ 现有设计未考虑新特性

✗ 额外的系统级支撑开销

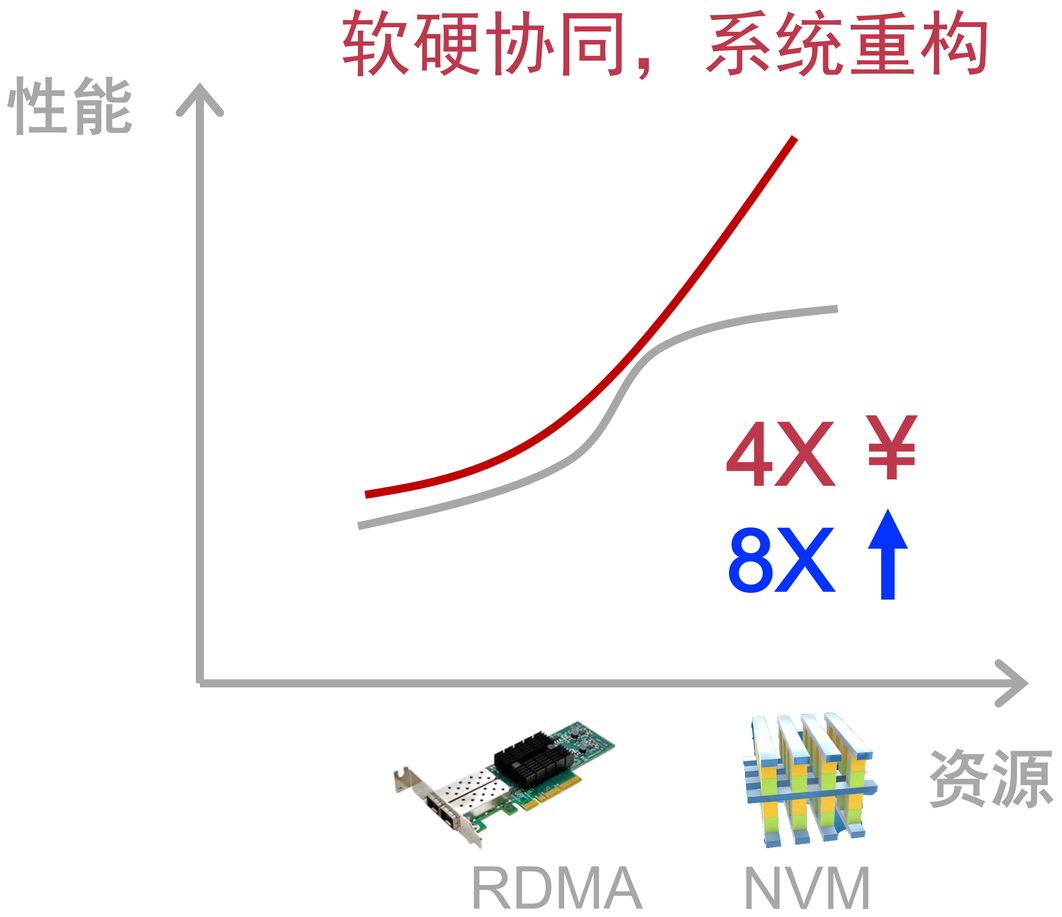
✗ 硬件功能单一，使用受限

极致性价比往往需要软硬件协同设计

① 以持久化内存为中心的新型存储系统

② 基于NVM的高效可靠存储技术

③ 新型异构硬件的协同优化技术



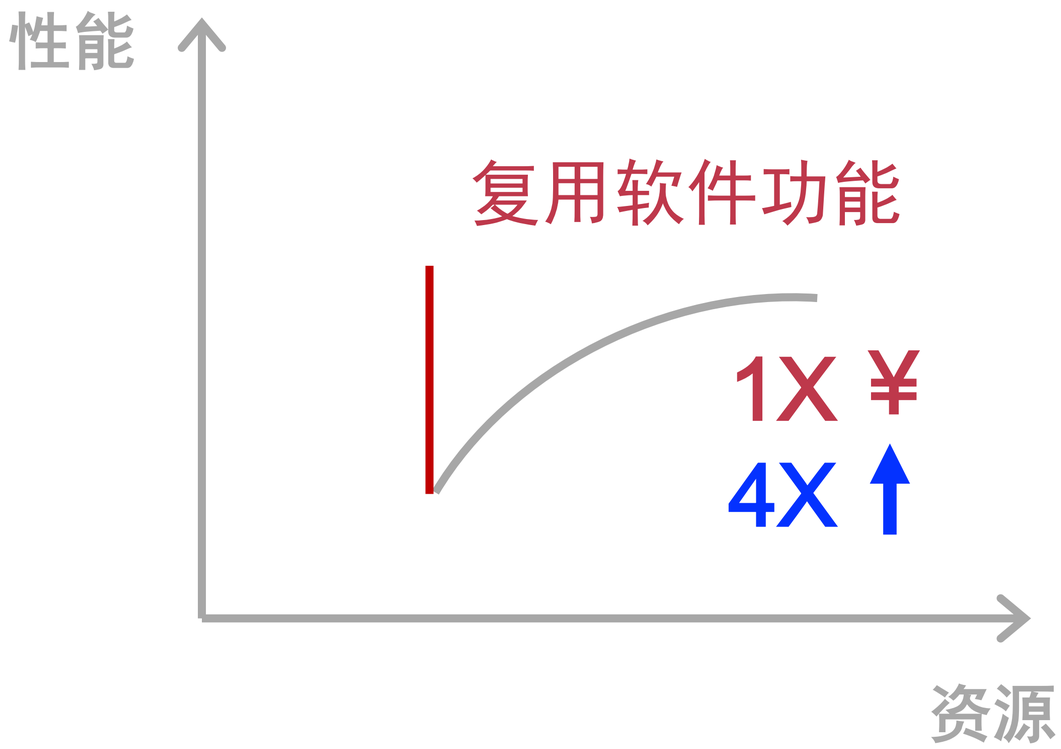
提升系统“性价比”的方法：复用软件功能

提升性价比：代价不变，性能（功能）提升

► 更多的必要**功能** = 更高的整体**性能**

极致场景：软件功能复用

► （接近）零投入



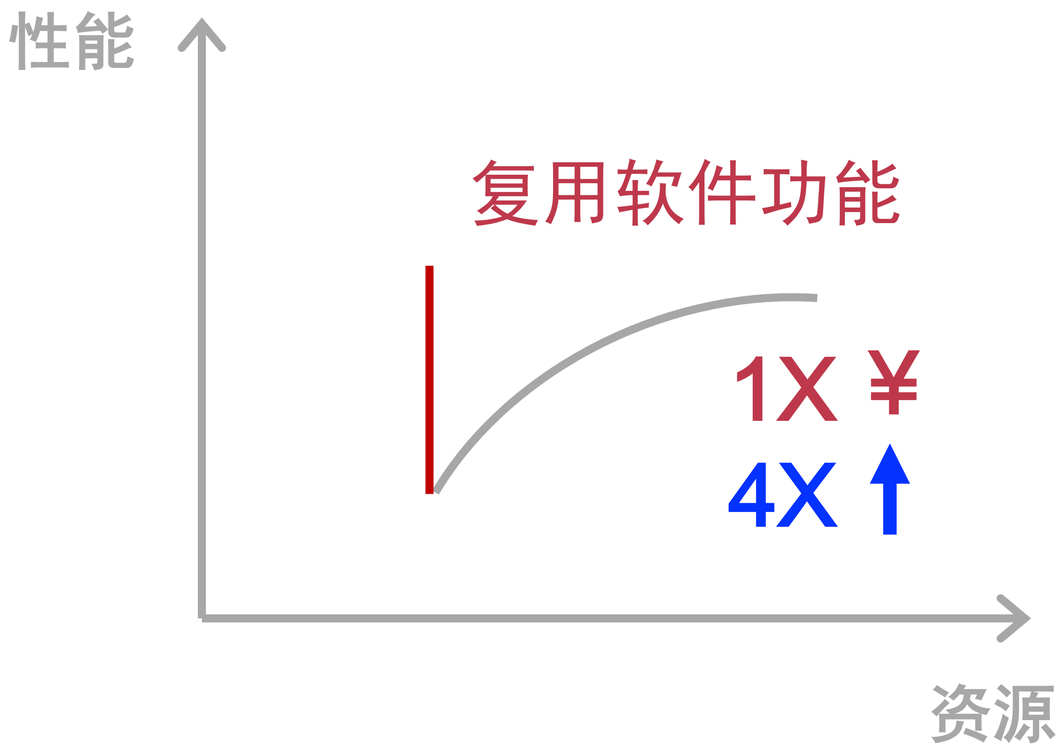
提升系统“性价比”的方法：复用软件功能

提升性价比：代价不变，性能（功能）提升

- 更多的必要**功能** = 更高的整体**性能**

极致场景：软件功能复用

- （接近）零投入
- 早饭（Breakfast）
  + 午饭（Lunch）
  = 早午饭（Brunch）



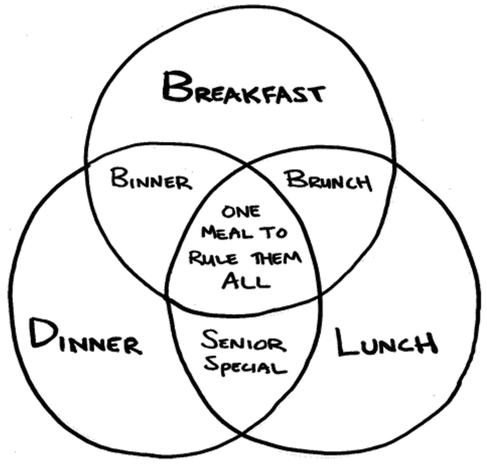
提升系统“性价比”的方法：复用软件功能

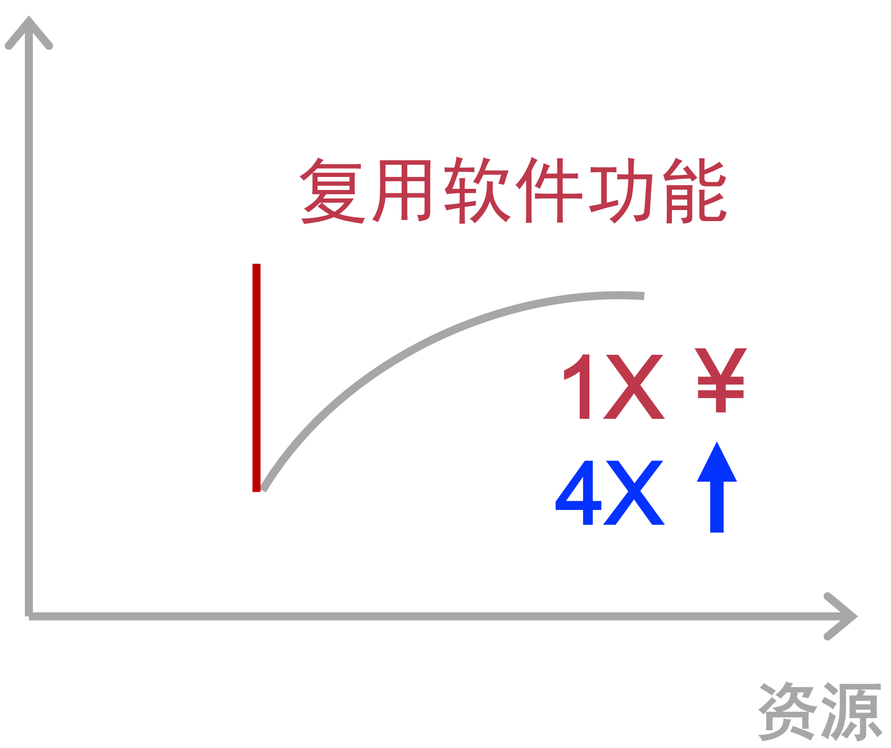
提升性价比：代价不变，性能（功能）提升

- 更多的必要**功能** = 更高的整体**性能**

极致场景：软件功能复用

- （接近）零投入
- 早饭（Breakfast）
  + 午饭（Lunch）
  = 早午饭（Brunch）

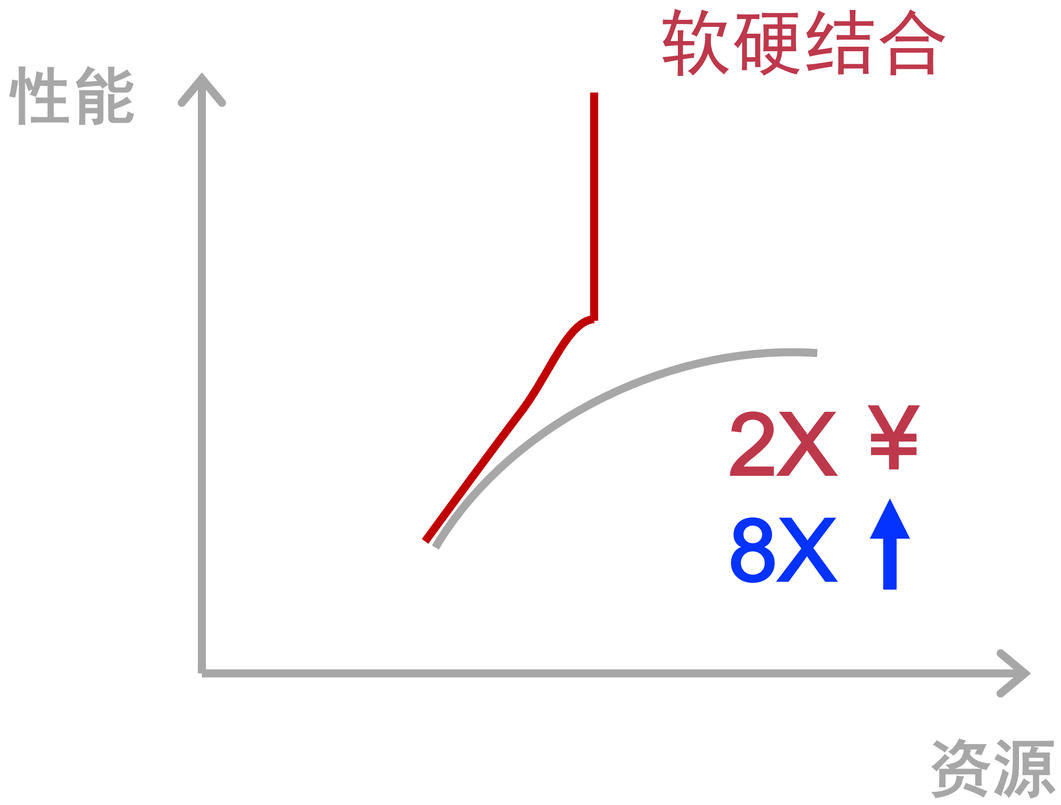




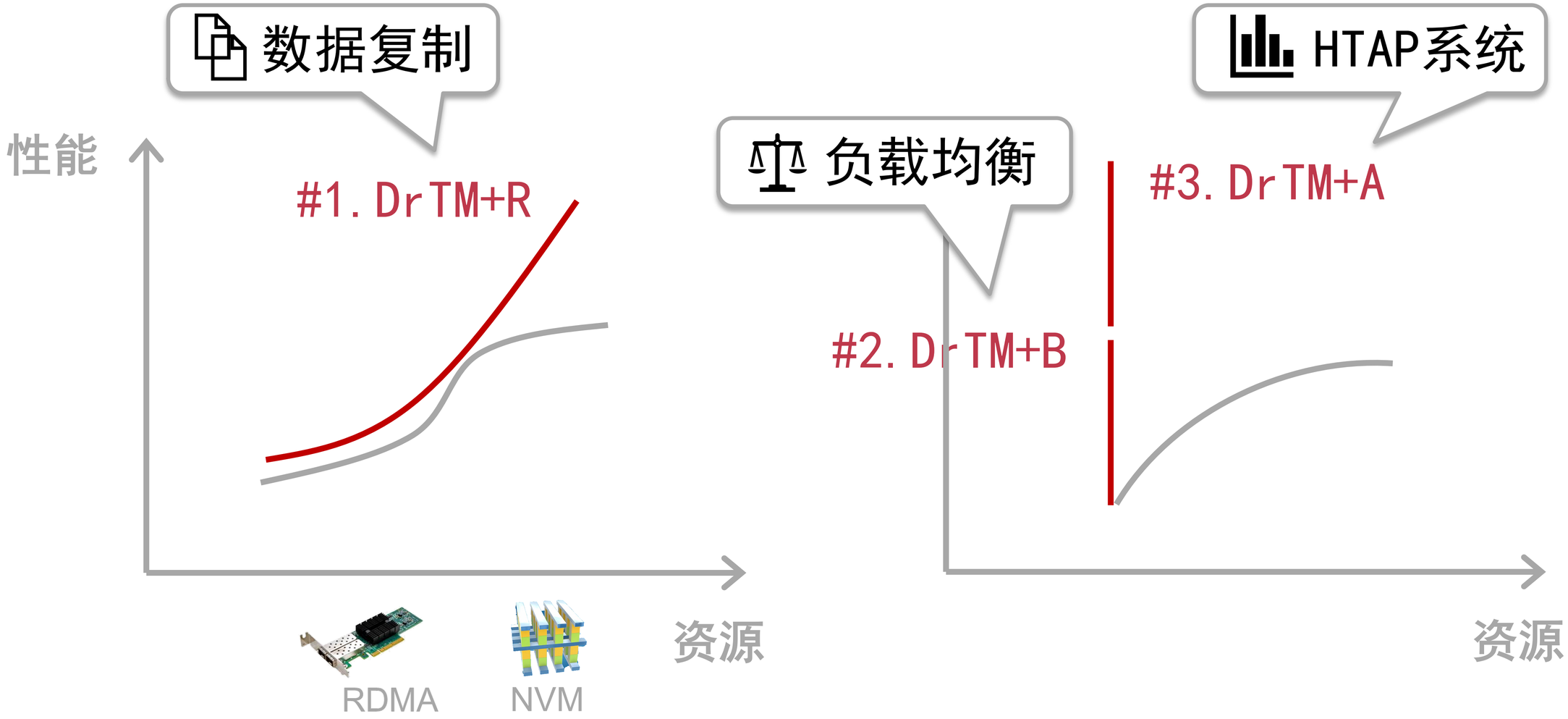
提升系统“性价比”的方法：软硬结合

极致性价比：兑现硬件红利 + 服用软件功能

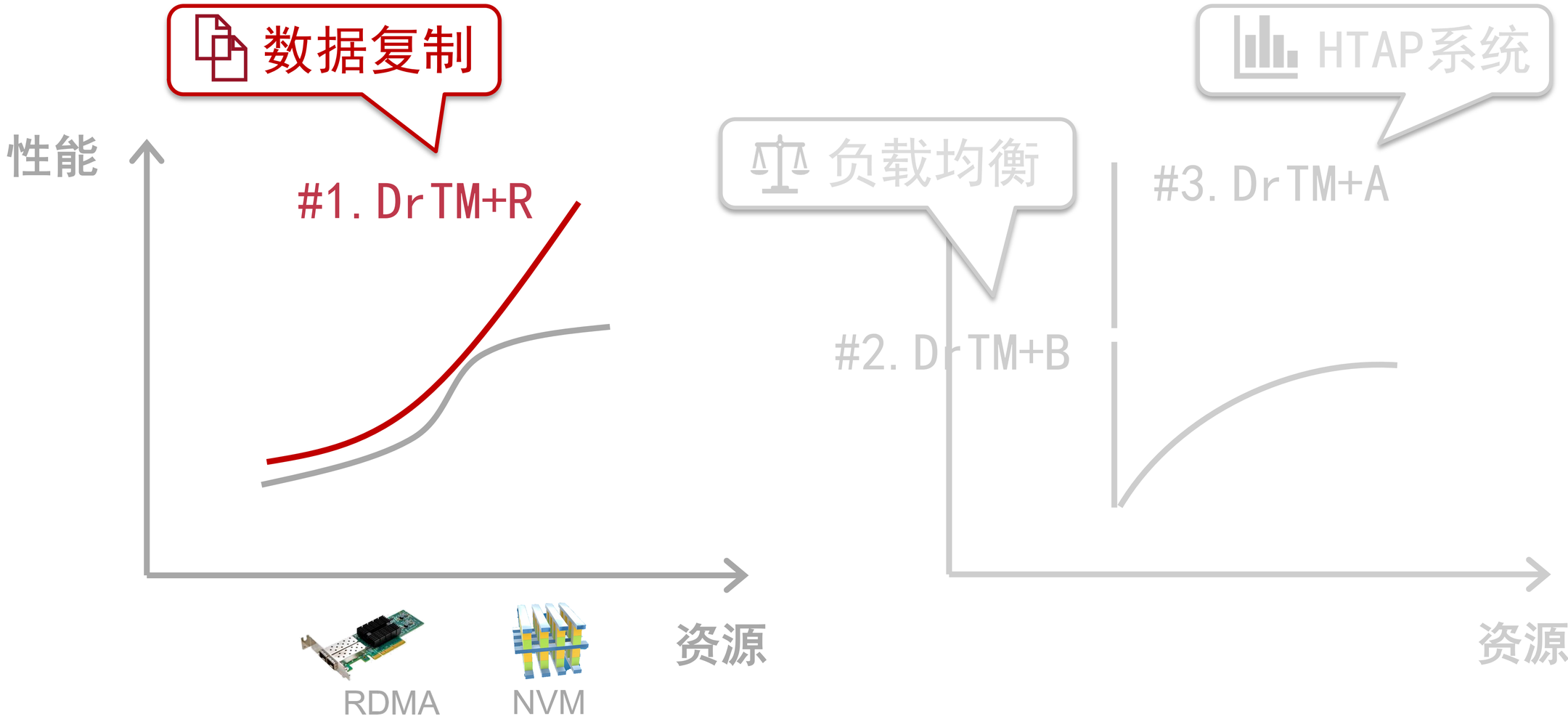
- 两种方法并不冲突，甚至可以正交
- **“一顿丰盛的早午饭”**



我们的尝试：高性价比数据库



DrTM+R：基于新硬件的分布式数据复制



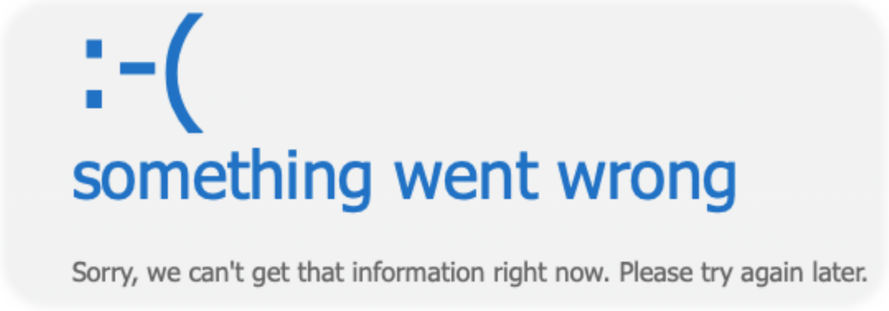
数据高可用与持久化是存储系统的必备需求

某台机器出现宕机是大规模集群的
最常见问题（Common Case）

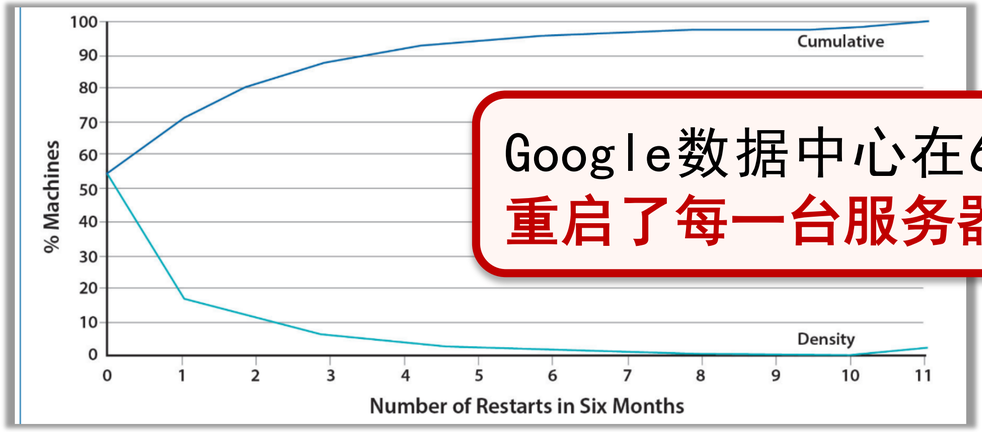
✗ 大规模数据中心每天都有机器下线

宕机意味着数据不可用，甚至丢失

✗ 用户体验极速下降



一个拥有66个节点的邮件服务器集群，
每台只能**连续(不重启)工作不超过12天**

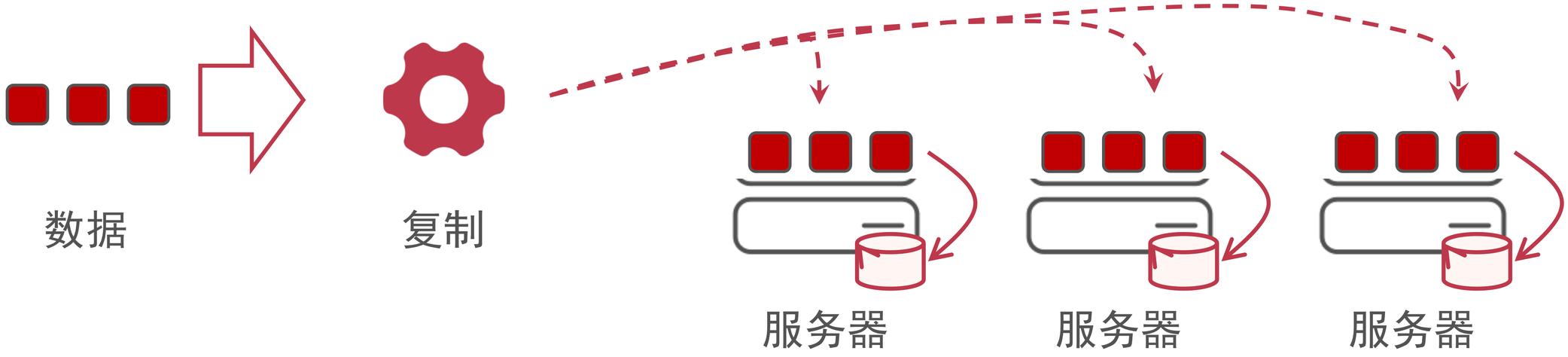


Google数据中心在6个月内
重启了每一台服务器

分布式数据复制（Data Replication）

分布式数据复制：同一份数据在多台机器上存储多个相同副本

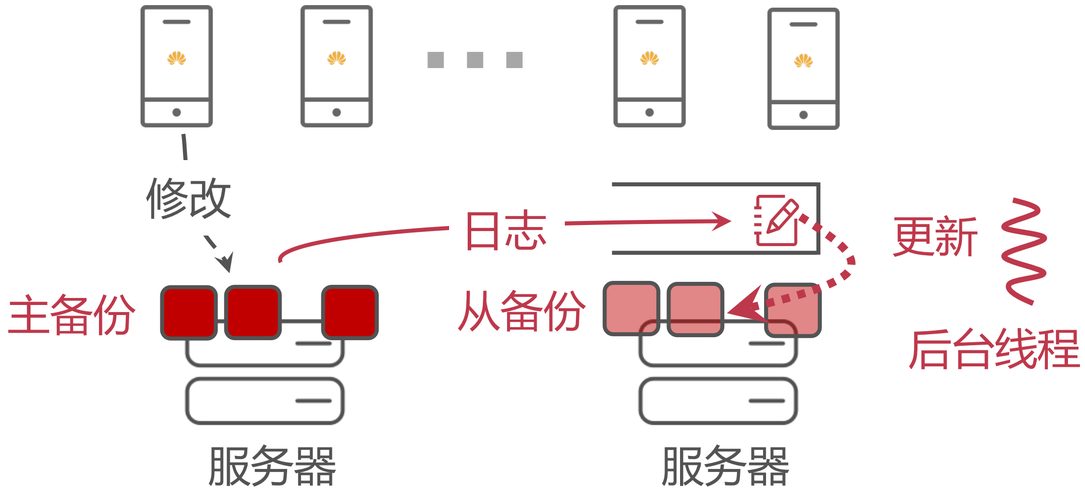
- 高可用：当一台机器宕机时，从另一台机器读取数据
- 持久化：将数据副本存储在持久化介质中（落盘）



基本方法（DrTM）

可用性：基于日志的主从备份

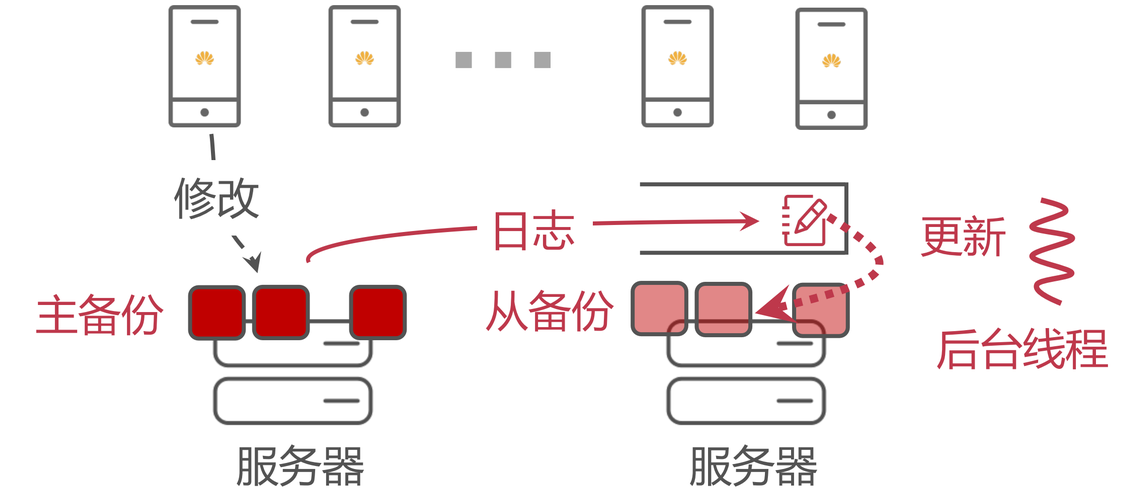
- 同步写入日志（WAL）
- 异步更新副本



基本方法（DrTM）

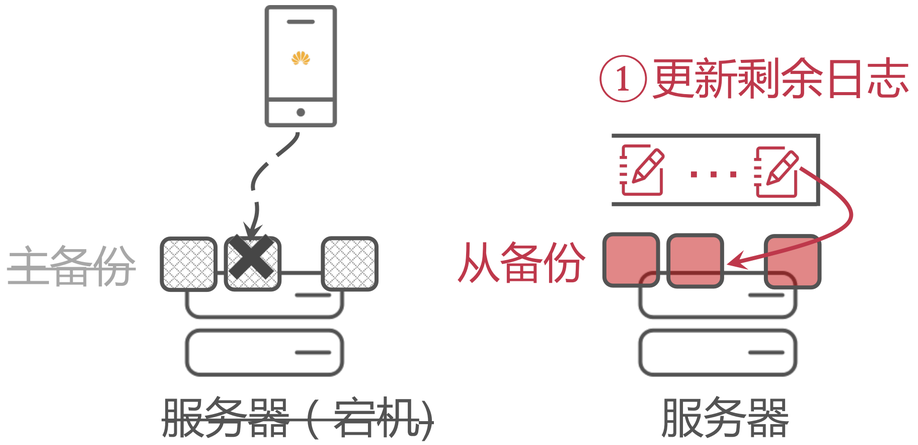
可用性：基于日志的主从备份

- ► 同步写入日志（WAL）
- ► 异步更新副本



宕机系统恢复

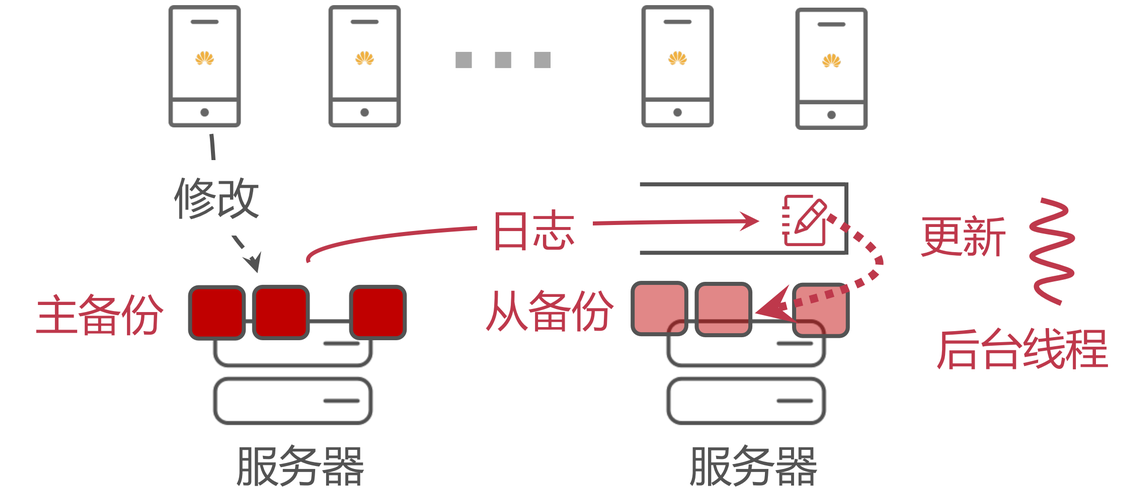
① 清理剩余日志更新副本



基本方法（DrTM）

可用性：基于日志的主从备份

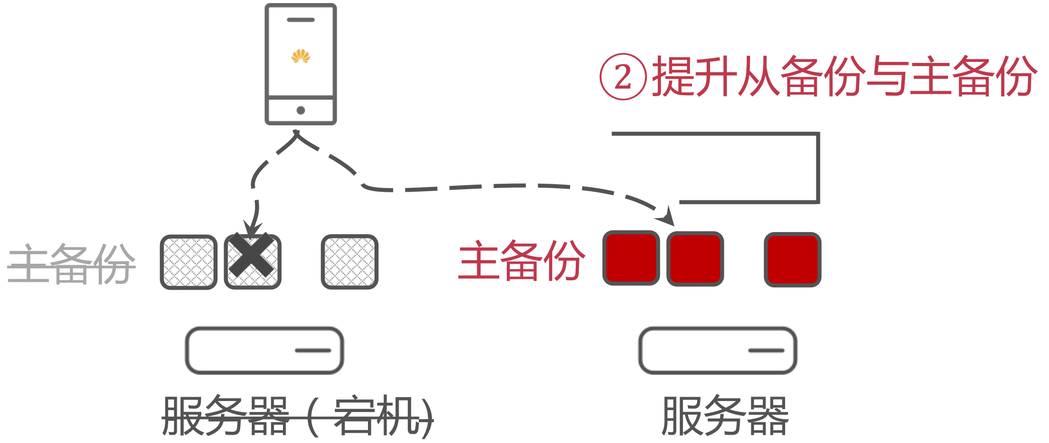
- ► 同步写入日志（WAL）
- ► 异步更新副本



宕机系统恢复

① 清理剩余日志更新副本

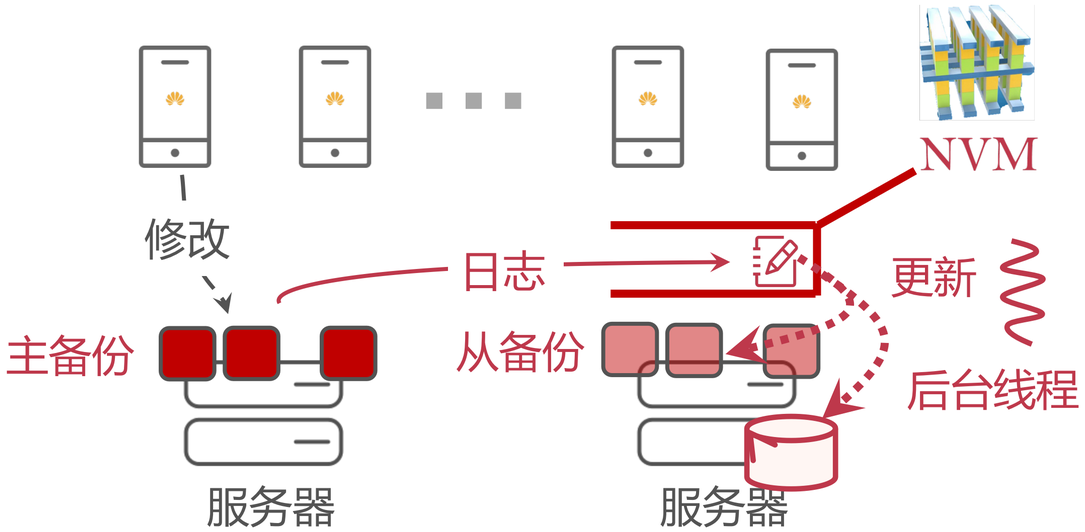
② 切换主从备份

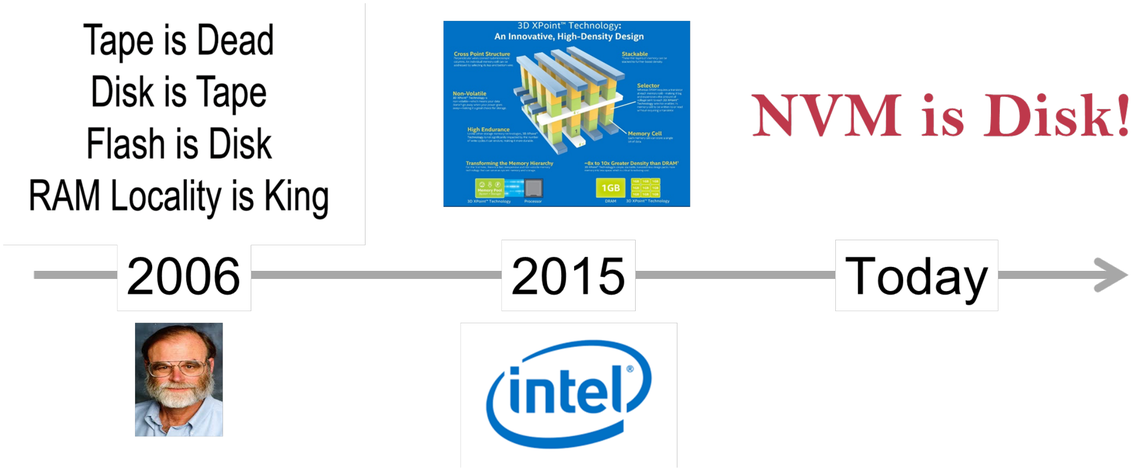


基本方法（DrTM）

持久化：基于持久化内存（NVM）

- 同步存储日志到NVM
- 异步更新副本和落盘





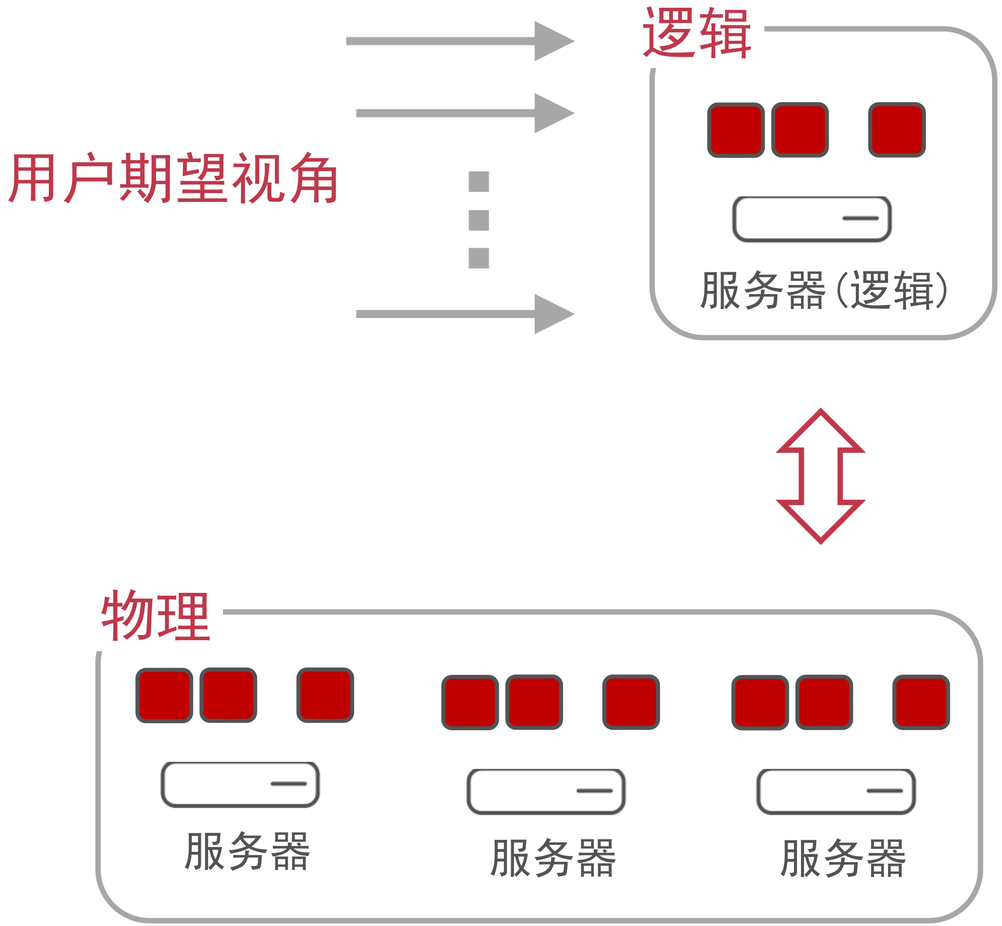
系统开销分析

高可用造成的开销

- 存储开销：多副本
- 时延开销：日志跨节点同步写入

持久化造成的开销

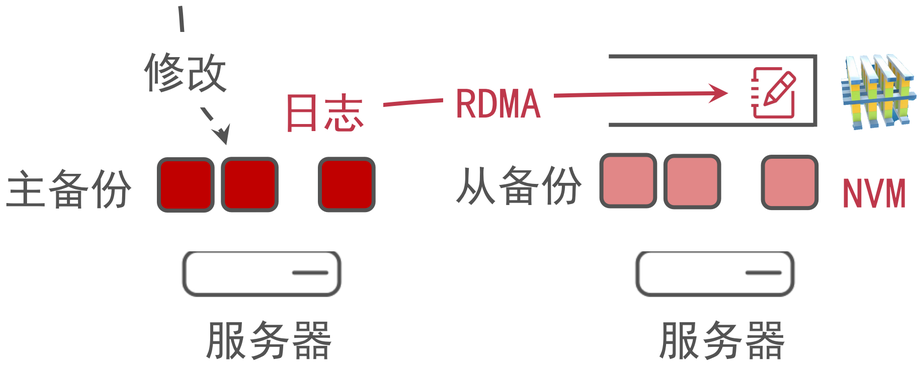
- 时延开销：持久化写入



基于RDMA和NVM的分布式数据复制

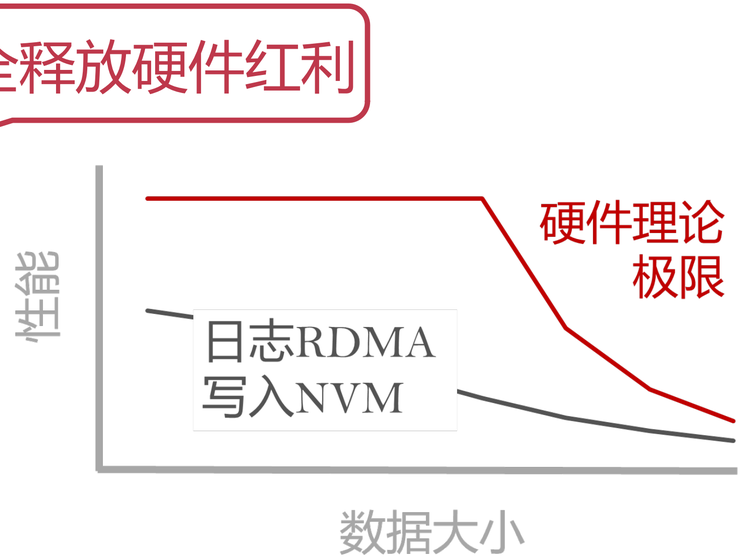
远程直接内存访问（RDMA）

- ✓ 微秒级跨节点访存（单边读写）
- ✓ 节省CPU资源（CPU/Kernel Bypassing）
- ✓ 能够直接写入NVM



直观的设计：使用RDMA WRITE同步写NVM日志

- ► RDMA写日志无法达到硬件(NVM)理想性能
- ► 需要两次RDMA单边操作实现“持久化”

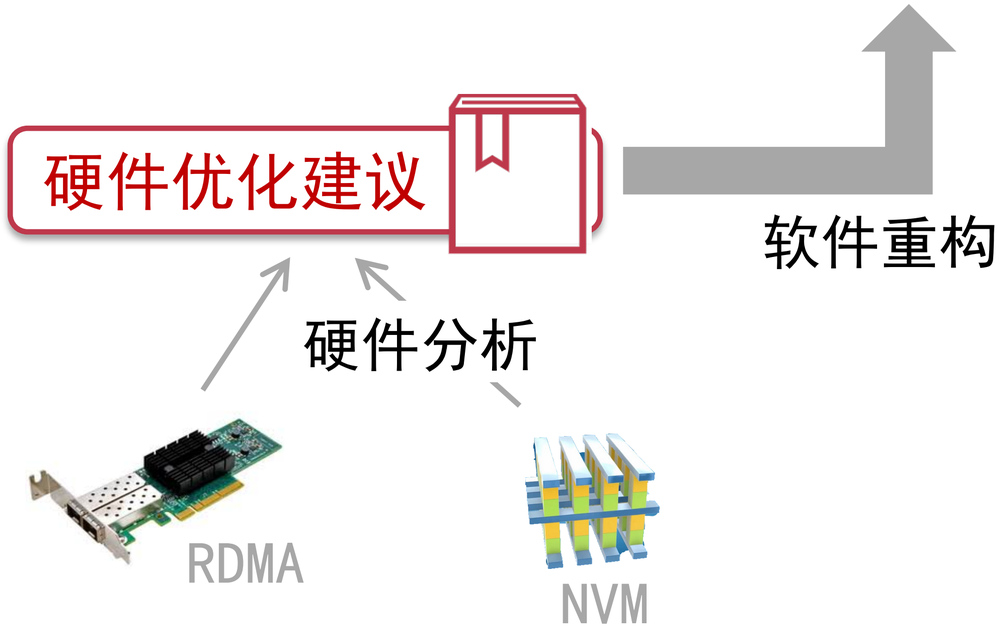


我们的方法：软硬件协同设计

硬件分析：提出硬件优化建议（RDMA+NVM）

软件重构：根据硬件优化建议重构系统

DrTM+R



DrTM+R：高效分布式数据复制

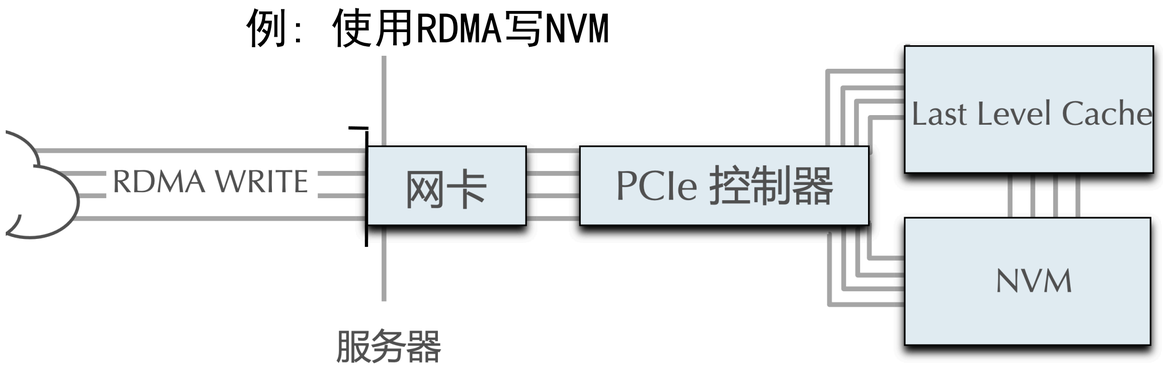
- ✓ 6百万事务每秒（TPC-C）
- ✓ 30μs 持久化数据复制延迟

硬件分析：提出硬件优化提示

问题：无法达到理想性能

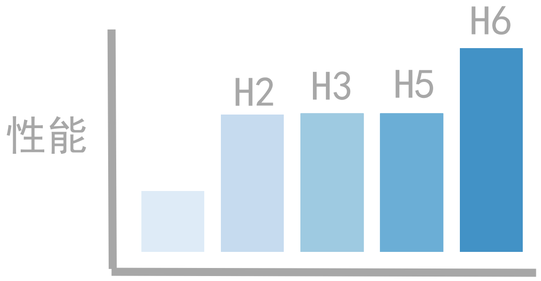
① 独特的硬件特性

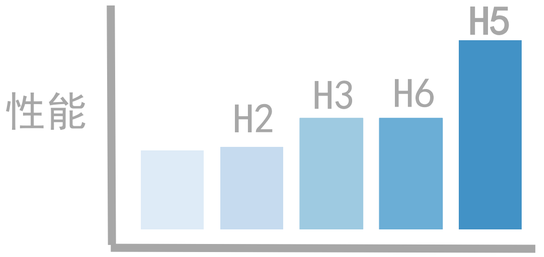
② 硬件之间复杂的交互方式



RDMA-NVM硬件优化提示

RDMA-NVM 优化提示(H1-H9)
H1. 对于大数据传输使用NT-store
H2. 避免使用跨socket访问
H3. 避免以小于xpline粒度的访问
H4. 减少并发CPU NVM访问
…

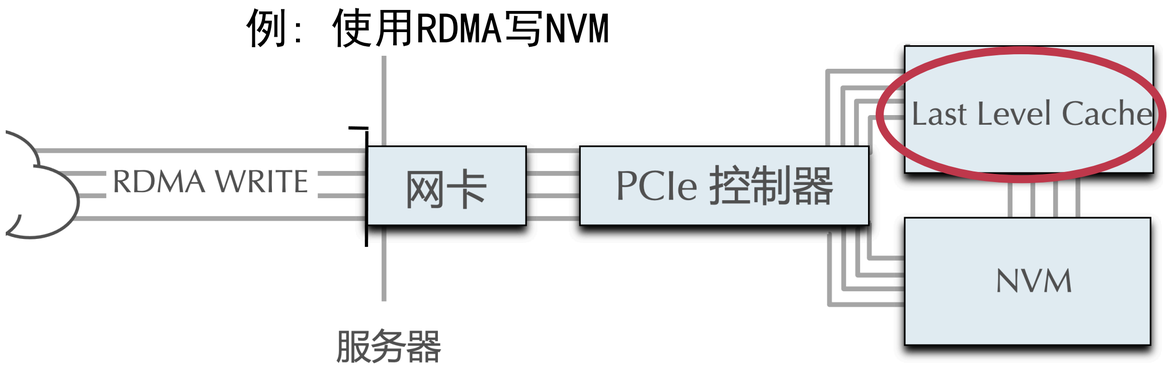




硬件分析：提出硬件优化提示

RDMA-NVM硬件优化提示

H5 关闭DDIO[1]以绕过处理器缓存

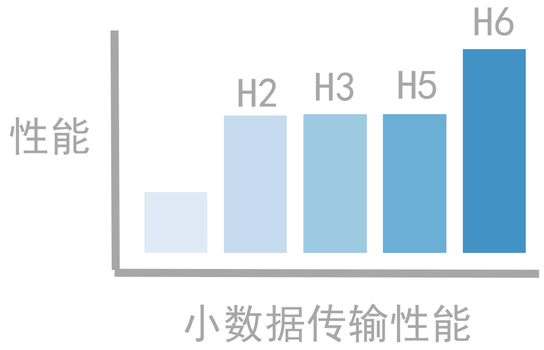


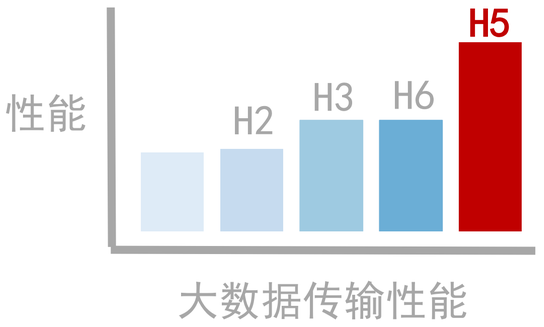
NVM硬件特性

- 性能：顺序写 >> 随机写

默认RDMA-NVM行为

- DDIO会优先将RDMA写入缓存
- 缓存以随机方式写回NVM



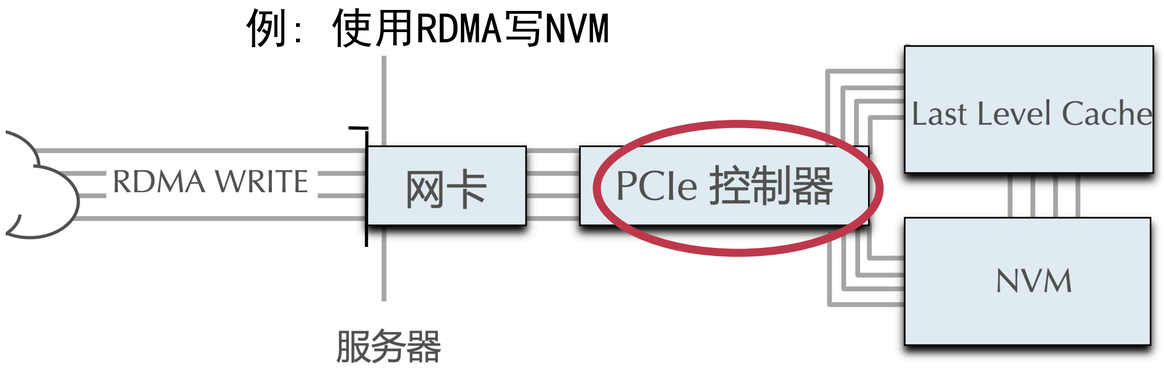


[1] DDIO: Intel data direct IO

硬件分析：提出硬件优化提示

RDMA-NVM硬件优化提示

H6 按PCIe最小传输粒度访问NVM

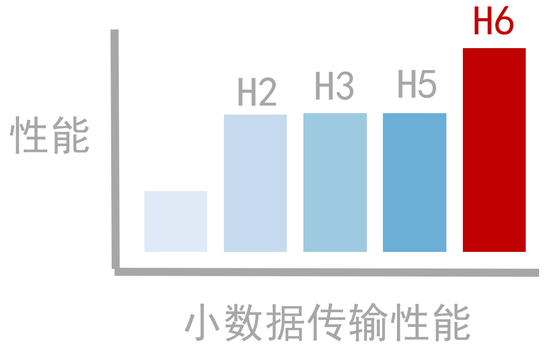


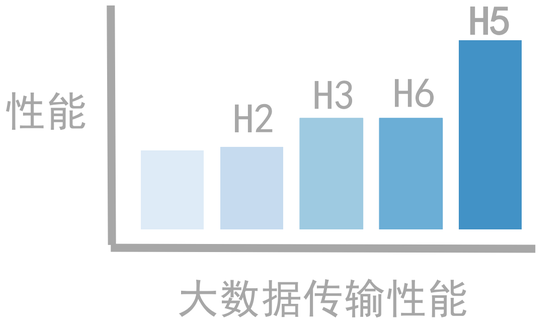
NVM硬件特性

- NVM读请求严重影响写请求

默认RDMA-NVM行为

- RDMA请求不满足PCIe最小粒度
- PCIe将写变成先读再写





软件重构：基于建议重构系统

利用RDMA-NVM硬件优化建议指导软件重构：1.45X性能提升（相同硬件资源）

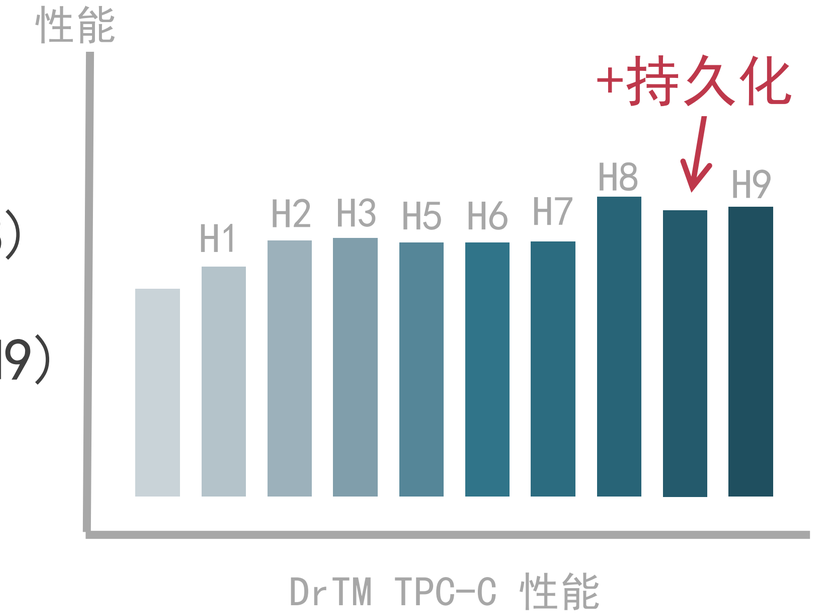
RDMA-NVM 优化提示(H1-H9)
H1. 对于大数据传输使用NT-store
H2. 避免使用跨socket访问
H3. 避免以小于xpline粒度的访问
H4. 减少并发CPU NVM访问
…
…
…
…

DrTM+R系统化设计

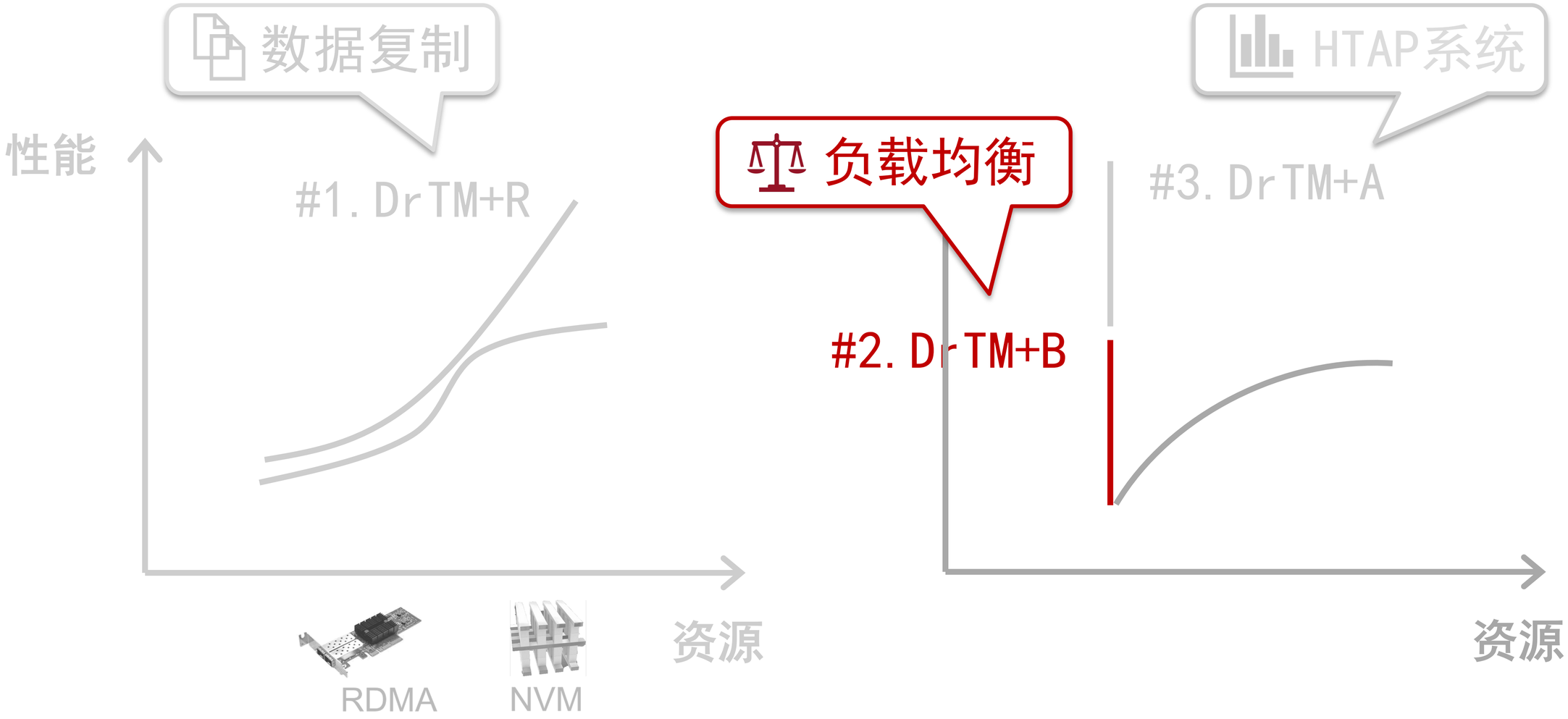
① 优化系统配置(H2, H5)

② 优化存储&日志结构(H3-H8)

③ 优化日志持久化写入(H1, H9)

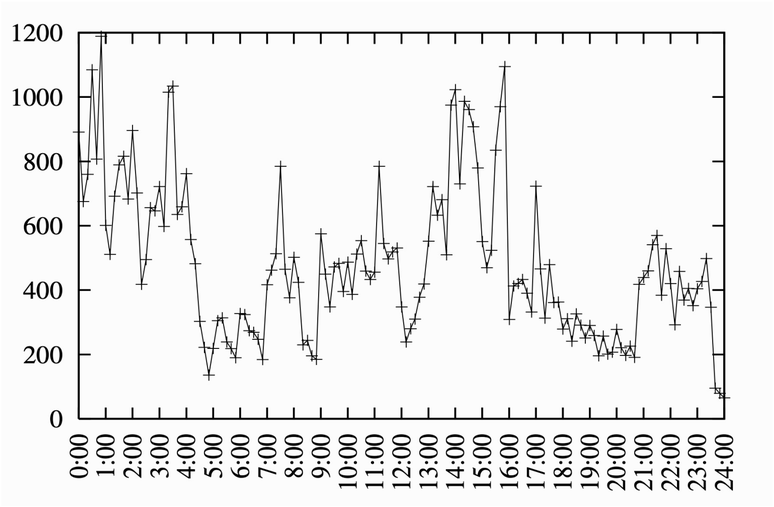


DrTM+B：基于数据复制的动态负载均衡支持



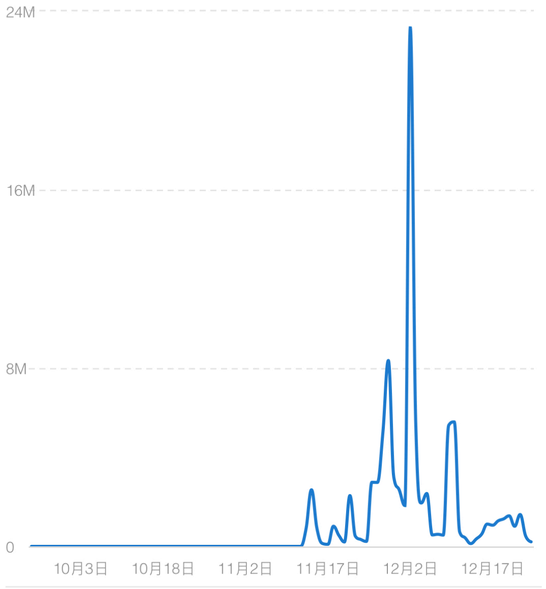
数据访问模式不断快速变化

随时间变化



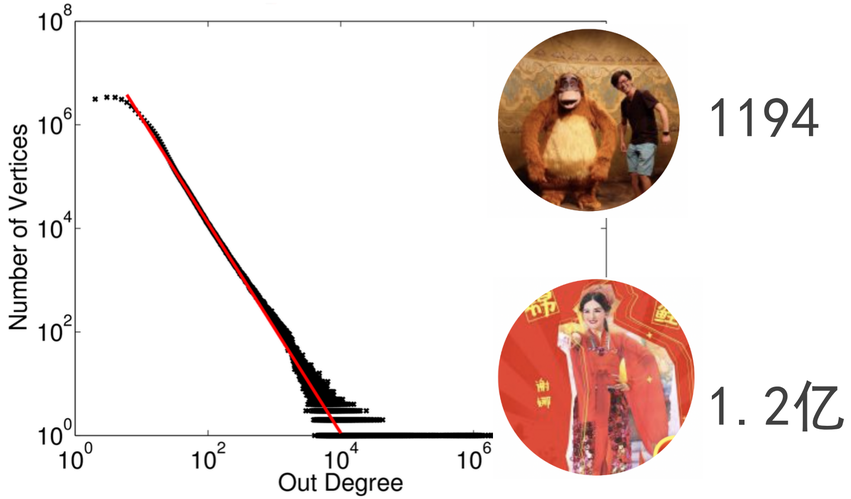
一天内淘宝请求量变化

随热度变化



微博“丁真”搜索变化

不均匀的访问



微博用户的关注度

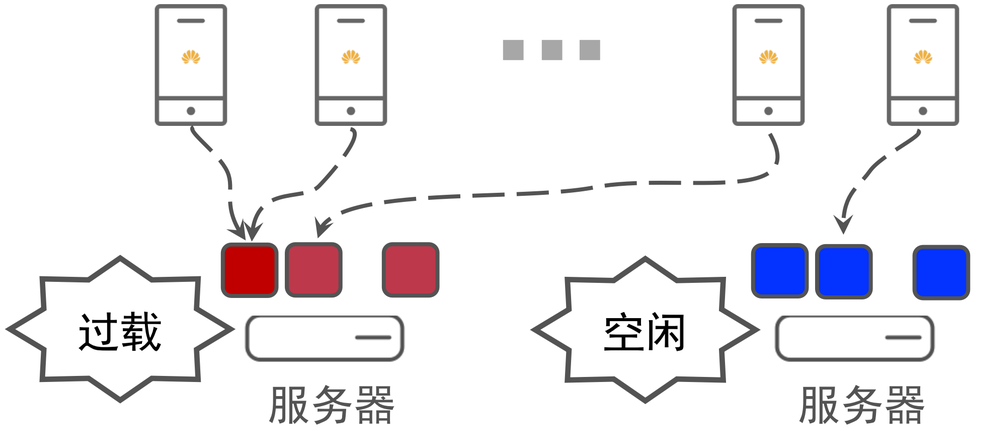
负载不平衡降低系统性价比

系统性能下降 & 资源利用率下降

✗ 空闲的机器 =〉 资源浪费

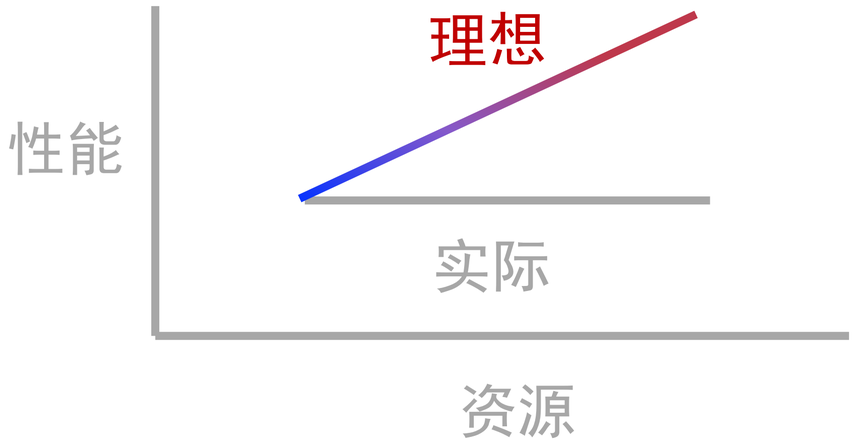
✗ 过载的机器 =〉 性能下降

✗ 性价比：1 / X



需要动态响应数据访问模式

✓ 性价比：1 / 1

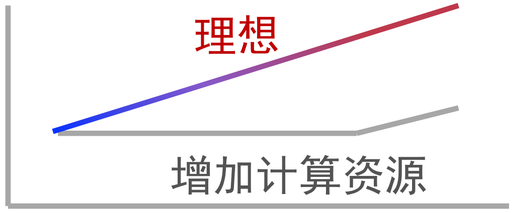


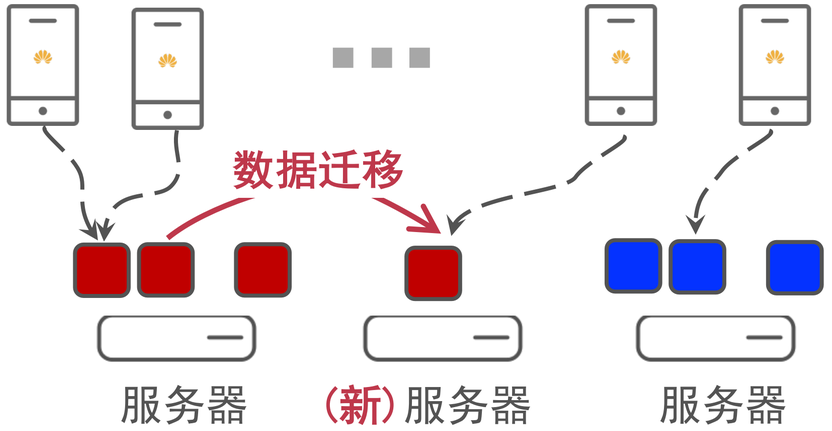
负载不平衡降低系统性价比

传统方法：增加资源 & 负载平衡

① 增加资源

✗ 性价比提升有限





② 负载平衡：现有方法

✗ 基于post-copy的数据迁移

✗ 性能损失大：保证迁移过程数据一致性

✗ 迁移时间长：数秒内无法响应用户请求

✗ 资源浪费多：需要有空闲机器（standby）

我们的方法：复用数据备份实现数据迁移

分布式数据复制提供了**多个数据备份**

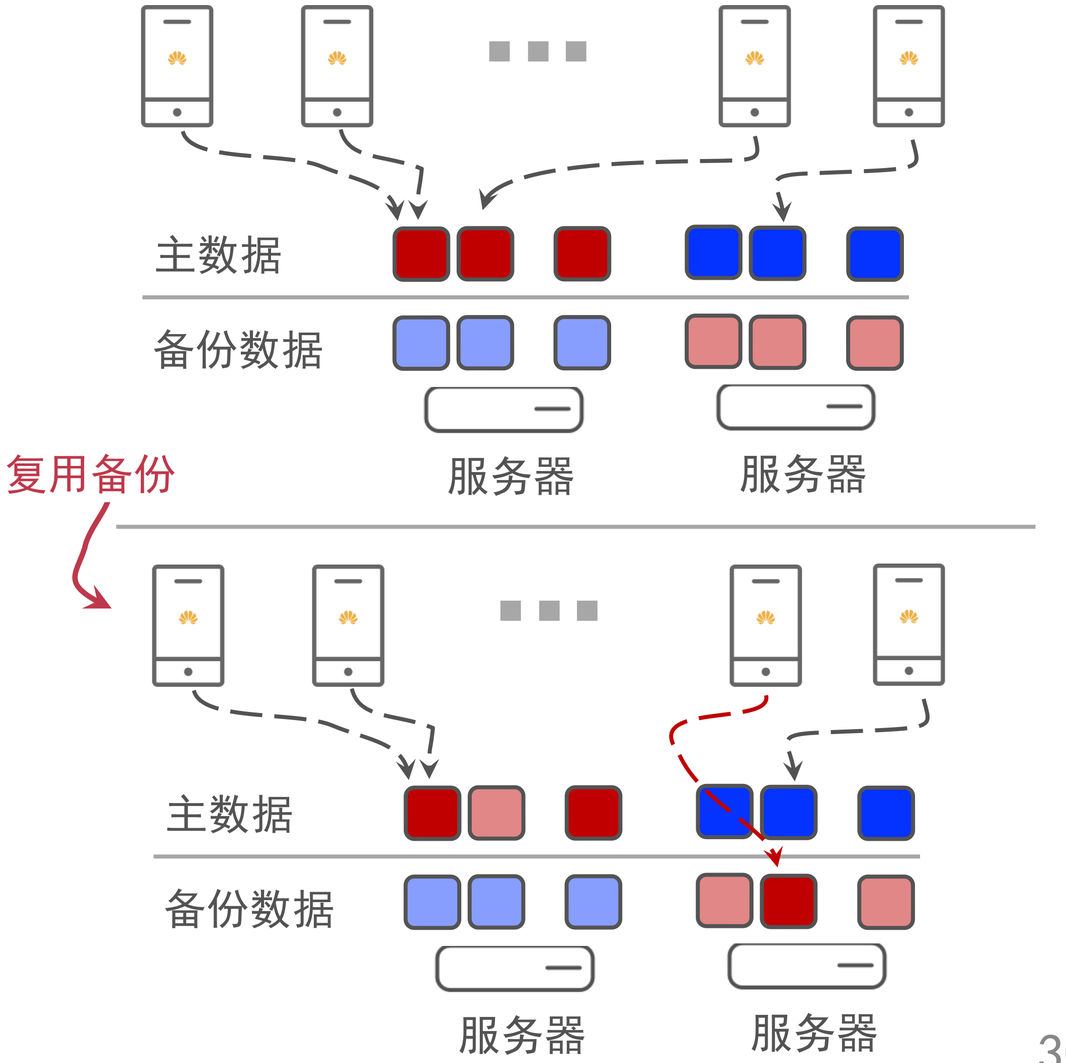
- 思路：利用主从备份切换实现数据迁移
- 优点：计算复用（数据同步）
  存储复用（兼容高可用）

挑战

① 如何确保迁移过程中的**数据一致性**？

② 如何避免由于数据迁移出现**二次过载**？

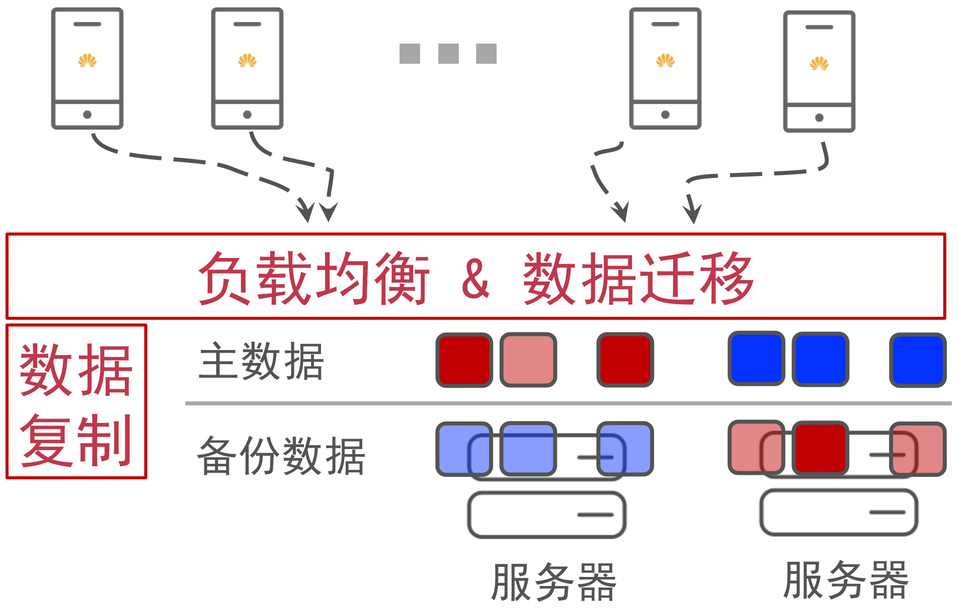
③ 如何实现在多台机器**均分热点负载**？



DrTM+B：基于数据复制的负载均衡技术

采用Pre-copy策略的动态数据迁移技术

- 性能损失低：原子主从副本切换协议
- 迁移速度快：复用备份避免多轮copy
- 节省资源：备份感知的迁移规划



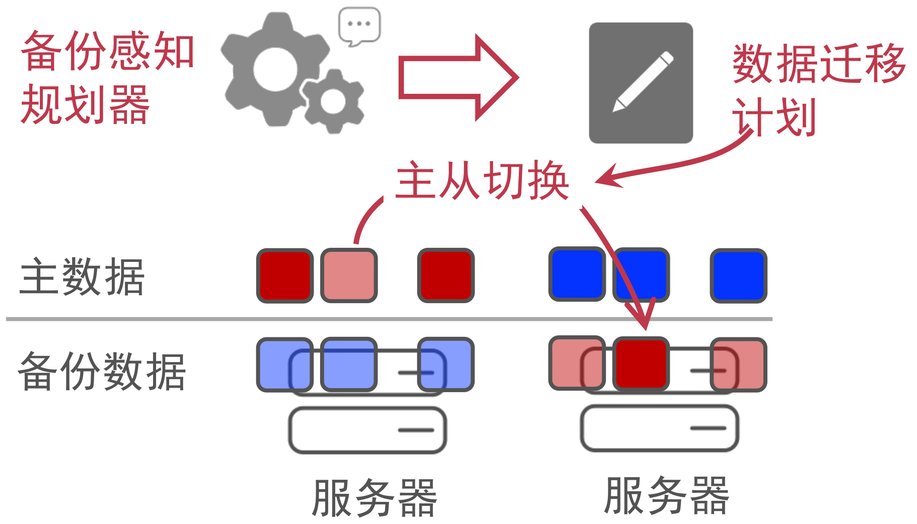
优化

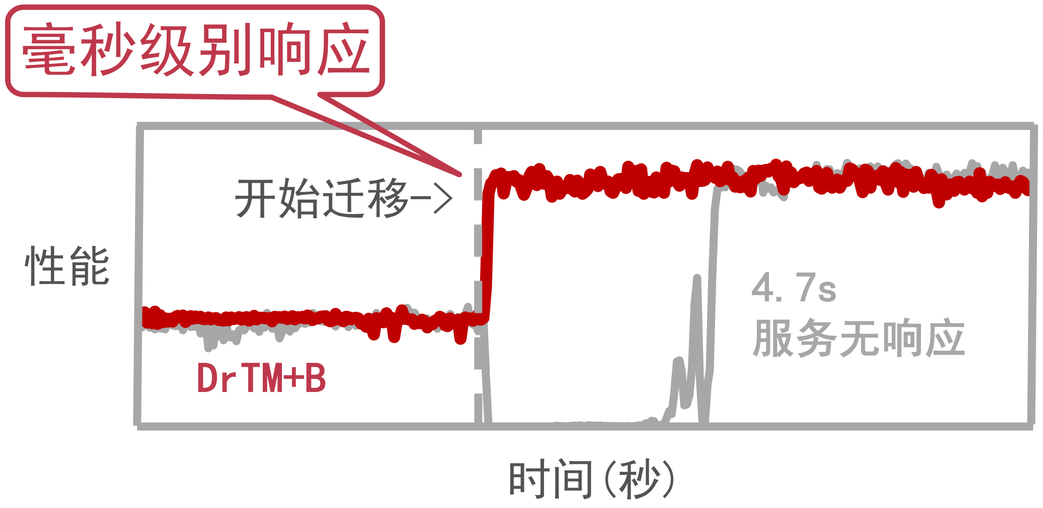
- 基于RDMA的主动迁移避免二次过载
- 采用数据分块支持细粒度局部迁移

DrTM+B：基于数据复制的负载均衡技术

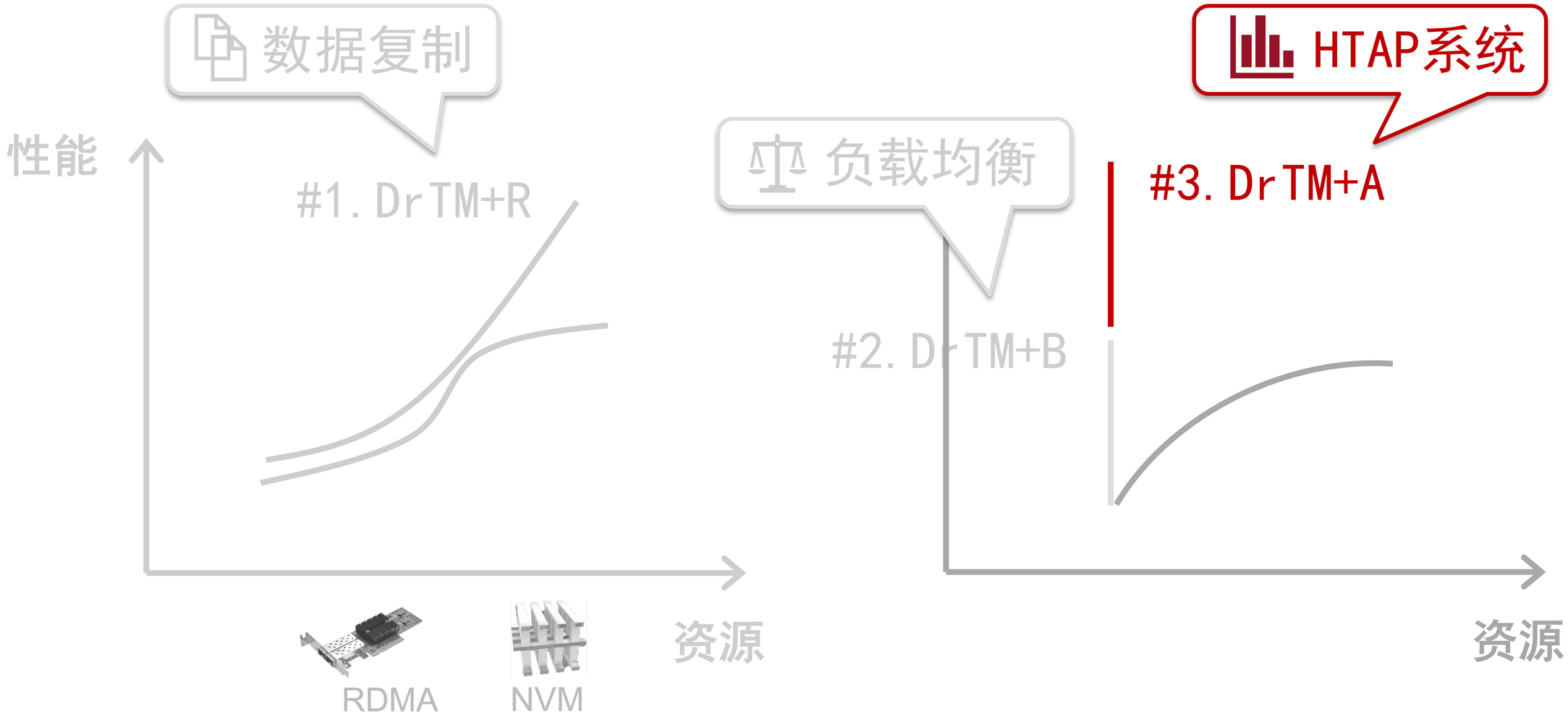
在DrTM+R上实现动态负载均衡

- **无需增加硬件资源**支持动态数据迁移
- 迁移过程中性能损失**小于10%**
- 迁移时间缩短至**40毫秒以内**





DrTM+A：基于数据复制的混合事务分析支持



混合事务分析处理（HTAP:OLTP+OLAP）

OLTP（在线事务处理）

► 写入数据、行优先存储

OLAP（在线分析处理）

► 读取数据、列优先存储

HTAP：混合事务分析处理

① 性能影响小（OLTP/OLAP）

② 低延迟（几十毫秒级）

2018年双十一峰值吞吐六百万事务每秒

- ✓ 检查交易欺诈
- ✓ 用户商品查询
- ✓ 供应商追踪销量
- ✓ . . .

场景	延迟
欺诈检测	20ms
系统检测	20ms
广告推荐	100ms
股票交易	200ms

HTAP目标：<10%性能影响，10毫秒数据延迟

现有HTAP解决方案

方案1：双系统（连接现有系统）

► 性能影响小、数据延迟高

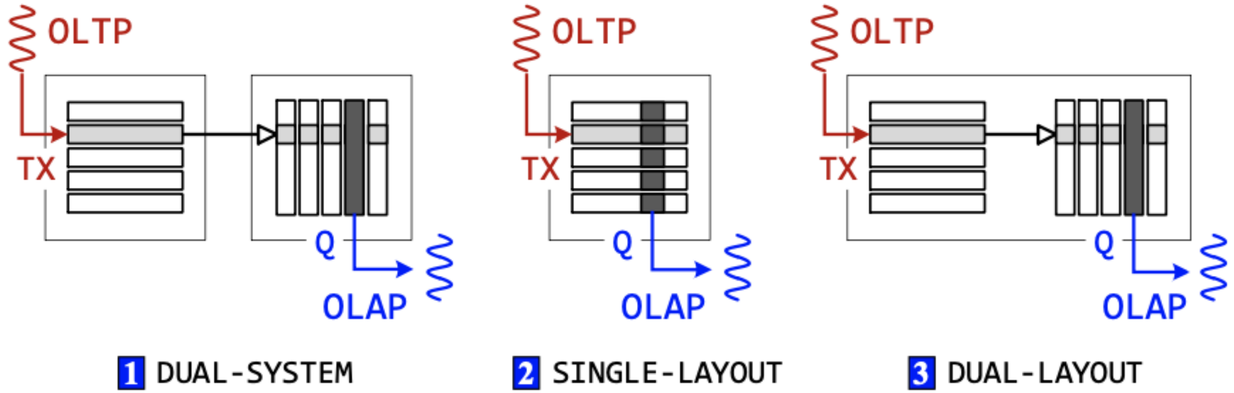
方案2：单系统单数据格式

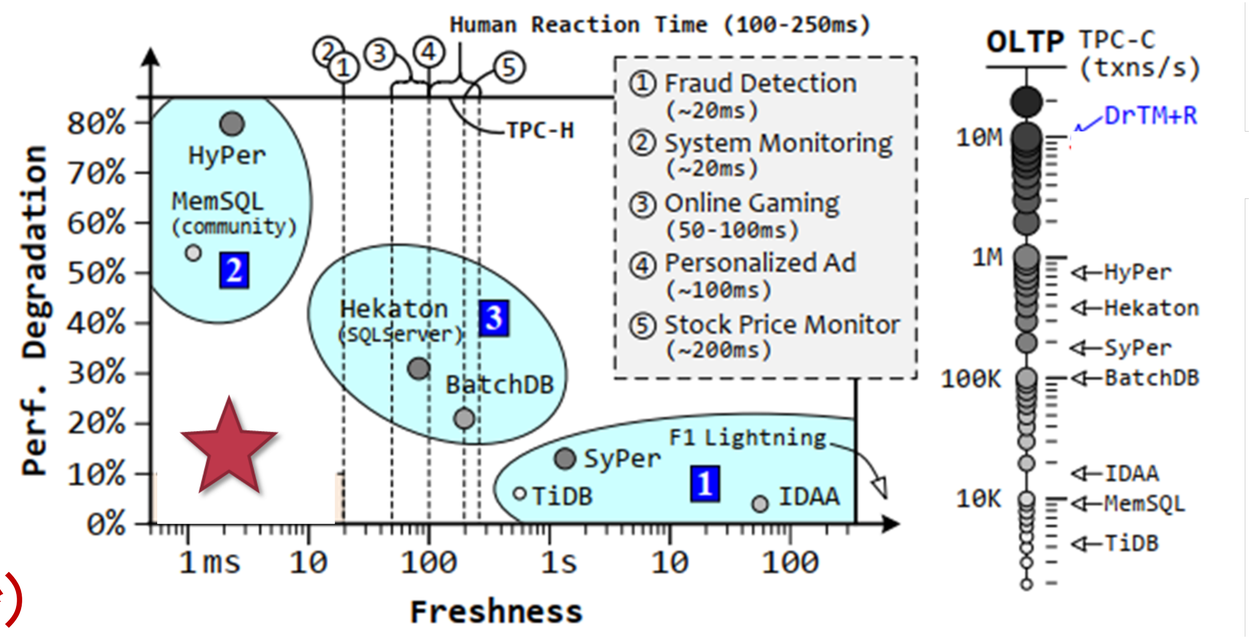
► 数据延迟低、性能影响大

选择3：单系统双数据格式

► 性能影响和数据延迟间权衡

新挑战：超高OLTP性能(百万事务每秒)





DrTM+A：基于数据复用的HTAP系统

数据复制能够提供HTAP系统关键功能

- ✓ 复用“从备份”执行分析负载 → **性能影响小**(性能隔离)
- ✓ 复用“同步日志”保证数据一致 → **数据延迟低**

} **高性价比**

难点：超高吞吐下实现新鲜、列式、容错的数据复制（AP备份）

- ✓ 列式的：OLAP性能高
- ✓ 新鲜的：数据延迟低
- ✓ 容错的：系统开销小

技术挑战

根本矛盾：读（分析）写（事务）冲突

挑战1：并发日志清理

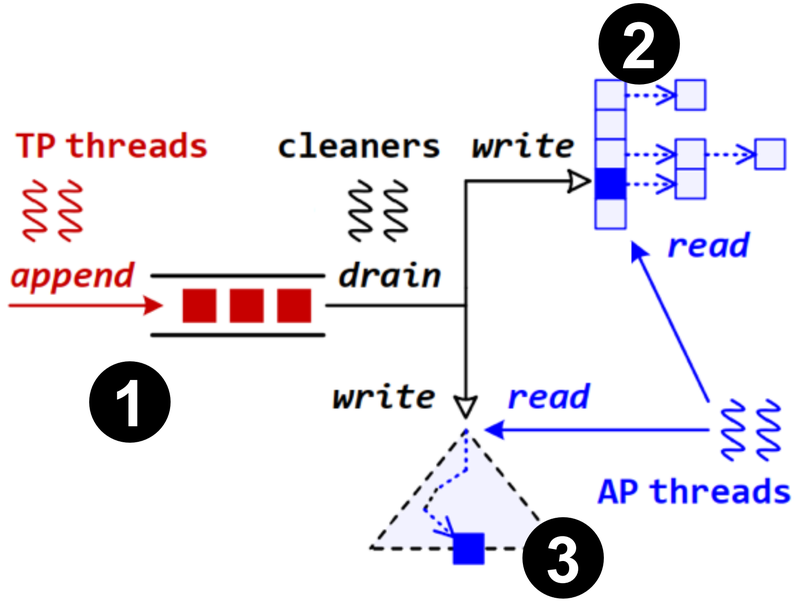
► 如何保证并发清理数据的一致性

挑战2：列式多版本存储

► 如何抱持读写操作的局部性

挑战3：树结构查询索引

► 如何实现读写操作的并发执行



解决方案

新瓶装旧酒：Epoch思想应用在HTAP场景

技术点1：按需同步的并发日志清理

► “写”者负责节点内一致、“读”者负责跨节点一致

技术点2：块结构的列式多版本存储

► 块结构提供“读”局部性、行分割+列合并挖掘“写”局部性

技术点3：两阶段并发插入协议

► 延迟插入避免“读-写”竞争、批量插入避免“写-写”竞争

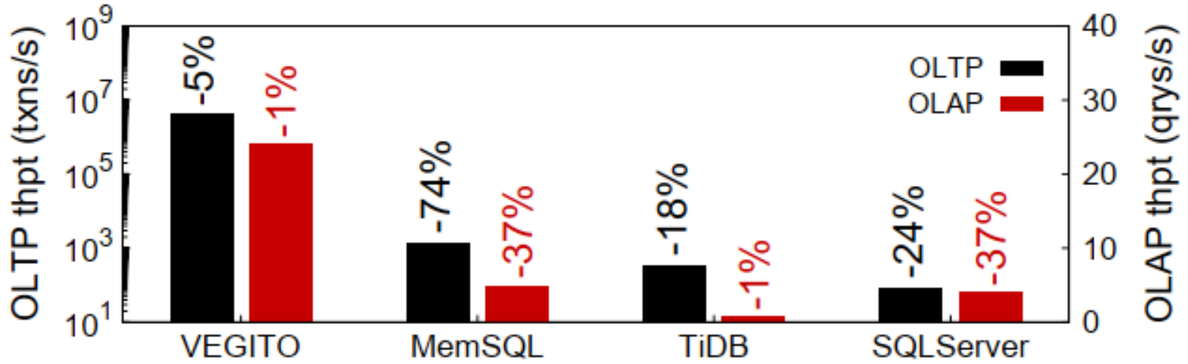
评测

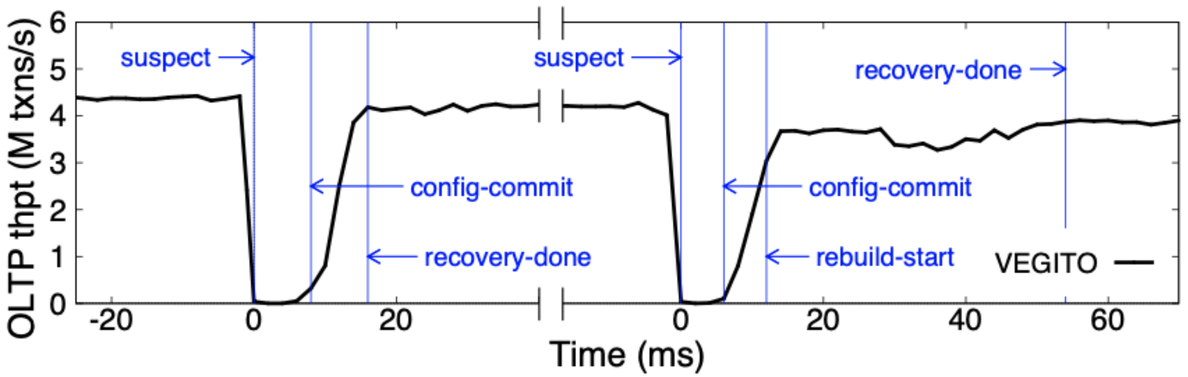
绝对性能相比传统系统有**数量级提升**

性能影响：混合负载下 **<5%**

数据延迟：平均 **<10ms**

容错机制和性能几乎**无影响**





小结

提升系统“性价比”的两个可能方向

① 兑现硬件红利

② 复用软件功能

“软硬结合”尝试：基于功能复用构建高性价比分布式系统

- DrTM+R: 基于RDMA和NVM的分布式数据复制
- DrTM+B: 基于数据复制的动态负载均衡支持
- DrTM+A: 基于数据复制的混合事务分析支持

谢谢