Mnesia table fragmentation
过程及算法分析

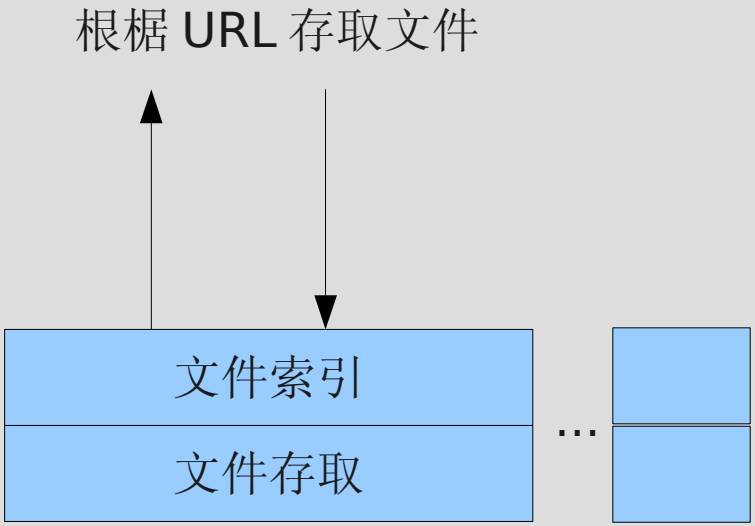
李小红 (lixiaohong@gmail.com)
2008 年 11 月 3 日

目录

- 问题域及解决思路
- Mnesia table frgmentation 解决方案
- Linear hashing 分片过程及算法
- Linear hashing 在 erlang 中的应用

问题域及解决思路

问题：新闻文件存取服务

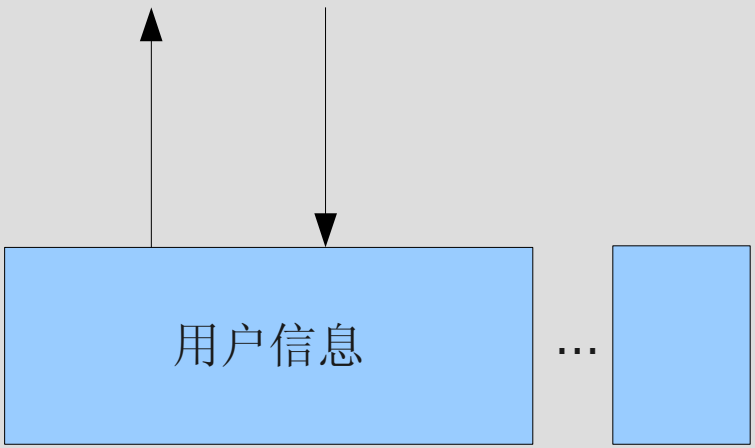


. 每日正常读取 1 亿次
. 不考虑高峰，读取次数在增长
. 读取时间 <300ms
. 存取服务可用性 >99.99%

. 当前 5 千万个文件，包括图片
. 每日增长 10 万个文件

此页数字纯属虚构，纯作解释用

问题：注册用户管理及认证服务

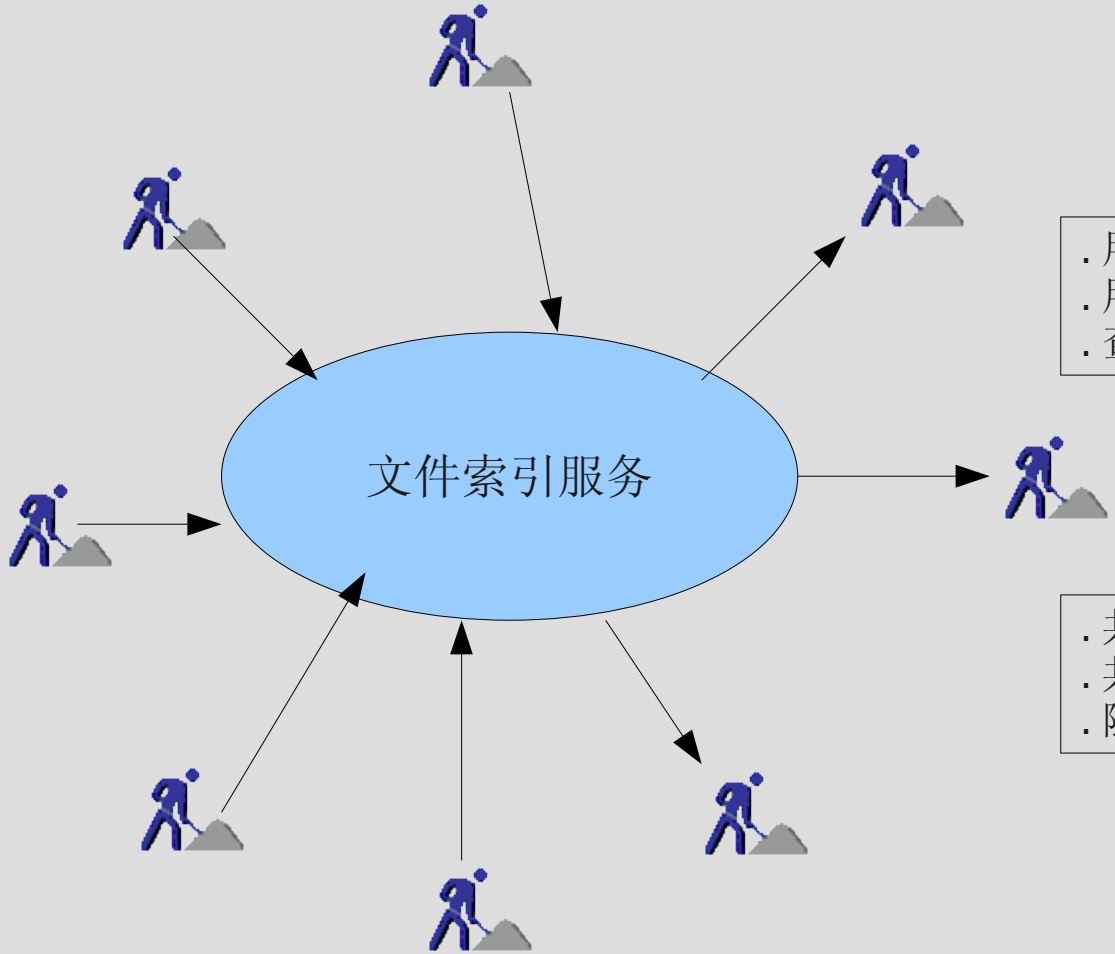


. 每日认证 5 亿次
. 不考虑高峰，认证次数在增长
. 用户相关操作 <300ms
. 用户服务可用性 >99.99%

. 当前 2 亿注册用户
. 每日增长 20 万个新用户
. 每日注销 5 千个用户

此页数字纯属虚构，纯作解释用

问题：p2p 共享文件索引服务



. 用户并发查询文件的次数非常高
. 用户查询文件时间 < 30s
. 查询结果数量方面精度要求不高

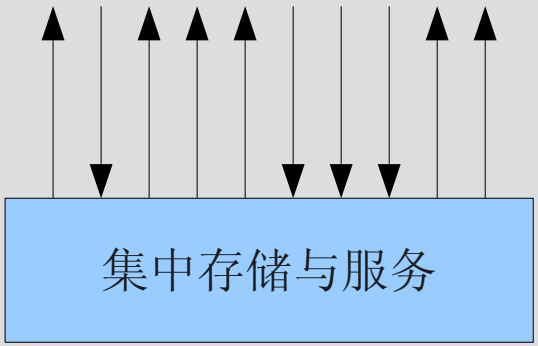
. 共享网络有 2 百万在线用户
. 共享网络中有几十亿个文件
. 随时都有人上线或下线

此页数字纯属虚构，纯作解释用

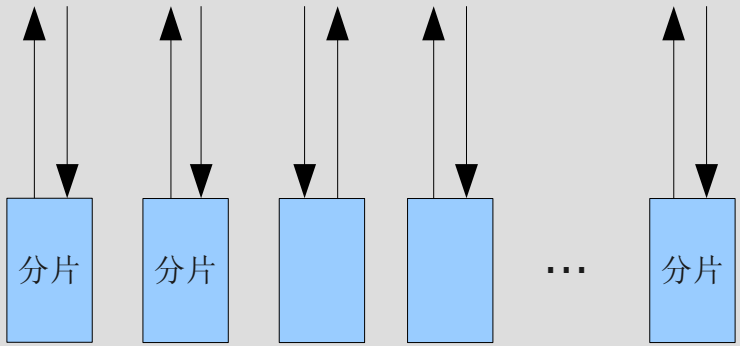
问题特点及解决思路

. 数量大且增长快
. 数据存取量大
. 数据存取性能要求高
. 存取服务的可用性要求高

. 易管理与统计
. 存储容量存在瓶颈
. 性能存在瓶颈，不易扩展
. 保证高可用性的成本高
. 其它问题，如 p2p 易被封



. 管理难度大
. 数据容量方便扩展，并瓶颈
. 数据存取性能无瓶颈
. 可用性高，可用性成本低



数据分片的策略

- 按范围拆表 (range)
  - 1-1000, 1001-2000, 2001-3000
- 按指定的列表拆表 (list)
  - (湖南、四川) , (北京) , (山东、河北)
- 按关键词的 hash 值拆表 (hash)
  - Linear hashing

Mnesia table frgmentation

解决方案

mnesia 解决方案特点

- 使用 linear hashing 算法进行分片，好管理
- 每个分片都可当作正常的表，被复制、移动
- 每个分片的可用性都好保证
- 提供了非常好的查总数据操作接口，对使用者透明

Hash

- 元素 Key 计算出对应的 hash 数值 H -> O(1)
- 将 H mod Buckets 得到 B -> O(1)
- 到 B 对应的 bucket 中线性查找元素 -> O(n)

0 8 16	1 5 9 13 17	2 6 10 14 18	3 7 11 15 19	4 12

Hash 表中总元素 /Bucket 数 (N / B) 越高，
查找的效率越低。(太挤了，呵呵)

保持 hash 性能的策略

- 由最多的元素数计算出 Buckets 数量，程序运行时不再变化
  - 不灵活，一般需要用户干预
  - 浪费空间
  - 不适合放入数量未知或变化范围极大的场合
- 动态调整 buckets 数量
  - 需要控制调整 buckets 时搬迁的元素数量
  - 需要控制计算谁搬迁或重新计算 hash 值的计算成本
  - 需要把握好何时做调整

Linear hashing 特点

- 每次只增加或减少一个分片
- 每次只影响原来的一个分片中住户
- 扩充时受影响分片中有将近一半用户迁到新分片中
- 缩减时一个分片中的用户都迁到另一个分片中
- 在大多数情况下，各分片中住户数量不均衡

建立一个表并插入数据

```
$ erl
Erlang (BEAM) emulator version 5.6.3 [source] [async-threads:0] [kernel-
poll:false]

Eshell V5.6.3  (abort with ^G)
1> mnesia:create_schema([node()]).
ok
2> mnesia:start().
ok
3> rd(test, {key,value}).
test
4> Tab = test.
test
5> mnesia:create_table(Tab, [{attributes,record_info(fields,test)}]).
{atomic,ok}
6>
6> Write = fun(Keys) -> [mnesia:write({Tab,K,K}) || K <- Keys], ok end.
#Fun<erl_eval.6.13229925>
7>
7> mnesia:activity(sync_dirty, Write, [lists:seq(1, 1000)], mnesia_frag).
ok
```

起用拆表

```
8> mnesia:change_table_frag(user, {activate, []}).
{aborted,{no_exists,{user,cstruct}}}
9> mnesia:change_table_frag(Tab, {activate, []}).
{atomic,ok}
10> mnesia:table_info(Tab, frag_properties).
[{base_table,test},
 {foreign_key,undefined},
 {hash_module,mnesia_frag_hash},
 {hash_state,{hash_state,1,1,0,phash2}},
 {n_fragments,1},
 {node_pool,[nonode@nohost]}]
12> Read = fun(Item) -> mnesia:table_info(Tab, Item) end.
#Fun<erl_eval.6.13229925>
13>
13> Dist = mnesia:activity(sync_dirty, Read, [frag_dist], mnesia_frag).
[{nonode@nohost,1}]
14>
14> mnesia:activity(sync_dirty, Read, [frag_size], mnesia_frag).
[{test,1000}]
15>
```

增加到 5 个 frags

15> **mnesia:change_table_frag(Tab, {add_frag, Dist}).**
{atomic,ok}
16> **mnesia:activity(sync_dirty, Read, [frag_size], mnesia_frag).**
[{test,476},{test_frag2,524}]
17> **Dist2 = mnesia:activity(sync_dirty, Read, [frag_dist], mnesia_frag).**
[{nonode@nohost,2}]
18> **mnesia:change_table_frag(Tab, {add_frag, Dist2}).**
{atomic,ok}
19> **mnesia:activity(sync_dirty, Read, [frag_size], mnesia_frag).**
[{test,230},{test_frag2,524},{test_frag3,246}]
20> **Dist3 = mnesia:activity(sync_dirty, Read, [frag_dist], mnesia_frag).**
[{nonode@nohost,3}]
21> **mnesia:change_table_frag(Tab, {add_frag, Dist3}).**
{atomic,ok}
22> **Dist3 = mnesia:activity(sync_dirty, Read, [frag_dist], mnesia_frag).**
** exception error: no match of right hand side value [{nonode@nohost,4}]
23> **mnesia:activity(sync_dirty, Read, [frag_size], mnesia_frag).**
[{test,230}, {test_frag2,233}, {test_frag3,246}, {test_frag4,291}]
24> **Dist4 = mnesia:activity(sync_dirty, Read, [frag_dist], mnesia_frag).**
[{nonode@nohost,4}]
25> **mnesia:change_table_frag(Tab, {add_frag, Dist4}).**
{atomic,ok}
26> **mnesia:activity(sync_dirty, Read, [frag_size], mnesia_frag).**
[{test,121}, {test_frag2,233}, {test_frag3,246}, {test_frag4,291}, {test_frag5,109}]

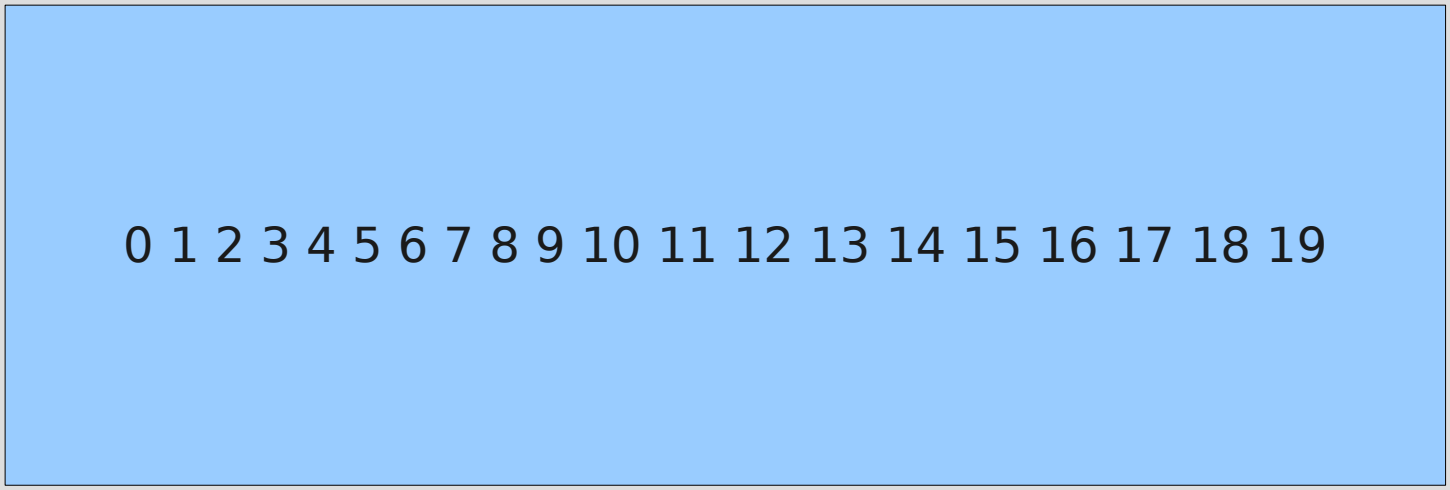
发现了什么规律？

```
1 [{test,1000}]
2 [{test,476},{test_frag2,524}]
3 [{test,230},{test_frag2,524},{test_frag3,246}]
4 [{test,230},{test_frag2,233},{test_frag3,246},{test_frag4,291}]
5 [{test,121},{test_frag2,233},{test_frag3,246},{test_frag4,291},{test_frag5,109}]
```

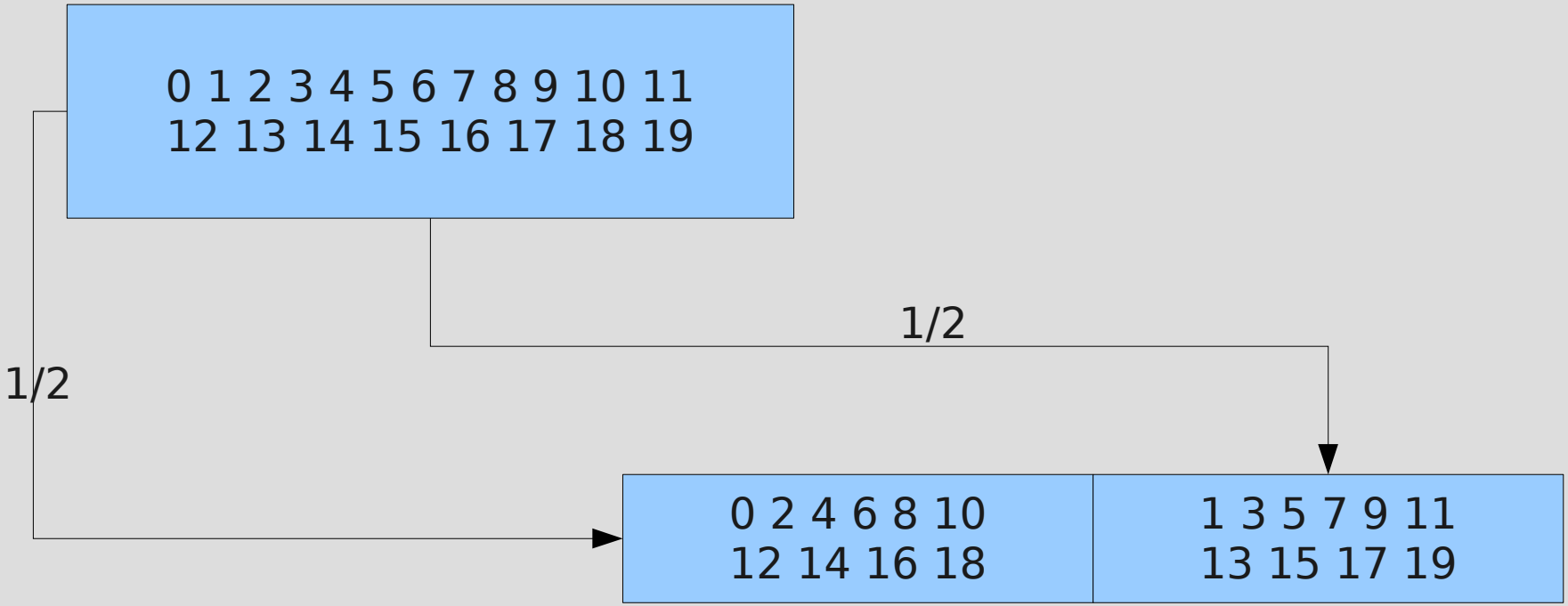
Linear hashing 分片过程及算法

初始状态（无 frags）

有 20 个记录，各记录以它们的 hash 值为标记

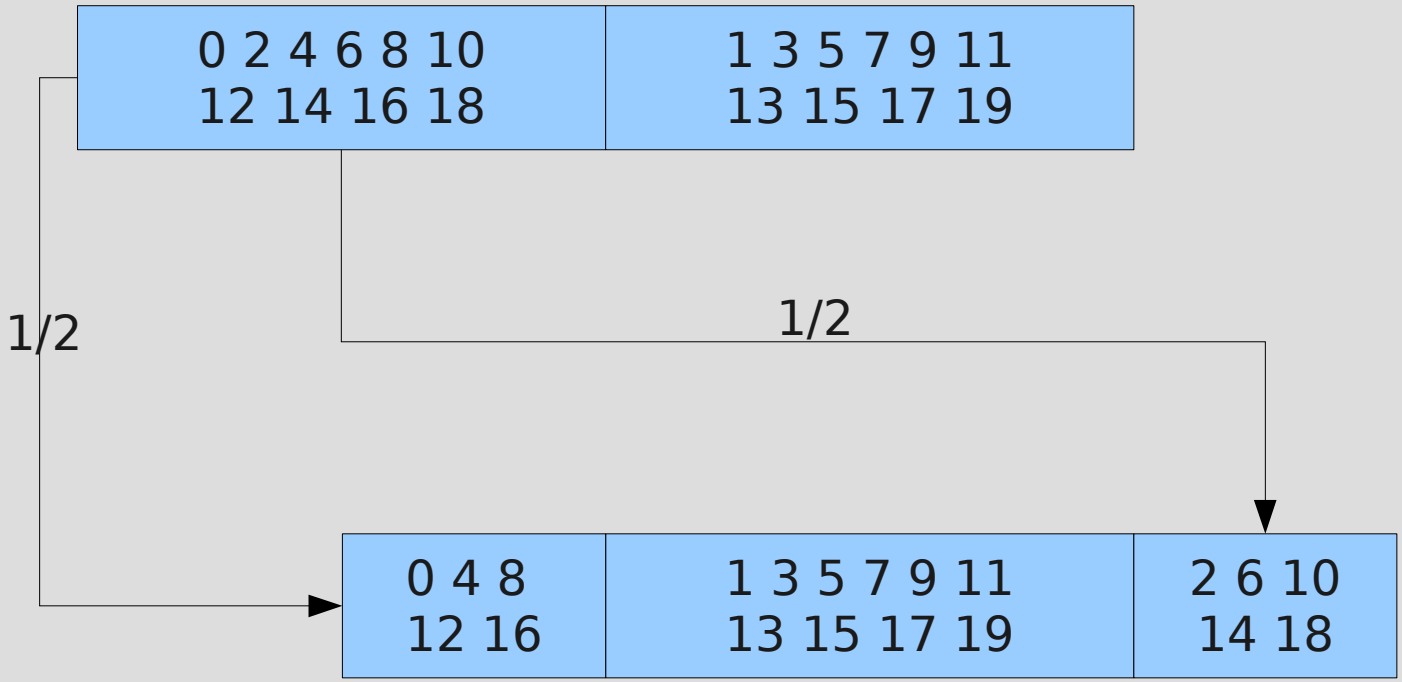


由 1 个 frag -> 2 个 frags



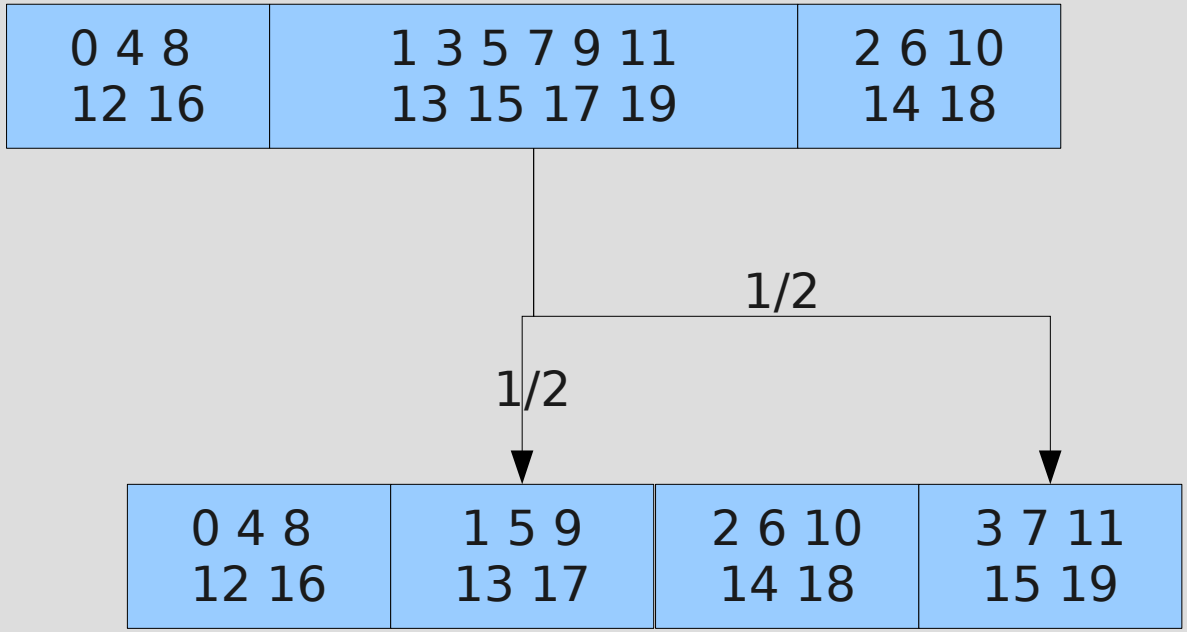
K mod 2 = 0 留在 frag0
K mod 2 = 1 移到 frag1

从 2 个 frags -> 3 个 frags



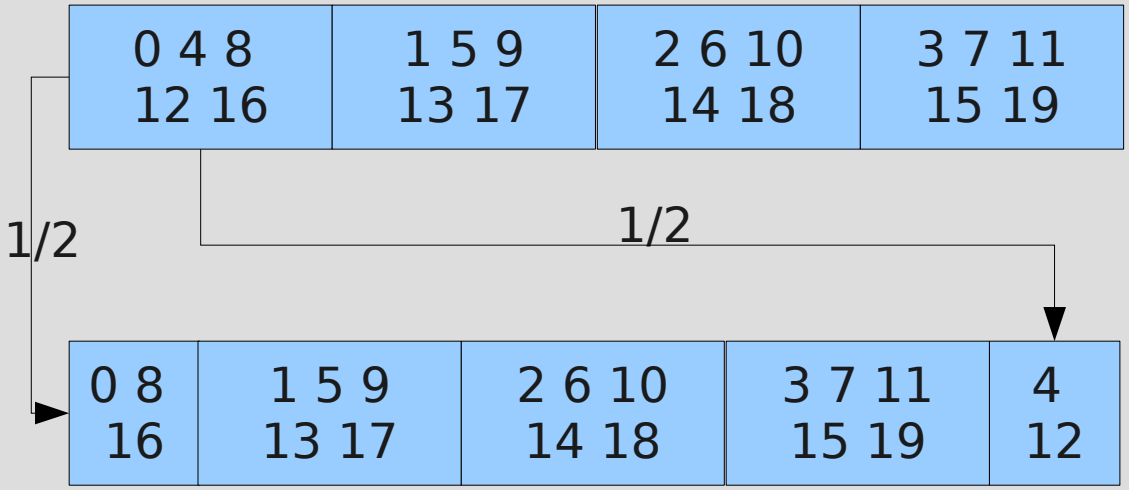
K mod 4 = 0 留在 frag0
K mod 4 = 2 移到 frag2
而这些元素 K mod 2 都是 0

从 3 个 frags -> 4 个 frags



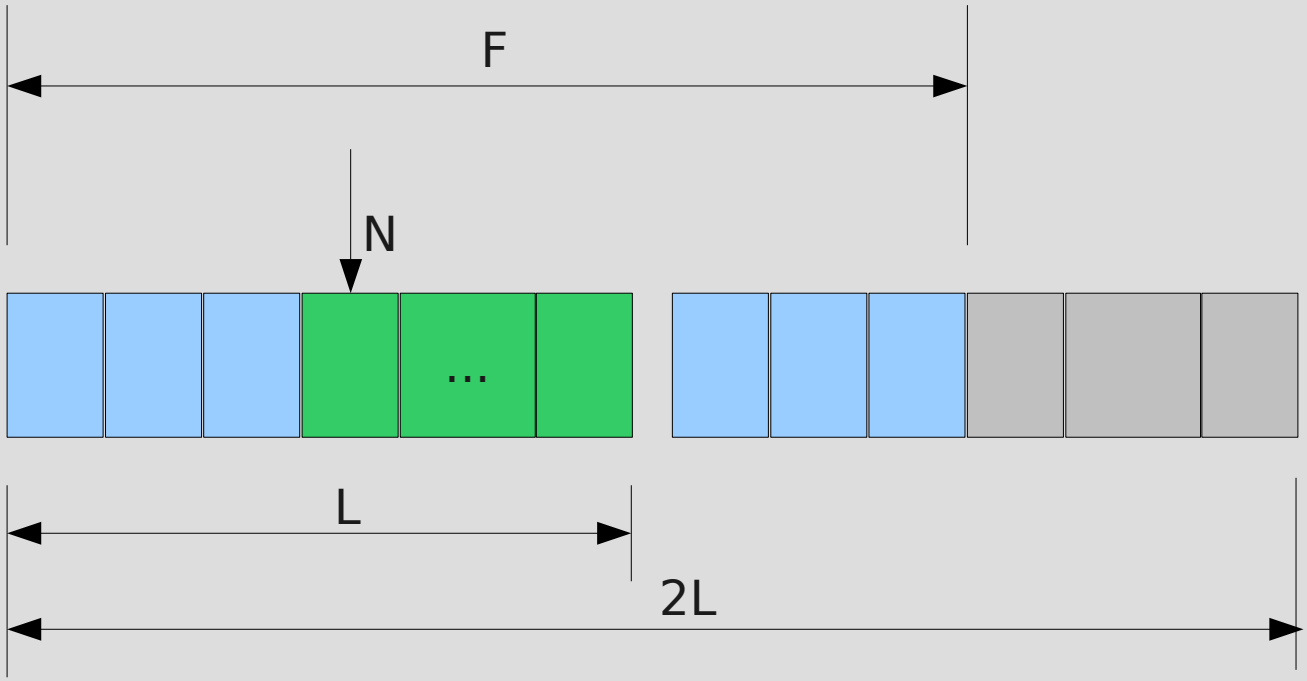
K mod 4 = 1 留在 frag1
K mod 4 = 3 移到 frag3
而这些元素 K mod 2 都是 1

从 4 个 frags -> 5 个 frags



K mod 8 = 0 留在 frag0
K mod 8 = 4 移到 frag4
而这些元素 K mod 4 都是 0

Linear Hashing 算法



.F： 当前的分片数
.N： 下一个拆分的片
.L： 当前取模的值

Linear Hashing 状态变量

```
-record(hash_state,
    {n_fragments,
     next_n_to_split,
     n_doubles,
     function}).
```

下一个拆分的 bucket

当前 frags 数

Hash 表增长的倍数

初始值：

n_framents = 1
next_n_to_split = 1
n_doubles = 0

定位某个 key 对应的 bucket

```
key_to_frag_number(#hash_state{function = phash2,
next_n_to_split = SplitN, n_doubles = L}, Key) ->
    P = SplitN,
    A = erlang:phash2(Key, power2(L)) + 1,
    if
   A < P ->
       erlang:phash2(Key, power2(L + 1)) + 1;
   true ->
       A
    end;
```

```
bucket = hash(key) mod (2 ** n_doubles) + 1
if bucket < next_n_to_split then
   bucket = hash(key) mod ( 2 ** (n_doubles+1) ) + 1
end
```

扩充一个 bucket

```
add_frag(#hash_state{next_n_to_split = SplitN, n_doubles =
L, n_fragments = N} = State) ->
    P = SplitN + 1,
    NewN = N + 1,
    State2 = case power2(L) + 1 of
        P2 when P2 == P ->
            State#hash_state{n_fragments      = NewN,
                    n_doubles        = L + 1,
                    next_n_to_split = 1};
        _ ->
            State#hash_state{n_fragments     = NewN,
                    next_n_to_split = P}
         end,
    {State2, [SplitN], [NewN]};
```

```
n_fragments = n_fragments + 1
next_n_to_split = next_n_to_split + 1
if next_n_to_split = (2**n_doubles + 1) then
     n_doubles = n_doubles + 1
     next_n_to_split = 1
end
```

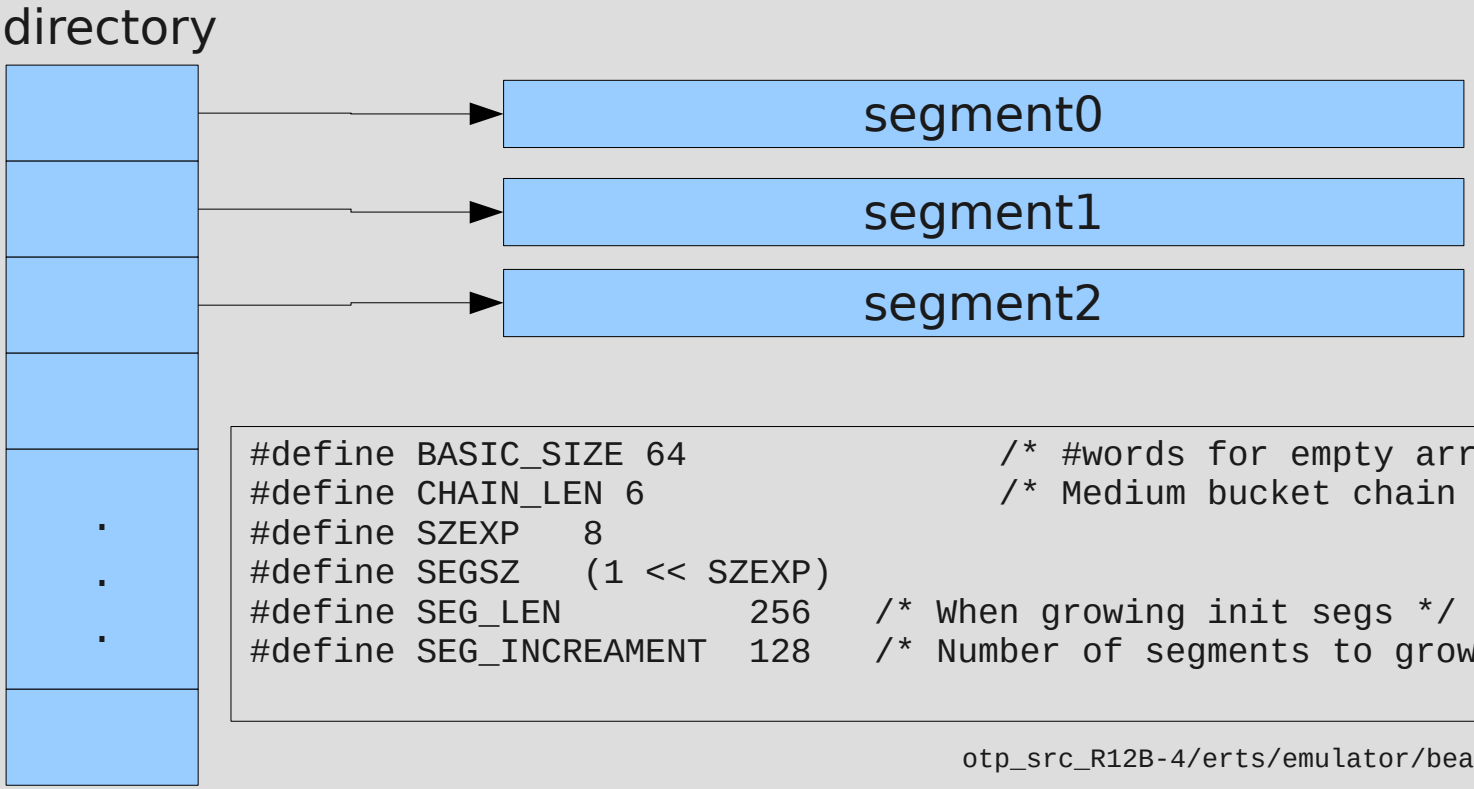
减少一个 bucket

```
del_frag(#hash_state{next_n_to_split = SplitN, n_doubles = L,
n_fragments = N} = State) ->
    P = SplitN - 1,
    if
    P < 1 ->
        L2 = L - 1,
        MergeN = power2(L2),
        State2 = State#hash_state{n_fragments     = N - 1,
                     next_n_to_split = MergeN,
                     n_doubles       = L2},
        {State2, [N], [MergeN]};
    true ->
        MergeN = P,
        State2 = State#hash_state{n_fragments     = N - 1,
                     next_n_to_split = MergeN},
        {State2, [N], [MergeN]}
    end;
```

```
n_fragments = n_fragments - 1
next_n_to_split = next_n_to_split - 1
if next_n_to_split < 1 then
     n_doubles = n_doubles - 1
     next_n_to_split = 2**n_doubles
end
```

Linear hashing 在 erlang 中的应用

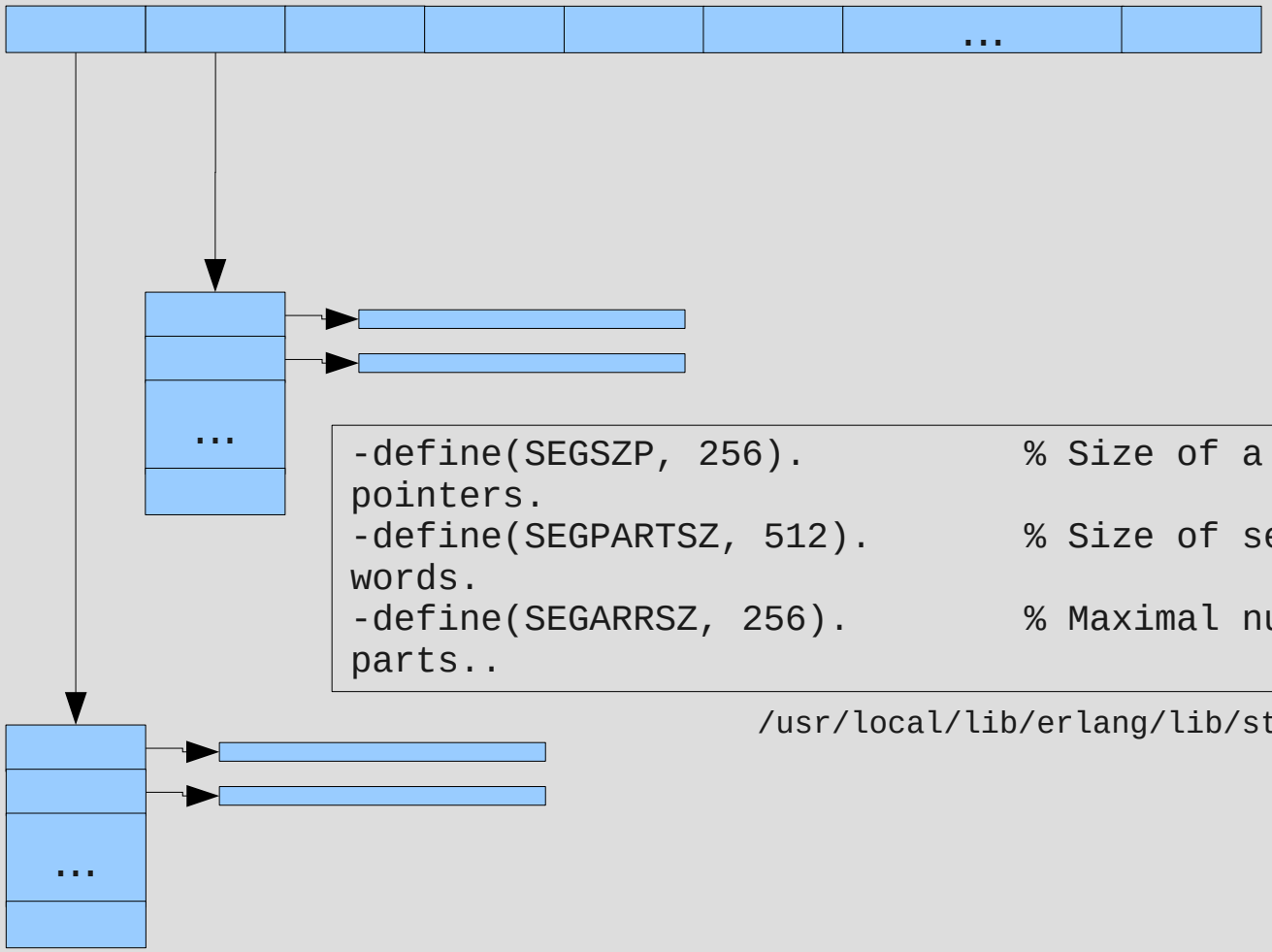
ets




```
#define BASIC_SIZE 64                  /* #words for empty array       */
#define CHAIN_LEN 6                    /* Medium bucket chain len      */
#define SZEXP   8
#define SEGSZ   (1 << SZEXP)
#define SEG_LEN         256   /* When growing init segs */
#define SEG_INCREAMENT  128   /* Number of segments to grow */
```

otp_src_R12B-4/erts/emulator/beam/erl_db_hash.c

dets




```
-define(SEGSZP, 256).          % Size of a segment, in number of
pointers.
-define(SEGPARTSZ, 512).       % Size of segment array part, in
words.
-define(SEGARRSZ, 256).        % Maximal number of segment array
parts..
```

/usr/local/lib/erlang/lib/stdlib-1.15.3/src/dets_v9.erl

其它的动态扩展 **hash**

- expandiable hashing
- consistent hashing

Q&A

谢谢！