**Memshare: a Dynamic Multi-tenant Key-value Cache**

**实验报告**

# 背景介绍

对于web应用来讲，基于动态随机存储器DRAM 的key-value caches非常重要，这一高速缓存可以显著减小应用的响应延迟，并吸收web应用中大量的数据库请求负载。而高访问速率和慢后端数据库性能意味着降低了cache的不命中率时，将直接转化为端到端应用性能的提升。当cache的平均延迟为100μs以及MySQL数据库的平均访问时间为10ms时，cache命中率每提升1%，将降低36%的cache延迟并减少2.3倍的访问数据库负载。

现存的一些DRAM caches技术静态地为共享cache的应用划分内存，这可能导致内存的利用率低且限制cache的命中率。因此，为了进一步降低延迟，提高cache命中率，作者提出了Memshare架构，同样作为DRAM cache，在保证性能隔离的前提下，能够自动动态的为应用分配池化合空闲的内存资源，并适用于多租户的情况。

Memshare通过log-structured结构设计用于存储各个应用的项目，实现隔离，并使其具有应用感知的清理功能，可在具有不同大小项目的应用程序中实现内存互换。高效内存的设计，使Memshare在单租户和多租户环境中均获得了更高的cache命中率。

# 主要实验

论文中作者进行了很多试验，其中包括使用LSM-SIM（Log-structured Merge Tree Simulation）工具来模拟应用在静态分区和池化内存两种方案下的性能表现，此外还使用microbenchmarks和YCSB框架来对CPU和内存带宽利用进行施压实验，实验对象是Memshare系统，以此来检验其性能。由于作者未给出Memshare实现源码，且实验环境要求较高，因此，我们组重点进行了LSM-SIM模拟实验，以得到不同应用的性能表现，以下简单介绍作者实验的相关内容。

## 模拟实验

模拟实验使用LSM-SIM模拟工具集来实现，它专门用于基于DRAM的Web caches，比如Memcached，可以展现不同cache实现方案下的潜在性能收益。此外，该工具集可用于比较现存的基于slab的内存分配器和提出的log-structured分配器（比如LSC）所达到的性能。如图所示为论文中一周的追踪结果，图 2‑1结果是Memcachier中静态分区和池化两种方案下三个应用分别的平均访问命中率。如图 2‑2所示为一周时间内不同应用的不命中率和内存占用。

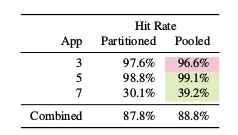


图 2‑1 两种方案下应用的平均访问率

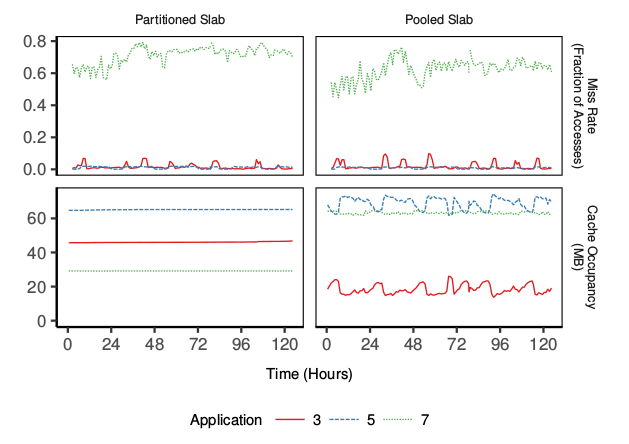


图 2‑2 两种方案下应用的不命中率和内存占用

除此之外，作者在实现Memshare的基础上进行了其他的比较实验。

## Benchmark实验

该实验为了测试实现的Memshare的性能，在YCSB框架中运行benchmark实验，分别比较5%写操作/95%读操作和100%写操作下Memshare和memcached的吞吐量对比，如图 2‑3所示为统计结果。

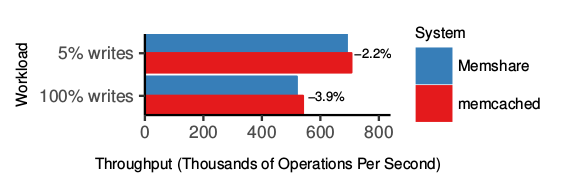


图 2‑3 不同工作量下memshare和memcached吞吐量对比

# 实验要求

作者在论文中给出了该模拟工具的github地址：https://github.com/utah-scs/lsm-sim/，查看文件结构和README后发现包含了bash脚本文件和Makefile文件，因此实验系统为Ubuntu，在Makefile文件中要求g++版本至少4.9，且使用了c++14，最终的实验环境为Ubuntu 16.04，g++版本为5.4.0，python版本为2.7.12。

# 复现实验

将模拟实验文件下载下来后，发现并没有详细的给出模拟的步骤，只能结合README文件查看文件结构并阅读相关代码后，依次进行以下实验步骤。

## 源码编译

阅读代码后发现所有脚本文件都使用了lsm-sim可执行文件，因此首先进行源码的编译，makefile文件已经给出，直接make后生成了可执行文件。如图 4‑1所示为编译结果，生成了lsm-sim文件。

make

图 ‑1 编译结果

## 脚本测试

查看其他文件夹，发现只有scripts和Tests文件夹中包含可用脚本。因此，实验的主要部分在这两个文件夹中。scripts文件夹中run-top3是用于生成论文中的数据的主要脚本，以模拟三个应用在不同策略下的性能表现，但该脚本介绍需要大量的配置信息，主要以lsm-sim来进行相关操作。

查看README和源码发现相关命令行参数如图 4‑2所示。

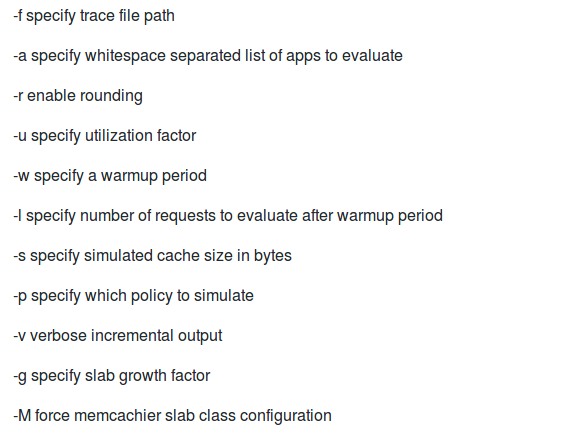


图 4‑2 命令行参数

结合脚本文件，“-f”参数后指定了trace的文件路径，但整个模拟文件中似乎缺少该文件，尝试运行脚本文件发现报错，于是确定该trace文件需要自己添加。README文件中给出了trace的文件格式，主要是CSV文件格式，每一行的格式如下：

time (floating point), app id, type (get, put etc.), key size in bytes, value size, key id, hit(1)/miss(0) bool.

其中，type表示请求类型，主要包括GET，SET，Delete，Add，Increment，Stats，Other，论文中出现GET和SET，因此主要使用这两值，分别以1和2表示。

查看Tests文件夹发现其中有部分测试使用的trace文件，仿照该文件格式给run-top3脚本指定trace路径，运行生成了很多log文件和data文件，如图 4‑3所示。log文件为标准错误输出，data文件是程序的输出，并没有给出其使用方法。



图 ‑3 run-top3脚本运行生成

其他的脚本文件运行也得到类似的结果。模拟实验的进行陷入了困境。多方查阅资料甚至阅读整个源码也没有什么收获。

但在这个过程中，对模拟实验的大致实现有一定了解。步骤大致如下：

1. 运行run-top3脚本生成实验所需数据，但trace文件不知如何给定;
2. 运行run-curves脚本利用之前生成的数据获取各应用的命中率曲线;
3. compute python文件用于统计生成应用最终命中率，输入同样是trace文件;
4. 使用其余脚本文件生成其他应用的相关数据。

# 总结

在整个复现实验过程中，刚开始看到作者在论文中给出模拟实验的网址时，我们是异常兴奋的，因为有可能完整的实现论文的部分结果，但模拟实验的部分细节的缺失使得我们的工作进行得很慢。在这期间，小组成员进行过很多此讨论，也各自查阅了资料，仔细研究了README文件，虽然最终还是没能完成实验，但我们对大致的模拟流程和LSM技术有了一定了解。