## Sentinel: Runtime Data Management on Heterogeneous Main Memory Systems for Deep Learning

总体概括：

当内存容量不足成为瓶颈时，我们会考虑使用异构内存来帮助提升系统的存储容量。然而为了减少异构设备的数据管理开销，我们会设计相应的模式将不同介质中的数据进行动态转移操作。本文我们重点关注DNN并对DNN在运行过程中所产生的相关数据进行了分析测试。除此之外还提出了Sentinel系统，并针对异构内存进行动态的数据转移优化。最终，Sentinel能够避免许多不必要的数据转移操作并使得使用20%的内存达到与使用全部内存相当的效率（仅损失8%的性能）。

本文贡献：

使用以数据对象为中心的方法对DNN的性能进行了系统分析；提出了一种动态运行系统来优化数据放置与转移过程；使用该系统对TensorFlow进行测试。

DNN中内存访问分析：

本文设计了Profiling框架来帮助对DNN的分析，通过页层面的一些技巧来捕获一个Tensor所需的内存访问次数、数据大小以及生产周期。

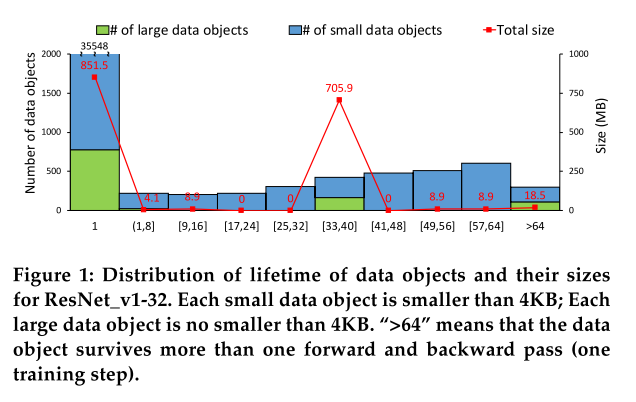


图1 不同生存周期的数据的大小

由图1可知92%的数据存活时间不超过1层，而其中98%的数据小于4kb。

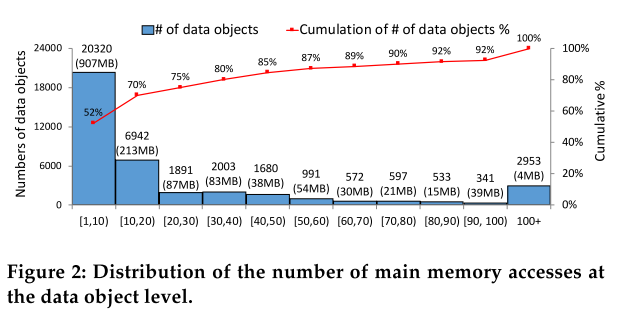


图2 数据的访问次数与数据大小的关系

由图2可得52.3%的数据使用了54%的内存，但是其数据仅被访问10次以下。其中98%的数据小于4kb，如图3所示。

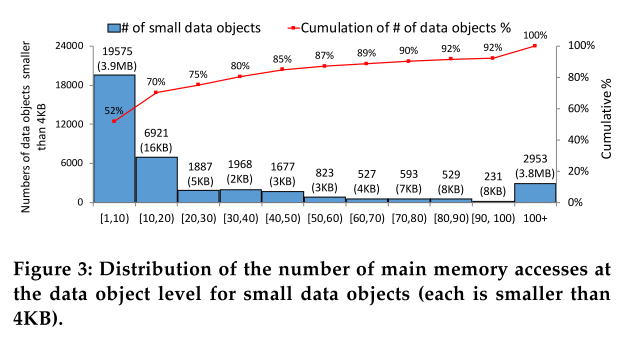


图3 小于4KB的数据访问次数与数据大小的关系

总结一下，DNN在TensorFlow框架运行时所会产生大量的小的寿命短的数据，不过这个寿命长短与访问次数并没有很明确的给出。

设计细节：

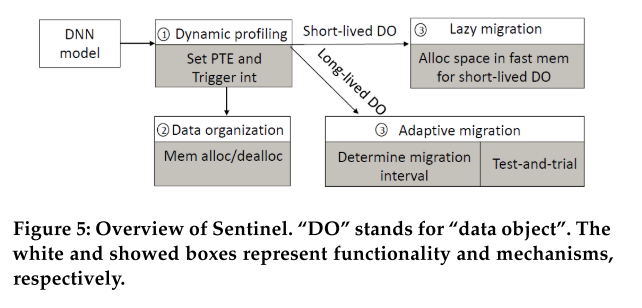


图4 Sentinel系统流程

如图4所示，Sentinel为数据在异构存储介质上的转移做了许多设计工作。首先Sentinel会动态的捕获一些基本的信息，之后会为一些寿命短的数据动态分配一些内存空间；之后针对不同类型的数据进行不同的操作以提高性能。此时Sentinel需要寻找较为合适的interval延时周期来将数据从慢速设备转移到快速设备。

重要结论：

如图5所示，在不同模型下使用不同的Memory比例进行了一系列测试，结果表明使用了60%Memory后本文所提供的方法的效果便于100%Memory相同；使用20%的Memory便能达到全Memory的92%的效果。

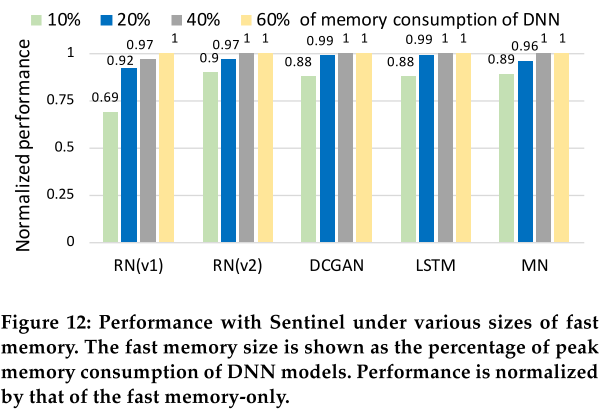


图5 Sentine在不同的Memory比例下的性能