Efficient Memory Disaggregation with INFINISWAP
NSDI'17
University of Michigan

【背景】
问题背景 1.内存密集型应用逐渐流行,因为服务请求只需要从内存中读取，磁盘访问被最小化，延迟降低并且吞吐量增加
问题 1.1 当工作集不能完全容纳进物理内存时，性能会遭受巨大的损失
问题详述 1.1.1 当应用寻址一个虚拟地址，它对应的物理地址指向的页不在内存中时会发生缺页错误
接下来，虚拟内存管理器(VMM)会查询页表然后从磁盘中将页读入内存中
问题详述 1.1.1 页面交换有显著的非线性的性能损失[已证明]
问题详述 1.1.2 页面交换的影响在尾延迟上非常显著[已证明]
@洞察点 1：巨大的性能差异表明理论上 100%的高效内存解体方案可以带来巨大的收益
(前提是这个方案要求的一切条件都得到满足)

基本解法 1.2.1 适当调整内存分配
难点: 应用经常过高估计它们的内存使用量，或者试图分配峰值使用时的内存大小
基本解法 1.2.2 增加每台机器的有效内存容量

问题背景 2.集群范围内结点的内存利用率非常不平衡，这些内存密集型应用在将内存页换出到磁盘上时也不能利用集群范围内的空闲内存
问题 2.1 分析 google 和 facebook 两大集群的 trace,内存利用率确实不均衡[已证明]
@洞察点 2：充分利用集群范围内的内存
@洞察点 3：短时间间隔内，内存利用率保持稳定

环境背景 3.现代 RDMA 网络可以满足对于巨大内存负载的内存解体架构对延迟的要求
@洞察点 4：网络速度比磁盘读写速度高出一个数量级

【已有工作】
出发点：在磁盘-网络时延缺口条件下，探索远程内存交换做法 since from 1990

工作方向：在集群范围内对分散的内存进行利用

已有工作 1：资源分解
工作：为了解耦资源规模和提高数据中心效率，资源分解和机架规模计算近年来收到重视，其中以内存分解为主

已有工作 2：远程内存交换
a.
缺点：网络慢，cpu 负载高，导致性能差,依赖集中控制来做远程放置选择，踢出选择和负载均衡，导致可扩展性差
b.HPBD,Mellanox nbdX [Network-attached-storage]
做法：服务端使用 RAMdisk，RDMA 网络
缺点：数据拷贝到 RAMdisk 造成 CPU 瓶颈,不支持动态内存管理，忽略内存踢出的可能和相关问题，没有考虑容错以及出错时的性能损失

已有工作 3：DSM [ATC'15 Latency-Tolerant Softeware Distributed Shared Memory]
做法：
1.传统上，DSM 系统会为了维持一致性而承受严重的通信开销
2.为了避免开销，HPC 社区采用了 PGAS(划分全局地址空间)模型，但是需要重写用户应用程序并且必须感知远程数据访问
3.有了 RDMA 技术以后，又重新开始了 DSM 的研究，尤其是通过键值存储接口来做研究
缺点：限制于接口，需要用户应用程序的重写

【作者工作】
出发点：#想干什么
在不修改应用或者操作系统的条件下
1.增加整个集群的内存利用率
2.优化内存密集型应用页面交换性能
3.可扩展 *
4.可容错 *
5.对应用透明，远端机器性能不受影响 *

适用范围：RDMA 网络环境#ref: a remote memory paging system designed specifically for an RDMA network

解决方法：使用 RDMA 网络做[远程内存页面交换]

具体做法：
1.INFINISWAP 块设备
向 VMM 提供一个传统的块设备 I/O 接口，当做一个固定大小的交换空间，执行 I/O 请求
2.INFINISWAP daemon
管理远程内存

解 决 的旧问题：#怎么干的
旧问题 1：扩展性
解法：
利用 power of choices 技术来决定 slab 放置和踢出，消除集中式协调需求
slab 管理：
使用指数加权移动平均值在每一秒钟检测内存页活动率[可以反应历史值]
公式：Ac(s) = α * Am(s) + (1 - α) * Ao(s)
A(s)指对于 slab s 来说，page in 和 page out 活动的总次数
当 A(s) > HotSlab (op/s)，将 Slab 映射到远程内存上

评价：
没有给出 Slab 的选择
远程 slab 放置：
目标：将同一个块设备的 slabs 分散到尽可能多的远端机器上去，以尽可能减少被挤出的代价
平衡集群范围内结点的内存利用率，以最小化将来被挤出的可能
必须去中心化以无需集中协调来提供低延迟的映射
做法：将机器结点分为两个集合 new 和 old，先从 new 里面随机抽取两个机器，选择内存利用率较低的那个，如果有必要，再去 old 里面选择
评价：非最优解，考虑并不全面，难以适应复杂情况
1.默认假定 new 机器内存利用率比 old 机器内存利用率低,并不合理
2.仍有优化空间
3.解法粗暴
处理 slab 挤出：
挤出指标：
使用指数加权移动平均值在每一秒钟检测内存利用率
公式：Uc = β * Um + (1 - β) * Uo
U 指整个内存使用情况
daemon 维持在这个机器上有 HeadRoom 大小的自由内存，最优值应该是根据内存总量和应用动态决定的
挤出条件：当自由内存低于 HeadRoom 时
做法：
阶段 1：释放尚未映射的 slabs
阶段 2：如有必要，踢出 E 个已经映射的 slabs
踢出算法:
目标：最小化远端机器的性能损失
难点：
1.由于 one-side，无法预知每个块的使用情况
2.集中式搜集信息将会要求频繁的报告 slab 活动情况
3.随机挤出 slab 会有很大概率挤出 busy 状态的 slab
做法：如果要踢出 E 个 slab,则选择 E+E'个 slab 与对应的远端进行通信，然后踢出 E 个 slab
概率 = Sum(E,E+E'){(1-p)^i * p^(E+E'-i) * C(i,E+E')}
评价:
1.没有方法确定 HeadRoom 的值
2.并非全局最优，没有性能保证，仍有优化空间
3.E'值没有全局方法确定，只能靠经验和测试
4.做法不纯粹

旧问题 2：容错性
解法：
将块设备的地址空间划分成多个 slab，分散到多台机器上去
失效时依赖多个远程内存和本地磁盘进行恢复

处理远程失效：
检测到远端不可达时，找出映射到该远端的 slab 然后置为未映射，所有后续的请求定向到磁盘 I/O 队列中
处理远程挤出：
1.收到 EVICT 消息后，将 slab 标记为未映射，后续对应的请求定向到磁盘 I/O 队列中
2.等待当前 RDMA I/O 队列中的请求都完成以后给对方回应一个 DONE 消息
3.如果被 EVICT 的 slab 的 A(s)仍然高于 HotSlab 阈值，就立刻进行重新映射

I/O 流水线：
1.每个 cpu 核配置一个块请求软件队列
2.Router 负责处理所有软件队列的请求
3.所有的请求会流向磁盘 I/O 队列和 RDMA I/O 队列
4.映射方式使用 hash 映射，想做到负载均衡
内存页写：
已映射：
复制页到 RDMA buffer 中一份，在磁盘 I/O 队列和 RDMA I/O 队列各放一个请求，共享 RDMA buffer 内容
一旦 RDMA write 完成，回收原来的内存，不用等到磁盘写入完成，但 RDMA buffer 必须等到磁盘写入完成后释放
未映射：
只放入磁盘 I/O 队列中，等待写入完成
内存页读：
已映射：
在 RDMA I/O 队列中放入一个请求
未映射：
在磁盘 I/O 队列中放入一个请求
多页请求：批处理
当 page 出现跨 slab 边界时，即有些 page 是已映射，有些是未映射，则必须等到这一批中的页的所有请求完成才能返回
评价：
1.hash 映射并没有证明作用

【遗留问题】#哪没干好
1.Slab 的大小选择问题，怎么平衡管理开销和内存利用率以及灵活性
2.由于应用透明，针对一些冷热数据的应用无法实现特定优化[应用感知]

3.交换会引入额外的开销，比如上下文切换[系统感知]
4.应用区分，因为无法区分内存页和应用之间的对应关系，无法对每个应用限定远端内存的使用量，也就无法做到应用隔离
5.网络瓶颈,当使用 RDMA 网络的应用增加时，需要对这些竞争的应用提供网络隔离

【知识补充】
虚拟地址：
物理地址：
MMU：

@edit by QHN
2017.10.23
NJU Dislab