参考资料

- 论文地址
- B站视频
- zh分页机制
- runpod分页内存

整体概述

- PagedAttention是通过高效利用现存, 避免显存浪费, 达到提升LLm模型推理吞吐量的效果
- 以A100为例, 推理一个13B的模型, KV Cache占现存比例在30%以上, 随着batch size增大, 传统的大模型推理服务显存占用迅速上升, vllm则是有一个相对平缓很多的显存上升曲线, 同样显存占用下的每秒钟tokens吞吐量是传统方式的大概3倍

- 从这里可以看出vllm提升内存使用效率的方式, 主要是通过减少保留内存和内存碎片的占用, 具体做法是受操作系统中的虚拟内存和分页内存的启发, 将传统上必须连续存放的一条请求的 KV Cache 拆分为固定大小的逻辑块（KV Block）,再映射到实际的物理块上
- 产生这么多内存碎片的原因:
    - 输出长度不确定：大模型在推理时，生成的 token 数量只有在遇到 eos 标记后才会确定。为了防止溢出，通常需要为 KV cache 预留一段足够大的显存空间。但如果实际生成的 token 数少于预留值，多出来的空间就被浪费了。
    - 预留空间的延迟使用：即便最终生成的 token 数量正好填满了预留空间，在推理初期，大部分预留区域仍然处于闲置状态，实际并未被使用，也是一种浪费。
    - 连续内存的限制：KV cache 的分配需要一段连续的显存。当显存中剩余的碎片空间虽然总量足够，但单块小于 KV cache 的需求时，就无法满足分配，导致这些零散的内存被闲置。

pagedattention实现

连续内存模型

- 内存的低位地址用来存放操作系统, 一开始没有进程的时候内存状态如下:
- 随着不断拉起进程, 连续分配内存的情况下, 内存占用变成了:
- 当有进程退出的情况下, 内存占用:
- 随着有进程退出内存中出现了大大小小的空洞, 如果这时候想要再起一个进程, 有可能会出现剩余内存总量足够但是没有一片连续内存能够放下这个新进程的情况, 这就是连续内存模型容易出现的外部碎片（External Fragmentation）, 显存中也是同样的道理, 在pagedattention出现之前就是使用的连续内存模型, 造成了大量的显存浪费

分页内存模型

- 分页内存的核心思想是主动把内存分割成固定大小的框, 把进程的逻辑地址空间分割成大小固定的页, 框和页的大小是相同的, 通过页表(page tabel)建立进程的逻辑内存地址到物理内存地址的映射, 这样从进程的角度来看它访问的内存是连续的, 分页内存模型的缺点是从逻辑地址到物理地址的转换增加了额外的寻址开销, 当没有占满一个page的情况下浪费的这部分内存称为内部碎片(Internal Fragmentation), 当然内部碎片要比外部碎片小得多
- 把操作系统的分页内存套用到显存的管理上也是同样的道理, 只不过vllm这里不叫page而是称为KV Block, 并且KB Block是可以在内存也可以在显存的, 这里具体的调度由vllm内部管理
- vllm中的内存管理如下, block table中左边是逻辑地址到物理地址的映射, 右边是这个block中被填充了几个元素, 带有圆圈标号只是标注这个元素是推理循环中的第几轮被写入的,

复杂解码场景下的vllm实现

并行采样(Parallel sampling)

- 有时候需要对同一个prompt采样多个候选序列, vllm会将两个推理请求的prompt只在

物理显存中存储一份, 然后分别映射给两个推理进程维护的kvcache, 这样就节省了prompt的kv多次存储的显存

- vllm会额外记录物理block总共被多少个逻辑block引用了, 这样像图中为新增的token缓存kv的时候会检查这个物理block的引用次数是多少, 如果物理block的引用次数大于1会被设置为只读模式, 这里A1在写入的时候从block1复制到了block3, 然后在新增的这个block中写入, 原始物理block的引用次数由2变为1, 这样A2在写入的时候发现block1的引用次数是1就可以正常写入了, 这就是copy-on-write机制

Beam search