google-perftools 使用积累

——google-perftools 在 NOVA.BS 性能测试中的使用

cpu-profiler

背景：

性能测试过程中发现 NOVA.BS 占用 CPU 比较高的问题，一直比较难定位和优化。

为了帮助寻找 NOVA.BS 在 CPU 资源使用方面需要优化和有优化的空间地方，尝试使用 cpu-profiler 在压力过程中对 NOVA.BS 做一次函数热点分析。根据函数热点分析的结果，可以发现占用 CPU 资源比较高的函数集，从而制定相应的优化策略。

工作原理：

cpu-profiler 的工作原理是通过 CPU 中断采样的方式来统计每个函数被采样的次数，每个函数被采样的次数占总采样次数的比例大致就是该函数在采样时间段内占用 CPU 资源比例。

采样次数占总采样次数比例越高的函数，就表示其占用 CPU 资源越高，也就越有优化的空间。

使用方式：

cpu-profiler 的使用方式有很多，具体可以参考该工具的主页：
http://google-perftools.googlecode.com/svn/trunk/doc/cpuprofile.html

下面说明 NOVA.BS 性能测试中使用的方式的具体步骤：

1. **编译可采集函数性能数据的 NOVA.BS**
   - NOVA.BS 代码中添加一个信号 SIGUSR2 的处理函数 gprof_callback

```
signal(SIGUSR2, gprof_callback);
```

   - 在 gprof_callback 中调用 ProfilerStart 和 ProfilerStop

     注：编译时需要包含 google/profiler.h 文件，链接时需要链接 libprofiler.a 库文件

```
#include <google/profiler.h>

static void gprof_callback(int sig)
{
    bool static perf = false;
    if(!perf) {
        ProfilerStart( "bs.prof" );
    }
    else {
        ProfilerStop();
    }
    perf = !perf;
}
```

- 如果是多线程程序，需要在每个线程的初始化函数处调用一次 ProfilerRegisterThread

```
int init_thread_data()
{
    ProfilerRegisterThread();
```

- 在 64 位系统中系统自带的堆栈解旋库可能会导致 cpu-profiler 无法正常工作，这时需要联编开源库 libunwind.a 来解决这个问题

2. 使用步骤 1 中编译出来的 NOVA.BS 采集压力情况下的性能数据

- 部署 NOVA.BS 压力环境，启动压力
- 发送 SIGUSR2 信号至 NOVA.BS 开始采集性能数据（kill -SIGUSR2 pid 即可）
- 压力一段时间后，再次发送 SIGUSR2 信号至 NOVA.BS 停止采集性能数据
- NOVA.BS 会在当前目录下产生一个名为 bs.prof 的二进制文件（文件名由 ProfilerStart 函数的参数控制）

3. 使用 pprof 工具，统计步骤 2 中产生的性能数据

步骤 2 中产生的性能数据并不是可读的，可以通过 google-perftools 提供的一个名为 pprof 的工具将其转化为可读的文本格式或者图片格式。

NOVA.BS 的性能数据目前会转化为 text 格式和 gif 格式：

- text 格式的输出，可以方便查看各个函数占用 CPU 资源
- gif 格式的输出可以更方便直观地看到函数的调用关系图以及函数调用过程中各个函数占用 CPU 资源的多少

注：转化 gif 格式的输出，需要安装 dot 工具，可以通过安装 graphviz 来获取。

pprof 使用方式：（更多选项可使用 pprof –help 查看）

- pprof --text nova.bs bs.prof > bsfunc.text
- pprof --gif nova.bs bs.prof > bsfunc.gif

Text 输出结果：

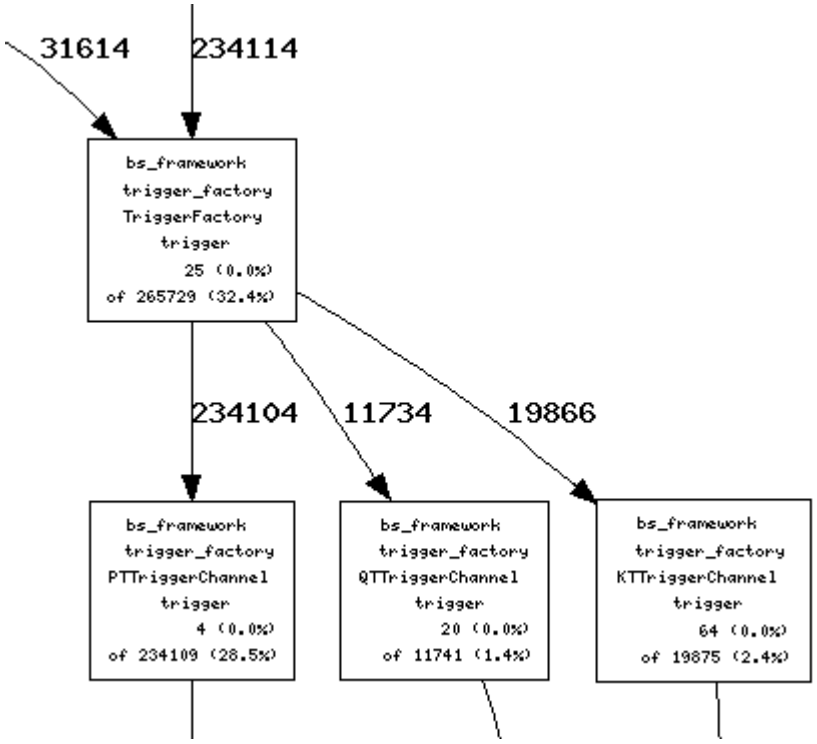
```
Total: 786061 samples
  125211  15.9%  15.9%   130685  16.6% BDADInfo::make_unit_key
   32878   4.2%  20.1%    42494   5.4% kc_trigger_process
   28804   3.7%  23.8%    29508   3.8% searcher::container::VBitMap::set
   22744   2.9%  26.7%    56512   7.2% BDADInfo::build
   21747   2.8%  29.4%    21869   2.8% SiteUnitPrice::_build_custom_dict
   21627   2.8%  32.2%    21627   2.8% searcher::container::VBitMap::get
   17718   2.3%  34.4%    17718   2.3% RegionFilter::is_filter
```

第一行统计了本次函数级性能采集过程中总的采样次数

从第二行开始，每行各有六列，分别表示：

- ✧ 第一列是该函数采样的次数
- ✧ 第二列是该函数采样次数占整个采样次数的比例
- ✧ 第三列是打印到这行为止总采样次数占整个采样次数的比例
- ✧ 第四列是该函数以及调用的子函数的采样次数
- ✧ 第五列是该函数以及调用的子函数的采样次数占整个采样次数的比例
- ✧ 第六列是该函数的函数名

Gif 输出结果：



其中，每个节点表示一个函数。除了叶子节点外，每个节点内部都有一“X1(Y1%) of X2(Y2%)”，叶子节点只有 X1(Y1%)。

X1 表示该函数采样的次数，Y1 表示该函数采样次数占整个采样次数的比例，X2 表示该函数以及调用的子函数的采样次数，Y2 表示该函数以及调用的子函数的采样次数占整个采样次数的比例。

节点之间的箭头，表示父子函数之间的调用；箭头旁的数字表示该对父子函数调用的采样次数。

heap-profiler

背景：

性能测试过程中发现 NOVA.BS 占用内存比较高以及疑似内存泄露的问题，一直比较难以定位和修复。

因此尝试使用 heap-profiler 工具对 NOVA.BS 的内存使用做函数级别的分析，并通过对不同阶段的结果进行多维度的统计，期待完成两个功能：

- 按函数、模块统计出 NOVA.BS 的占用
- 统计 NOVA.BS 压力过程中的内存申请，判断是否存在内存泄露

工作原理：

heap-profiler 的工作原理是 hook 了内存分配和释放的操作，并记录了每次的内存分配的堆栈和地址。根据内存地址，将内存分配和释放配对相减，就可以统计出最终申请内存的大小以及申请该内存堆栈信息。

使用方式：

heap-profiler 的使用方式有很多，具体可以参考该工具的主页：
http://google-perftools.googlecode.com/svn/trunk/doc/heapprofile.html

下面说明 NOVA.BS 性能测试中使用的方式的具体步骤：

1. **编译可采集内存分配数据的 NOVA.BS**
   - NOVA.BS 代码 main 函数开始处调用 HeapProfilerStart，结束处调用 HeapProfilerStop

```
//start heap profiling
HeapProfilerStart("bs.hprof");
```

```
HeapProfilerStop();
```

   - 在 NOVA.BS 启动完成的时候，调用 HeapProfilerDump

```
HeapProfilerDump("after_load");
```

   - NOVA.BS 代码中添加一个信号 SIGRTMIN+1 的处理函数 heapprof_callback

```
signal(SIGRTMIN + 1, heapprof_callback);
```

   - 在 heapprof_callback 中调用 HeapProfilerDump

     注：编译时需要包含 google/heap-profiler.h 文件，链接时需要链接 libtcmalloc.a 库文件

```
#include <google/heap-profiler.h>

static void heapprof_callback(int sig)
{
    HeapProfilerDump("signal");
}
```

   - 在 64 位系统中系统自带的堆栈解旋库可能会导致 heap-profiler 无法正常工作，这时需要联编开源库 libunwind.a 来解决这个问题

2. 使用步骤 1 中编译出来的 NOVA.BS 采集 NOVA.BS 不同阶段的内存分配数据

- env HEAP_PROFILE_INUSE_INTERVAL=1073741824000
- env HEAP_PROFILE_ALLOCATION_INTERVAL=1073741824000
- 部署 NOVA.BS 压力环境，NOVA.BS 启动完成之后，会在当前目录生成 bs.hprof.0001.heap 文件
- 启动压力，一段时间之后发送信号 SIGRTMIN+1 至 NOVA.BS（kill -35 pid），会在当前目录生成 bs.hprof.0002.heap 文件
- 停止压力，循环 touch data，使得 NOVA.BS 循环 reload data，一段时间之后发送 SIGRTMIN+1 至 NOVA.BS，会在当前目录生成 bs.hprof.0003.heap 文件

3. 使用 pprof 工具，统计步骤 2 中产生的性能分配数据

步骤 2 中，统计了 NOVA.BS 三个阶段内存分配数据：
bs.hprof.0001.heap 中统计了 NOVA.BS 启动阶段的内存分配
bs.hprof.0002.heap 中统计了 NOVA.BS 启动阶段的内存分配+压力过程中的内存分配
bs.hprof.0003.heap 中统计了 NOVA.BS 启动阶段的内存分配+压力过程中的内存分配+NOVA.BS reload data 过程中的内存分配
因此，根据三个 heap 数据，可以分别统计出 NOVA.BS 启动阶段、压力过程中、reload 过程中内存分配：
NOVA.BS 启动阶段内存分配 = bs.hprof.0001.heap
NOVA.BS 压力过程内存分配 = bs.hprof.0002.heap - bs.hprof.0001.heap
NOVA.BS reload 过程内存分配 = bs.hprof.0003.heap - bs.hprof.0002.heap

内存分配的原始数据是可读的，格式如下：

```
heap profile: 290186: 25091314798 [443655: 26605426881] @ heapprofile
   210: 1612773120 [   210: 1612773120] @ 0x0057a47c 0x0057a0a8 0x0059eb44 0x00be461d
   356: 1493166528 [   356: 1493166528] @ 0x00570e2d 0x0057055d 0x0061a346 0x00615e93
   270: 1132434000 [   270: 1132434000] @ 0x00570e2d 0x0057055d
```

第一行统计了 NOVA.BS 到 dump 内存分配数据的时候总的内存分配情况：

- 290186 表示 NOVA.BS 有效分配内存的次数（有效分配内存是指分配的内存减去释放的内存）
- 25091314798 表示 NOVA.BS 有效分配内存的数量，单位字节
- 443655 表示 NOVA.BS 总的内存分配次数（所有的内存分配，包括分配之后被释放的）
- 26605426881 表示 NOVA.BS 总的内存分配数量，单位字节

第二行之后，是按堆栈统计的内存分配，每一行表示一个不同的堆栈分配内存的信息：

- @之前的部分，是有效的和总的内存分配次数和数量，和第一行类似
- @之后的部分，是一个个的地址，从前往后依次是分配内存操作的堆栈的由顶到底的地址信息

这样的数据虽然是可读的，但是由于堆栈都是一个个地址，并不能看出某个函数启动过程中申请了多少内存，而且如果要计算 NOVA.BS 压力过程某个函数分配内存，还需要用 bs.hprof.0002.heap 的数据减去 bs.hprof.0001.heap 的数据。
所幸，与处理 cpu-profiler 结果类似，pprof 提供了将 heap 数据转化成文本和图片格式的功能，同时还提供了比较两个 heap 数据的功能（--base 选项）。
注：转化 gif 格式的输出，需要安装 dot 工具，可以通过安装 graphviz 来获取。

pprof 使用方式：（更多选项可使用 pprof –help 查看）

- 以文本方式统计 NOVA.BS 启动过程内存分配

pprof --text nova.bs bs.hprof.0001.heap > bsmem_startup.text

- 以图片方式统计 NOVA.BS 启动过程内存分配

  pprof --gif nova.bs bs.hprof.0001.heap > bsmem_startup.gif
- 以文本方式统计 NOVA.BS 压力过程内存分配

  pprof --text --base=bs.hprof.0001.heap nova.bs bs.hprof.0002.heap > bsmem_press.text
- 以图片方式统计 NOVA.BS 压力过程内存分配

  pprof --gif --base=bs.hprof.0001.heap nova.bs bs.hprof.0002.heap > bsmem_press.gif

Text 输出结果：

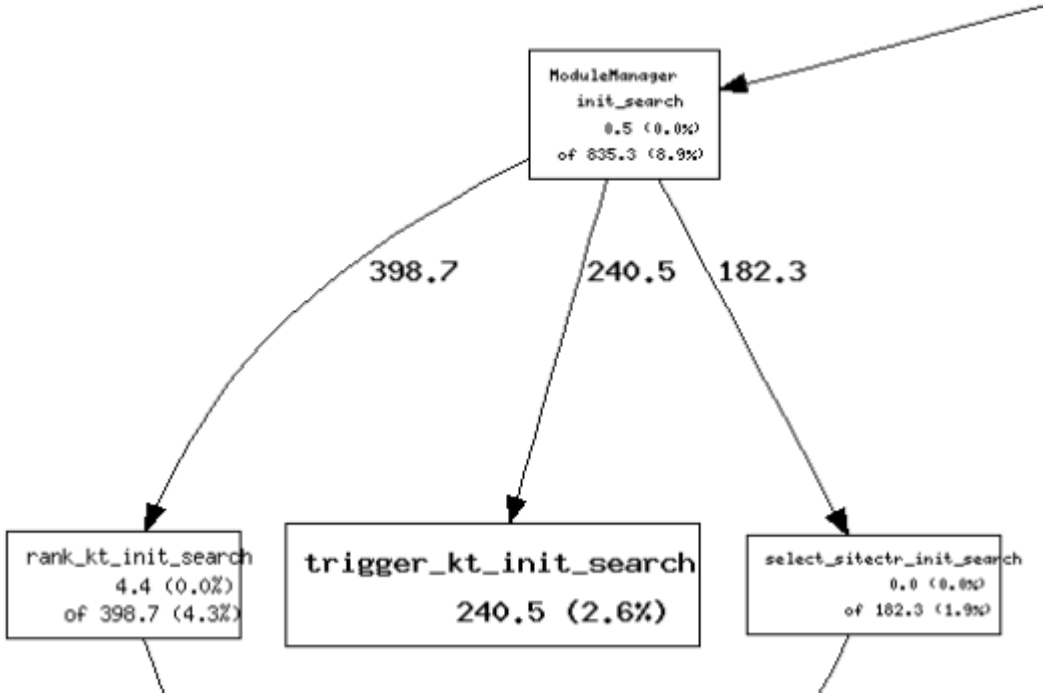
```
Total: 34974.3 MB
 24370.3  69.7%  69.7%  24370.3  69.7% st::TRCMemoryPool::alloc_mem
  8022.2  22.9%  92.6%   8022.2  22.9% afs::read_to_map
  1024.1   2.9%  95.5%   1024.1   2.9% st::BaseCowHashMap::resize
   582.0   1.7%  97.2%    731.2   2.1% NOVA_BS::load_cluster_index
   292.9   0.8%  98.0%    292.9   0.8% ul_malloc
   262.3   0.7%  98.8%    262.3   0.7% ul_calloc
    99.2   0.3%  99.1%    157.6   0.5% NOVA_BS::load_pt_relevance_index
    96.0   0.3%  99.4%    120.0   0.3% LocalIDManager::push
    65.6   0.2%  99.5%     65.6   0.2% std::vector::reserve
```

第一行统计了总内存分配的数量

从第二行开始，每行各有六列，分别表示：

- 第一列是该函数内存分配数量，单位 MB
- 第二列是该函数内存分配占总内存分配的比例
- 第三列是打印到这行为止内存分配占总内存分配的比例
- 第四列是该函数以及调用的子函数的内存分配数量，单位 MB
- 第五列是该函数以及调用的子函数的内存分配数量占总内存分配的比例
- 第六列是该函数的函数名

Gif 输出结果：



其中，每个节点表示一个函数。除了叶子节点外，每个节点内部都有一个“X1(Y1%) of X2(Y2%)”信息，叶子节点只有 X1(Y1%)。

X1 表示该函数内存分配数量（单位 MB），Y1 表示该函数内存分配占总内存分配的比例，X2 表示该函数以及调用的子函数的内存分配数量（单位 MB），Y2 表示该函数以及调用的子函数的内存分配占总内存分配的比例。

节点之间的箭头，表示父子函数之间的调用；箭头旁的数字表示该对父子函数调用的产生内存分配数量（单位 MB）。