Adaptive Readahead

State of the Art

Fengguang Wu

2006.9.22

the Stock Readahead

- **simple**
  - fast
  - understandable
- **dumb**
  - cache hit
  - thrashing
  - memory consumption
  - multiple streams

Why a Complex Replacement?

- **Yeah, 1500 LOC**
- **Not a problem, when it keeps**
  - simplicity of concept
  - efficiency of execution

Why a Complex Replacement?

- **Even good, when it brings**
  - robustness
  - new features
  - memory efficiency
  - higher I/O capability
  - a bunch of statistics

Call Scheme

- **stock**
  - on `read()` invocation
  - on look-ahead index
- **adaptive**
  - on page fault
  - on look-ahead mark

Call Scheme

- **benefits**
  - avoids cache hit problem
  - `fadvise()` harmony
  - work with semi-sequential I/O

Readahead Size

- **stock**
  - up: `*4` then `*2`
  - top: `readahead_max`
- **adaptive**
  - up: `*2 + readahead_max/16`
  - top: `threshing_threshold`

Key Components

- **stateful method**
  - the fast and default one
  - bails out on abnormal cases
- **stateless method**
  - the robust and failsafe one
  - queries the page cache

Stateful Algorithm

```
    chunk A                    chunk B
                        |<=============== global_shift ================|
+-------------+         +-------------------+                          |
|       #     |         |           #       |            inactive_list |
+-------------+         +-------------------+                     head |
        |---->|         |---------->|
           |                  |
           +-- stream_shift --+
```

While the stream reads stream_shift pages inside the chunks,
the chunks are shifted global_shift pages inside inactive_list.

**thrashing_threshold = free_mem *
stream_shift / global_shift**

thrashing_threshold *= readahead_ratio/100

Stateful Benefits

- **thrashing safe**
- **less memory consumption**
- **thousands of streams**
- **non-uniform streams**

Stateless Algorithm

1. **count history pages => `H`**
2. **readahead `R` pages**

- **simplified relation**

```
R = H
```

- **in practical**

```
R = min(H * readahead_ratio/100,
        readahead_max)
```

Stateless Benefits

- **semi-sequential I/O**
  - parallel/interleaved sequential scans on one file descriptor
  - sequential reads across file open/close lifetime
  - mixed sequential/random accesses
  - sparse/skimming sequential reads

Stateless Benefits

- **simplifies stateful method**
  - restart readahead after abnormal cases (i.e. after cache hit)
- **thrashing safe**

Stateless Overheads (tiny ones)

- **page cache lookup**
  - lock contention
    - solved by Nick Piggin's lockless patch
  - L1/2 cache miss
    - small scan: already cache warm
    - large scan: not a big problem for array;
      even better for radix-tree
- **page flag check**
  - 2 checks in normal

Stateless Overheads (major ones)

- **readahead miss**
  - on random I/O
  - can be a loss for sparse I/Os
- **NFS contention**
  - duplicate readaheads
    - triggered by concurrent, nearby reads
  - can occur when
    - `rsize <= 32K`
    - `near start of file`

the Duties

- **apps**
  - fadvise for **random** I/O
- **kernel**
  - detect **sequential** I/O
  - readahead at the right time and size

the Duties

- **fadvise for sequential I/O: not a good thing**
- **Why kernel? Information and resource management!**
  - memory availability
  - streams in the system
  - data layout
  - disk utilization

Default Choice?

- **stateful method: y/N**
  - candidate as stock readahead replacement in the long run
- **stateless method: y/M/n**
  - obvious benefits in some cases
  - obvious overheads for others

Work...

- **context method runtime selectable**
  - as module, or as tunable parameter?
- **statistics infrastructure**
  - almost complete
- **remove initial_readahead intelligence**
  - turn to a tunable

Work...

- **improve NFS performance**
  - untested idea
- **improve small files handling**
  - untested idea
- **docs**
- **benchmarks**

Benchmarks...

- **pure random I/O**
  - overheads on lockless pagecache
  - readahead miss on different sparseness
- **NFS**
  - rsize / file size combinations
  - performance tuning
- **small files**

Thank you.

Readahead Chunk

```
             a read-ahead chunk
+-----------------------------------------+
|       # PG_readahead                    |
+-----------------------------------------+
        ^ When this page is read, notify me
for the next read-ahead.
```

Readahead Chunks

```
   chunk A           chunk B                      chunk C                 head

   l01 l11           l12   l21                    l22
| |-->|-->|       |------>|-->|                |------>|
| +-------+       +-----------+                +-------------+               |
| |   #   |       |       #   |                |       #     |               |
| +-------+       +-----------+                +-------------+               |
| |<==============|<===========================|<============================|
        L0                     L1                            L2
```

```
Let f(l) = L be a map from
     l: the number of pages read by the stream
to
     L: the number of pages pushed into inactive_list in the mean time
then
     f(l01) <= L0

     f(l11 + l12) = L1
     f(l21 + l22) = L2
     ...
     f(l01 + l11 + ...) <= Sum(L0 + L1 + ...)
                        <= Length(inactive_list) = f(thrashing-threshold)

So the count of countinuous history pages left in the inactive_list is always
a lower estimation of the true thrashing-threshold.
```