# Lab 1++: Tune Your FS!

姓名：程可

学号：518021910095

## 1. 初始性能差距

Native file system:

```
root@ics:/mnt/hgfs/lab-cse/lab1# ./fxmark/bin/fxmark --type=YFS --root=./native
--ncore=1 --duration=5
# ncpu secs works works/sec
1 5.097093 3968.000000 778.482951
```

YFS:

```
root@ics:/mnt/hgfs/lab-cse/lab1# ./fxmark/bin/fxmark --type=YFS --root=./yfs1 --
ncore=1 --duration=5
# ncpu secs works works/sec
1 6.199136 512.000000 82.592155
```

## 2. 寻找性能瓶颈

为了寻找性能瓶颈，我调用了_rdtsc()函数来测量某一个函数的微观的CPU cycle数，测量方法如下：

```
#define func_num 18
#define CYCLE
int call_count[func_num] = {0};
unsigned long long max_cycle[func_num] = {0};
unsigned long long min_cycle[func_num] = {0};
FILE * fp;

unsigned long long get_current_time()
{
  return __rdtsc();
}

void calculate_cycle(int func_no, unsigned long long start)
{
  unsigned long long end = _rdtsc();
  unsigned long long cycles = end - start;
  call_count[func_no]++;
  if (cycles > max_cycle[func_no])
  {
    max_cycle[func_no] = cycles;
  }
  if (min_cycle[func_num] == 0 || cycles < min_cycle[func_num])
```

```
  {
    min_cycle[func_no] = cycles;
  }
  fp = fopen("cycle.txt", "a");
  fprintf(fp, "%s%d %s %d %s %s %lld  %s %lld\n", "Function", func_no, "called",
call_count[func_no], "times", "Max cycle:", max_cycle[func_no], "Min cycle:",
min_cycle[func_no]);

  fclose(fp);
}
```

测量的时候只要将

```
#ifdef CYCLE
unsigned long long start = get_current_time();
#endif
```

加在被测试代码片段的开头， 将

```
#ifdef CYCLE
calculate_cycle(0, start);
#endif
```

加在被测试代码片段的结束即可测试。

## inode_manager.cc

| 函数名 | 被调用次数 | 最大cycle数 | 最小cycle数 |
| --- | --- | --- | --- |
| disk::disk | 1 | 25697838 | 25697838 |
| disk::read_block | 7184 | 26331 | 535 |
| disk::write_block | 3977 | 25660 | 315 |
| block_manager::alloc_block | 0 | / | / |
| block_manager::free_block | 1152 | 26232 | 2735 |
| block_manager::block_manager | 1 | 30166784 | 30166784 |
| block_manager::read_block | 6031 | 12435752 | 626982 |
| block_manager::write_block | 2823 | 32497106 | 1113395 |
| inode_manager::inode_manager | 1 | 60321564 | 60321564 |
| inode_manager::alloc_inode | 130 | 29127416 | 8156378 |
| inode_manager::free_inode | 128 | 40844874 | 9532312 |
| inode_manager::get_inode | 3592 | 17004523 | 2183364 |
| inode_manager::put_inode | 1540 | 41429798 | 4739086 |
| inode_manager::read_file | 1027 | 51937976 | 9201138 |
| inode_manager::write_file | 385 | 187636712 | 14446537 |
| inode_manager::getattr | 1924 | 33136153 | 3311680 |
| inode_manager::remove_file | 128 | 69029921 | 23563854 |

## yfs_client.cc

1.

| 函数名 | 被调用次数 | 最大cycle数 | 最小cycle数 |
| --- | --- | --- | --- |
| yfs_client::yfs_client | 1 | 96045415 | 96045415 |
| yfs_client::yfs_client | 0 | / | / |
| yfs_client::n2i | 0 | / | / |
| yfs_client::add_entry_and_save | 129 | 61130823 | 38497621 |
| yfs_client::has_duplicate | 0 | / | / |
| yfs_client::filename | 0 | / | / |
| yfs_client::isfile | 1154 | 31183082 | 5468489 |
| yfs_client::isdir | 257 | 13888232 | 5401466 |
| yfs_client::getfile | 256 | 39805603 | 5587524 |
| yfs_client::getdir | 257 | 25884116 | 5226918 |
| yfs_client::getlink | 0 | / | / |
| yfs_client::setattr | 0 | / | / |
| yfs_client::create | 128 | 109172280 | 91354277 |
| yfs_client::mkdir | 1 | 64896681 | 64896681 |
| yfs_client::lookup | 642 | 55766159 | 15092110 |
| yfs_client::readdir | 899 | 54444302 | 12821005 |
| yfs_client::writedir | 257 | 39362426 | 14467083 |
| yfs_client::read | 0 | / | / |
| yfs_client::write | 128 | 172767198 | 99782224 |
| yfs_client::readlink | 0 | / | / |
| yfs_client::symlink | 0 | / | / |
| yfs_client::unlink | 128 | 132078204 | 79125227 |

# 3. 数据分析

通过分析以上数据可以发现：

1. 在 inode_manager.cc 中，disk::read_block，disk::write_block，inode_manager::get_inode，inode_manager::put_inode，inode_manager::get_inode，inode_manager::getattr是相对被调用次数比较多的，而其中消耗CPU cycle数最多的是block_manager::read_block。但是这个函数是直接调用disk::read_block的，而disk::read_block并没有太大的优化空间，所以只能放弃。此外，可以发现inode_manager::write_file虽然被调用次数不多，但是它消耗的CPU cycle数量却比别的函数大了一个数量级
2. 在 yfs_client.cc中，yfs_client::isfile是被调用次数最多的，并且它消耗的CPU cycle数量也不小。此外，yfs_client::create和yfs_client::write消耗的CPU cycle数量也比别的函数大了整整一个数量

级。

# 4. 优化

## 4.1 对yfs_client.cc的优化

通过第三节分析可以发现yfs_client::isfile消耗了数量可观的CPU cycle，而且其实它的逻辑十分简单，只是判断一个inode的类型是否为文件。与之逻辑类似的还有yfs_client::isdir。因此我想到了在yfs_client中加入缓存来记录一个inode的类型，这样就不用调用下层函数来获取inode的类型了。具体实现为：

首先加入缓存结构体，也就是一个std::map:

```
std::map <yfs_client::inum, extent_protocol::types> inode_type;
```

其次将yfs_client::isfile和yfs_client::isdir调用下层函数的逻辑改为直接访问缓存，如果缓存中没有再调用下层函数：

```
        if (inode_type.count(inum) > 0)
        {
            if (inode_type[inum] == extent_protocol::T_FILE)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        if (ec->getattr(inum, a) != extent_protocol::OK) {
        #ifdef CACHE
            inode_attr.insert({inum, a});
        #endif
        #ifdef TYPE_CACHE
            inode_type[inum] = (extent_protocol::types)a.type;
        #endif
        #ifdef PRINTF
        printf("error getting attr\n");
        #endif
        #ifdef CYCLE
        calculate_cycle_yfs(6, start);
        #endif
        return false;
    }

    if (a.type == extent_protocol::T_FILE) {
        #ifdef PRINTF
        printf("isfile: %lld is a file\n", inum);
        #endif
        #ifdef CYCLE
        calculate_cycle_yfs(6, start);
        #endif
        return true;
    }
```

同时，考虑到consistency的问题，要在inode被remove之后更新缓存：

```
inode_type.erase(it->inum);
```

经过如此优化，性能为：

```
root@ics:/mnt/hgfs/lab-cse/lab1# ./fxmark/bin/fxmark --type=YFS --root=./yfs1 --
ncore=1 --duration=5
# ncpu secs works works/sec
1 5.676626 768.000000 135.291633
```

可以发现相较原来有显著提升。

## 4.2 对inode_manager.cc的优化

通过第三节的分析，我将inode_manager::write_file和inode_manager::read_file作为优化目标，这是因为通过第三节分析可以发现inode_manager::write_file消耗了大量的cycle数，而inode_manager::read_file的逻辑与inode_manager::write_file相类似。通过分析这两个函数可以发现，它们主要部分是循环语句，类似于如下代码段：

```
    for(int i = 0;i < (new_block_num < NDIRECT ? new_block_num : NDIRECT);++i)
    {
      //If it is the last block, there may be some empty space in the last block
      if(i == (new_block_num - 1))
      {
        char padding[BLOCK_SIZE];
        bzero(padding,BLOCK_SIZE);
        memcpy(padding, buf + i * BLOCK_SIZE, size - i * BLOCK_SIZE);
        bm->write_block(ino->blocks[i], padding);
      }
      else
      {
        bm->write_block(ino->blocks[i], buf + i * BLOCK_SIZE);
      }
    }
```

因此我想到了采用循环展开的方法：

```
    int i;
    for(i = 0;i < (new_block_num < NDIRECT ? new_block_num : NDIRECT) - 1;i+=2)
    {
      //If it is the last block, there may be some empty space in the last block
      if(i == (new_block_num - 1))
      {
        char padding[BLOCK_SIZE];
        bzero(padding,BLOCK_SIZE);
        memcpy(padding, buf + i * BLOCK_SIZE, size - i * BLOCK_SIZE);
        bm->write_block(ino->blocks[i], padding);
      }
      else
      {
        bm->write_block(ino->blocks[i], buf + i * BLOCK_SIZE);
      }
```

```
      if(i == (new_block_num - 1) - 1)
      {
        char padding[BLOCK_SIZE];
        bzero(padding,BLOCK_SIZE);
        memcpy(padding, buf + (i+1) * BLOCK_SIZE, size - (i+1) * BLOCK_SIZE);
        bm->write_block(ino->blocks[i+1], padding);
      }
      else
      {
        bm->write_block(ino->blocks[i+1], buf + (i+1) * BLOCK_SIZE);
      }
    }
    for(;i < (new_block_num < NDIRECT ? new_block_num : NDIRECT);i++)
    {
      if(i == (new_block_num - 1))
      {
        char padding[BLOCK_SIZE];
        bzero(padding,BLOCK_SIZE);
        memcpy(padding, buf + i * BLOCK_SIZE, size - i * BLOCK_SIZE);
        bm->write_block(ino->blocks[i], padding);
      }
      else
      {
        bm->write_block(ino->blocks[i], buf + i * BLOCK_SIZE);
      }
    }
```

在优化之前，性能表现为：

```
root@ics:/mnt/hgfs/lab-cse/lab1# ./fxmark/bin/fxmark --type=YFS --root=./yfs1 --
ncore=1 --duration=5
# ncpu secs works works/sec
1 5.676626 768.000000 135.291633
```

在优化之后，性能表现为：

```
root@ics:/mnt/hgfs/lab-cse/lab1# ./fxmark/bin/fxmark --type=YFS --root=./yfs1 --
ncore=1 --duration=5
# ncpu secs works works/sec
1 5.676626 768.000000 134.328921
```

发现性能并未得到提升。

# 5. 总结

经过优化后YFS相较于native file system的性能差距从9.49倍缩小为了6.22倍。

最终结果：

```
root@ics:/mnt/hgfs/lab-cse/lab1# ./fxmark/bin/fxmark --type=YFS --root=./native
--ncore=1 --duration=5
# ncpu secs works works/sec
1 5.025576 4224.000000 840.500671
root@ics:/mnt/hgfs/lab-cse/lab1# ./fxmark/bin/fxmark --type=YFS --root=./yfs1 --
ncore=1 --duration=5
# ncpu secs works works/sec
1 5.676626 768.000000 135.291633
```