深入浅出lz4压缩算法

小拉风 关注

0.641 2018.09.12 11:50:51 字数 620 阅读 18,671

简介

lz4是目前综合来看效率最高的压缩算法，更加侧重压缩解压速度，压缩比并不是第一。在当前的安卓和苹果操作系统中，内存压缩技术就使用的是lz4算法，及时压缩手机内存以带来更多的内存空间。本质上是时间换空间。

压缩原理

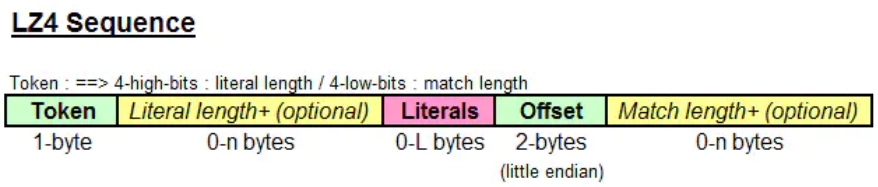
lz4压缩算法其实很简单，举个压缩的栗子

```
输入：abcde_bcdefgh_abcdefghxxxxxxx
输出：abcde_(5,4)fgh_(14,5)fghxxxxxxx
```

其中两个括号内的便代表的是压缩时检测到的重复项，(5,4) 代表向前5个byte，匹配到的内容长度有4，即"bcde"是一个重复。当然也可以说"cde"是个重复项，但是根据算法实现的输入流扫描顺序，我们取到的是第一个匹配到的，并且长度最长的作为匹配。

1.压缩格式

压缩后的数据是下面的格式




```
输入：abcde_bcdefgh_abcdefghxxxxxxx
输出：tokenabcde_(5,4)fgh_(14,5)fghxxxxxxx
格式：[token]literals(offset,match length)[token]literals(offset,match length)....
```

其他情况也可能有连续的匹配：

```
输入：fghabcde_bcdefgh_abcdefghxxxxxxx
输出：fghabcde_(5,4)(13,3)_(14,5)fghxxxxxxx
格式：[token]literals(offset,match length)[token](offset,match length)....
这里(13,3)长度3其实并不对，match length匹配的长度默认是4
```

Literals指没有重复、首次出现的字节流，即不可压缩的部分
Match指重复项，可以压缩的部分
Token记录literal长度，match长度。作为解压时候memcpy的参数

2.压缩率

可以想到，如果重复项越多或者越长，压缩率就会越高。上述例子中"bcde"在压缩后，用(5,4)

推荐阅读

阿里腾讯头条快手都在用的Clickhouse到底是什么?
阅读 1,365

C++ primer 第十四章-重载操作符和Class-type转换
阅读 128

MySQL技术内幕(InnoDB存储引擎)
阅读 369

20200819 行人重识别算法比赛
阅读 1,407

一道有意思的腾讯算法面试题
阅读 2,231

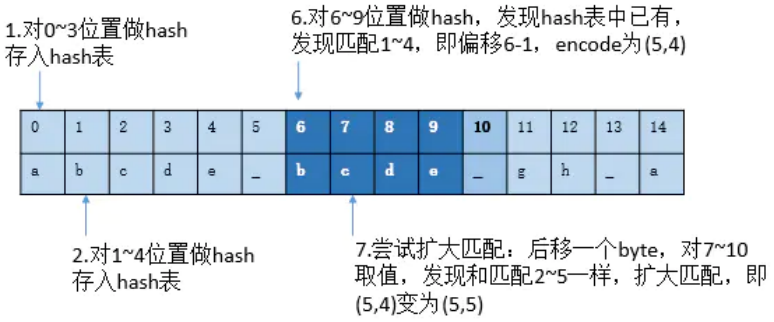
深入浅出lz4压缩算法

小拉风 关注 赞赏支持

大致流程，压缩过程以至少4个bytes为扫描窗口查找匹配，每次移动1byte进行扫描，遇到重复的就进行压缩。

由于offset用2bytes表示，只能查找到到2^16(64kb)距离的匹配，对于压缩4Kb的内核页，只需要用到12位。

扫描的步长1byte是可以调整的，即对应LZ4_compress_fast机制，步长变长可以提高压缩解压速度，减少压缩率。



我们来看下apple的lz4实现

```
//src是输入流，dst是输出，还需要使用一个hash表记录前面一段距离内的字符串，用来查找之前是否有匹配
void lz4_encode_2gb(uint8_t ** dst_ptr,
                    size_t dst_size,
                    const uint8_t ** src_ptr,
                    const uint8_t * src_begin,
                    size_t src_size,
                    lz4_hash_entry_t hash_table[LZ4_COMPRESS_HASH_ENTRIES],
                    int skip_final_literals)
{
  uint8_t *dst = *dst_ptr;        // current output stream position
  uint8_t *end = dst + dst_size - LZ4_GOFAST_SAFETY_MARGIN;
  const uint8_t *src = *src_ptr;  // current input stream literal to encode
  const uint8_t *src_end = src + src_size - LZ4_GOFAST_SAFETY_MARGIN;
  const uint8_t *match_begin = 0; // first byte of matched sequence
  const uint8_t *match_end = 0;   // first byte after matched sequence
//苹果这里使用了一个early abort机制，即输入流扫描到lz4_do_abort_eval位置的时候，仍然没有匹配，则认为
#if LZ4_EARLY_ABORT
  uint8_t * const dst_begin = dst;
  uint32_t lz4_do_abort_eval = lz4_do_early_abort;
#endif

  while (dst < end)
  {
    ptrdiff_t match_distance = 0;
    //for循环一次查找到一个match即跳出到EXPAND_FORWARD
    for (match_begin = src; match_begin < src_end; match_begin += 1) {
      const uint32_t pos = (uint32_t)(match_begin - src_begin);
      //苹果这里实现比较奇怪，还在思考为何同时查找连续四个bytes的匹配
      const uint32_t w0 = load4(match_begin);//该位置4个bytes的内容
      const uint32_t w1 = load4(match_begin + 1);
      const uint32_t w2 = load4(match_begin + 2);
      const uint32_t w3 = load4(match_begin + 3);
      const int i0 = lz4_hash(w0);
      const int i1 = lz4_hash(w1);
      const int i2 = lz4_hash(w2);
      const int i3 = lz4_hash(w3);
      const uint8_t *c0 = src_begin + hash_table[i0].offset;
      const uint8_t *c1 = src_begin + hash_table[i1].offset;
      const uint8_t *c2 = src_begin + hash_table[i2].offset;
      const uint8_t *c3 = src_begin + hash_table[i3].offset;
      const uint32_t m0 = hash_table[i0].word;//取出hash表中以前有没有一样的值
      const uint32_t m1 = hash_table[i1].word;
      const uint32_t m2 = hash_table[i2].word;
      const uint32_t m3 = hash_table[i3].word;
      hash_table[i0].offset = pos;
      hash_table[i0].word = w0;
      hash_table[i1].offset = pos + 1;
```

推荐阅读

阿里腾讯头条快手都在用的Clickhouse到底是什么?
阅读 1,365

C++ primer 第十四章-重载操作符和Class-type转换
阅读 128

MySQL技术内幕(InnoDB存储引擎)
阅读 369

20200819 行人重识别算法比赛
阅读 1,407

一道有意思的腾讯算法面试题
阅读 2,231

写下你的评论... 评论0 赞7 ···

深入浅出lz4压缩算法

小拉风 关注 赞赏支持

```
      hash_table[i3].word = w3;

      match_distance = (match_begin - c0);
      //比较hash表中的值和当前指针位置的hash值
      if (w0 == m0 && match_distance < 0x10000 && match_distance > 0) {
        match_end = match_begin + 4;
        goto EXPAND_FORWARD;
      }

      match_begin++;
      match_distance = (match_begin - c1);
      if (w1 == m1 && match_distance < 0x10000 && match_distance > 0) {
        match_end = match_begin + 4;
        goto EXPAND_FORWARD;
      }

      match_begin++;
      match_distance = (match_begin - c2);
      if (w2 == m2 && match_distance < 0x10000 && match_distance > 0) {
        match_end = match_begin + 4;
        goto EXPAND_FORWARD;
      }

      match_begin++;
      match_distance = (match_begin - c3);
      if (w3 == m3 && match_distance < 0x10000 && match_distance > 0) {
        match_end = match_begin + 4;
        goto EXPAND_FORWARD;
      }

#if LZ4_EARLY_ABORT
      //DRKTODO: Evaluate unrolling further. 2xunrolling had some modest benefits
      if (lz4_do_abort_eval && ((pos) >= LZ4_EARLY_ABORT_EVAL)) {
          ptrdiff_t dstd = dst - dst_begin;
          //到这仍然没有匹配，放弃
          if (dstd == 0) {
              lz4_early_aborts++;
              return;
          }

/*        if (dstd >= pos) { */
/*            return; */
/*        } */
/*        ptrdiff_t cbytes = pos - dstd; */
/*        if ((cbytes * LZ4_EARLY_ABORT_MIN_COMPRESSION_FACTOR) > pos)  { */
/*            return; */
/*        } */
          lz4_do_abort_eval = 0;
      }
#endif
    }
    //到这，整个for循环都没有找到match，直接把整个src拷贝到dst即可
    if (skip_final_literals) { *src_ptr = src; *dst_ptr = dst; return; } // do not emit the fi

    //  Emit a trailing literal that covers the remainder of the source buffer,
    //  if we can do so without exceeding the bounds of the destination buffer.
    size_t src_remaining = src_end + LZ4_GOFAST_SAFETY_MARGIN - src;
    if (src_remaining < 15) {
      *dst++ = (uint8_t)(src_remaining << 4);
      memcpy(dst, src, 16); dst += src_remaining;
    } else {
      *dst++ = 0xf0;
      dst = lz4_store_length(dst, end, (uint32_t)(src_remaining - 15));
      if (dst == 0 || dst + src_remaining >= end) return;
      memcpy(dst, src, src_remaining); dst += src_remaining;
    }
    *dst_ptr = dst;
    *src_ptr = src + src_remaining;
    return;

  EXPAND_FORWARD:

    // Expand match forward 查看匹配是否能向前扩展，扩大匹配长度
    {
      const uint8_t * ref_end = match_end - match_distance;
      while (match_end < src_end)
      {
```

推荐阅读

阿里腾讯头条快手都在用的Clickhouse到底是什么?
阅读 1,365

C++ primer 第十四章-重载操作符和Class-type转换
阅读 128

MySQL技术内幕(InnoDB存储引擎)
阅读 369

20200819 行人重识别算法比赛
阅读 1,407

一道有意思的腾讯算法面试题
阅读 2,231

深入浅出lz4压缩算法

小拉风 关注 赞赏支持

```
136
137     // Expand match backward 查看匹配是否能向后扩展，扩大匹配长度
138     {
139       // match_begin_min = max(src_begin + match_distance,literal)
140       const uint8_t * match_begin_min = src_begin + match_distance;
141       match_begin_min = (match_begin_min < src)?src:match_begin_min;
142       const uint8_t * ref_begin = match_begin - match_distance;
143
144       while (match_begin > match_begin_min && ref_begin[-1] == match_begin[-1] ) { match_begin
145     }
146
147     // Emit match 确定好match的offset和length以后，编码成压缩后的格式
148     dst = lz4_emit_match((uint32_t)(match_begin - src), (uint32_t)(match_end - match_begin), (
149     if (!dst) return;
150
151     // Update state
152     src = match_end;
153
154     // Update return values to include the last fully encoded match
155     //刷新src和dst位置，回到while重新开始for循环
156     *dst_ptr = dst;
157     *src_ptr = src;
158   }
159 }
160
```

安卓内存中压缩的实例

```
1  该例子是一个起址0xffffffc06185f000的4K页，大部分是0和1，由于length或者offset超长，多了一些特殊处理，
2
3  发现两个匹配，压缩后的数据为31bytes，压缩后概览如下
4  09-15 14:35:06.821 <3>[138, kswapd0][  638.194336]  src 0xffffffc06185f000 literallen 1
5  09-15 14:35:06.821 <3>[138, kswapd0][  638.194349]  src 0xffffffc06185f000 (1,219)   #(offset,
6  09-15 14:35:06.821 <3>[138, kswapd0][  638.194359]  src 0xffffffc06185f000 literallen 1
7  09-15 14:35:06.821 <3>[138, kswapd0][  638.194386]  src 0xffffffc06185f000 (3044,7)
8  09-15 14:35:06.821 <3>[138, kswapd0][  638.194400]  src 0xffffffc06185f000 count 2 compressed
9  ----------------------------对应压缩后的原始数据-----------------------------
10 第一个匹配：
11 09-15 14:35:06.821 <3>[138, kswapd0][  638.194411]   0xffffffc06185f000 31    #token:0001 1111
12 09-15 14:35:06.821 <3>[138, kswapd0][  638.194422]   0xffffffc06185f000 0     #literal
13 09-15 14:35:06.821 <3>[138, kswapd0][  638.194433]   0xffffffc06185f000 1     #offset 小端序01
14 09-15 14:35:06.821 <3>[138, kswapd0][  638.194444]   0xffffffc06185f000 0     #offset
15 09-15 14:35:06.821 <3>[138, kswapd0][  638.194459]   0xffffffc06185f000 255   #matchLength beg
16 09-15 14:35:06.821 <3>[138, kswapd0][  638.194469]   0xffffffc06185f000 255
17 09-15 14:35:06.822 <3>[138, kswapd0][  638.194483]   0xffffffc06185f000 255
18 09-15 14:35:06.822 <3>[138, kswapd0][  638.194494]   0xffffffc06185f000 255
19 09-15 14:35:06.822 <3>[138, kswapd0][  638.194505]   0xffffffc06185f000 255
20 09-15 14:35:06.822 <3>[138, kswapd0][  638.194551]   0xffffffc06185f000 255
21 09-15 14:35:06.822 <3>[138, kswapd0][  638.194565]   0xffffffc06185f000 255
22 09-15 14:35:06.822 <3>[138, kswapd0][  638.194579]   0xffffffc06185f000 255
23 09-15 14:35:06.822 <3>[138, kswapd0][  638.194590]   0xffffffc06185f000 255
24 09-15 14:35:06.822 <3>[138, kswapd0][  638.194602]   0xffffffc06185f000 255
25 09-15 14:35:06.822 <3>[138, kswapd0][  638.194612]   0xffffffc06185f000 255
26 09-15 14:35:06.822 <3>[138, kswapd0][  638.194624]   0xffffffc06185f000 219   #matchLength end
27 第二个匹配：
28 09-15 14:35:06.822 <3>[138, kswapd0][  638.194635]   0xffffffc06185f000 31    #Token:0001 1111
29 09-15 14:35:06.822 <3>[138, kswapd0][  638.194646]   0xffffffc06185f000 1     #literal
30 09-15 14:35:06.822 <3>[138, kswapd0][  638.194657]   0xffffffc06185f000 228   #offset
31 09-15 14:35:06.822 <3>[138, kswapd0][  638.194667]   0xffffffc06185f000 11    #offset 228(1110
32 09-15 14:35:06.822 <3>[138, kswapd0][  638.194678]   0xffffffc06185f000 255   #matchLength beg
33 09-15 14:35:06.822 <3>[138, kswapd0][  638.194689]   0xffffffc06185f000 255
34 09-15 14:35:06.822 <3>[138, kswapd0][  638.194701]   0xffffffc06185f000 255
35 09-15 14:35:06.822 <3>[138, kswapd0][  638.194712]   0xffffffc06185f000 255
36 09-15 14:35:06.822 <3>[138, kswapd0][  638.194747]   0xffffffc06185f000 7     #matchLength end
```

解压算法

压缩理解了其实解压也很简单

```
1  输入：[token]abcde_(5,4)[token]fgh_(14,5)fghxxxxxxx
2  输出：abcde_bcdefgh_abcdefghxxxxxxx
```


推荐阅读

阿里腾讯头条快手都在用的Clickhouse到底是什么?
阅读 1,365

C++ primer 第十四章-重载操作符和Class-type转换
阅读 128

MySQL技术内幕(InnoDB存储引擎)
阅读 369

20200819 行人重识别算法比赛
阅读 1,407

一道有意思的腾讯算法面试题
阅读 2,231

深入浅出lz4压缩算法

小拉风 关注 赞赏支持

遇到match，在从前面已经拷贝的literals复制到后面即可

7人点赞 > 日记本

"你的支持是我最大的写文章的动力~"

赞赏支持

共2人赞赏

小拉风
总资产3 (约0.22元) 共写了620字 获得8个赞 共4个粉丝
关注

评论

用户已关闭评论，与Ta简信交流吧

推荐阅读

阿里腾讯头条快手都在用的Clickhouse到底是什么?
阅读 1,365

C++ primer 第十四章-重载操作符和Class-type转换
阅读 128

MySQL技术内幕(InnoDB存储引擎)
阅读 369

20200819 行人重识别算法比赛
阅读 1,407

一道有意思的腾讯算法面试题
阅读 2,231

写下你的评论... 评论0 赞7