# 内存管理器性能测试

## 一、性能测试

测试一：
系统：Mac OS 10.14.5
内存管理器：

- glibc malloc
- gperftools-2.7 tcmalloc
- jemalloc 5.2.0

大内存测试：依次分配(1~1024K)大小的内存，测试单个线程的平均运行时间。

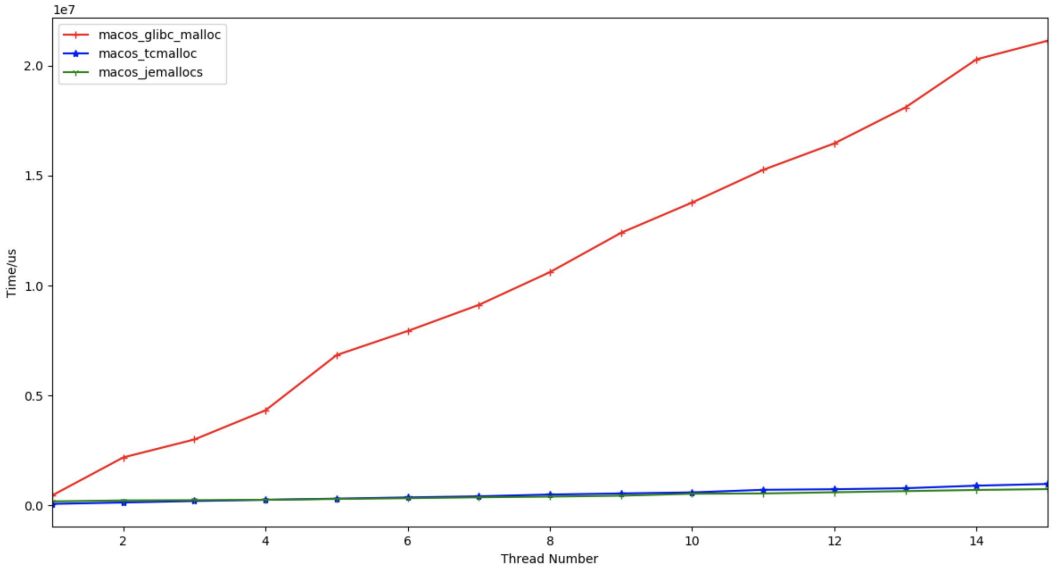


图 1 Mac OS 1~1024K 内存管理测试

在 MAC OS 环境下，glibc 自带的内存管理器，随着线程的增多，单个线程花费的时间线性增加，而使用 tcmalloc 以及 jemalloc，则消耗的时间一直在某个阈值以下，且基本保持在缓速的增长中。

测试二：
系统：Mac OS 10.14.5
内存管理器：

- glibc malloc
- gperftools-2.7 tcmalloc
- jemalloc 5.2.0

小内存测试：依次分配(1~32K)大小的内存，测试单个线程的平均时间。

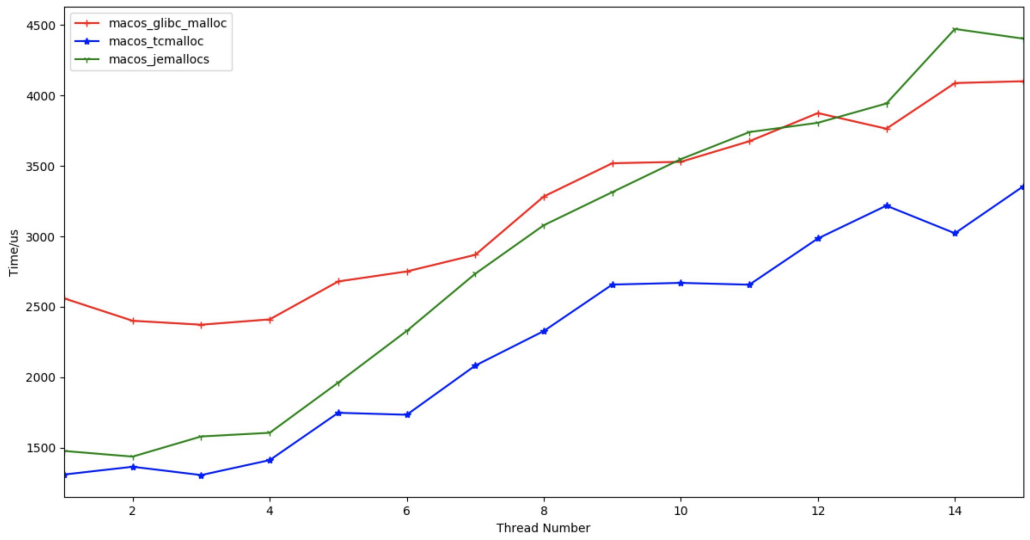


图 2 MAC OS (1~32K)内存管理测试

在小内存分配任务中，tcmalloc 性能明显优于 glibc malloc 以及 jemalloc，并且在线程数增多时，jemalloc 的耗时，甚至超过了 glibc malloc 的耗时，由此可见，在 Mac 系统中，在小内存使用中，tcmalloc 的性能更为优异。

测试三：
系统：Ubuntu 16.0
内存管理器：

- glibc malloc
- gperftools-2.7 tcmalloc
- jemalloc 5.2.0

大内存测试：依次分配(1~1024K)大小的内存，测试单个线程的平均运行时间。

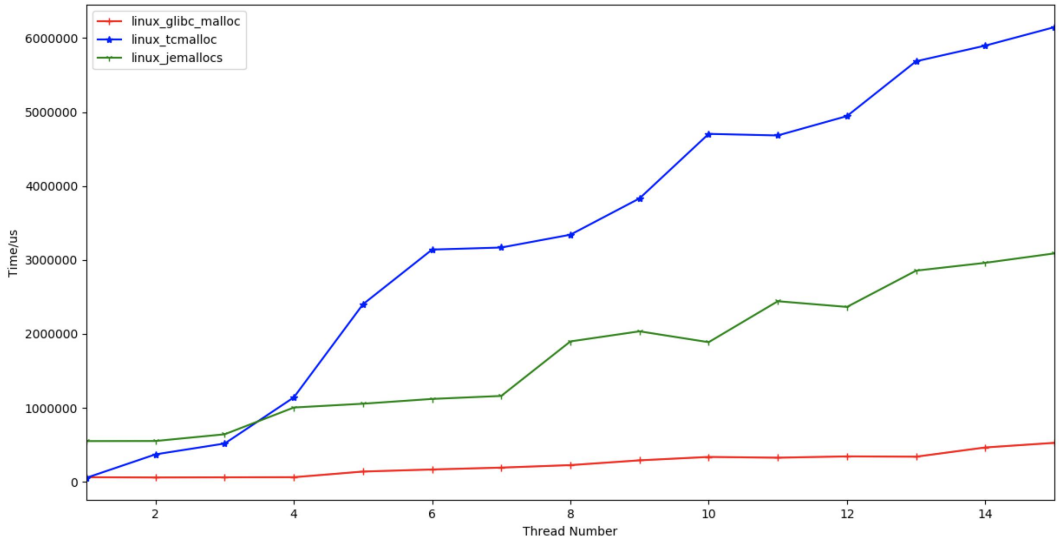


图 3 Ubuntu 16.0 (1~1024K)内存管理测试

在大内存的内存管理测试中，glibc malloc 的性能反而优于 tcmalloc 以及 jemalloc，这是因为 tcmalloc 以及 jemalloc 在申请大内存时，需要加锁，并且较大的时候，需要向 Linux 的 memmap 映射区申请内存，这些操作导致 tcmalloc 以及 jemalloc 的内存管理速度低于 glibc malloc 内存管理机制。而在大内存分配管理，tcmalloc 又没 jemalloc 性能好。

测试四：
系统：Ubuntu 16.0
内存管理器：

- glibc malloc
- gperftools-2.7 tcmalloc
- jemalloc 5.2.0

小内存测试：依次分配(1~32)大小的内存，测试单个线程的平均运行时间。

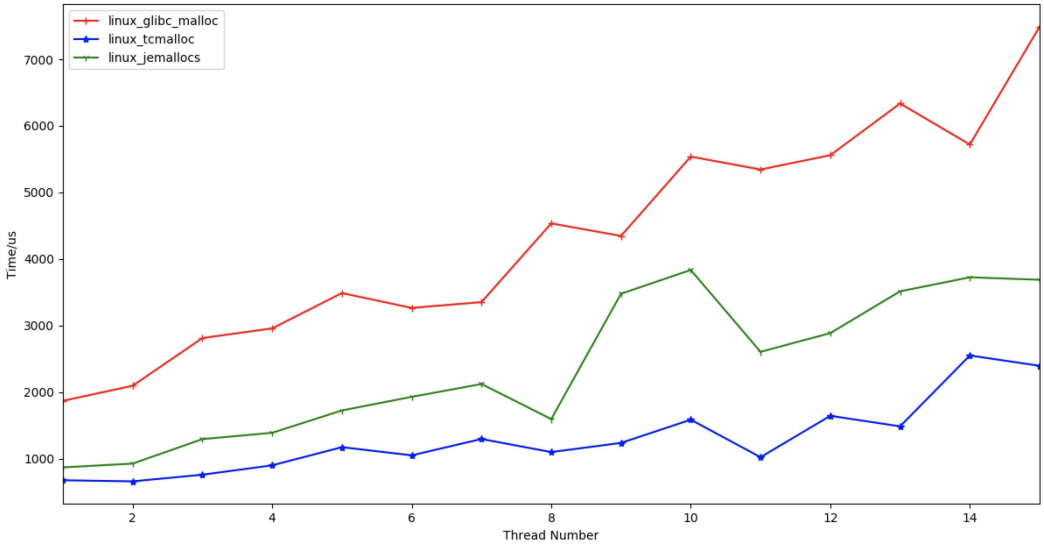


图 4 Ubuntu 16.0 (1~32K)内存管理测试

在 Linux 系统的小内存分配管理测试中，其效果与 MAC OS 测试结果相同，tcmalloc 的性能最好，jemalloc 的性能次之，glibc malloc 性能最差。tcmalloc 以及 jemalloc 随着线程数的增加，其时间消耗增长缓慢。

**结论：在大内存分配时，不同系统由于其系统低层机制的不同，不同的内存管理器具有不同的性能表现。但是在小内存时(<32K)，无论是 tcmalloc，亦或是 jemalloc 其性能都高于 glibc malloc(ptmalloc)，且 tcmalloc 较 jemalloc，具备更好的性能，远优于 glibc malloc。**

# 二、tcmalloc 与 jemalloc 的编译与使用

预安装：
autoconf, autogen 等程序包。
MAC OS：

1. `./autogen.sh` 生成configure文件
2. `./configure --prefix=PATH` 设置编译库存储的路径，并生成Makefile文件
    - jemalloc在此步骤后，include文件夹会生成相应的.h文件，后续的make则是编译该步骤生成的.h文件。
    - i. tcmalloc在此步骤，主要生成Makefile文件，无其他.h，.c文件生成。由于tcmalloc本身综合了很多工具，如果只需要使用tcmalloc，在这一步可以输入 `./configure --prefix=PATH --disable-cpu-profiler --disable-heap-profiler  --disable-heap-checker --disable-debugalloc --enable-minimal` 生成最小包
3. `make && make install` 此时则会在指定路径中生成相应的库。
4. 使用
    - 代码中无需添加 `#include<>`
    - 编译中加入 `gcc -L<PATH> -ltcmalloc_minimal` 比如 `gcc -L/usr/local/lib -ltcmalloc` 则可自动将 `malloc()`， `free()`， `new()` 等相应内存函数替换成tcmalloc机制。同理使用jemalloc内存分配器，编译选项为 `gcc -L<PATH> -ljemalloc`

## Linux:

1. `./autogen.sh` 生成configure文件
2. `./configure --prefix=PATH` 设置编译库存储的路径，并生成Makefile文件
    - jemalloc在此步骤后，include文件夹会生成相应的.h文件，后续的make则是编译该步骤生成的.h文件。
    - i. tcmalloc在此步骤，主要生成Makefile文件，无其他.h，.c文件生成。由于tcmalloc本身综合了很多工具，如果只需要使用tcmalloc，在这一步可以输入 `./configure --prefix=PATH --disable-cpu-profiler --disable-heap-profiler  --disable-heap-checker --disable-debugalloc --enable-minimal` 生成最小包
3. `make && make install` 此时则会在指定路径中生成相应的库。
4. 使用
    - 代码中无需添加 `#include<>`
    - 与MAC系统中不同的是，在Linux中，需要设置 `LD_LIBRARY_PATH` 变量，即在使用gcc编译前需要 `export LD_LIBRARY_PATH=<jemalloc/tcmalloc_lib_PATH>`，然后 `sudo ldconfig`，否则调用下面命令时，会出现错误。
    - 编译中加入 `gcc -L<PATH> -ltcmalloc_minimal` 比如 `gcc -L/usr/local/lib -ltcmalloc` 则可自动将 `malloc()`， `free()`， `new()` 等相应内存函数替换成tcmalloc机制。同理使用jemalloc内存分配器，编译选项为 `gcc -L<PATH> -ljemalloc`