Lustre transition to folio i/o

Why struct folio?

Shaun Tancheff <shaun.tancheff@hpe.com>

Lustre transition to folio i/o

Why struct folio?

Upstream will not accept a file system using struct page now that struct folio is the standard.

Shaun Tancheff <shaun.tancheff@hpe.com>

Lustre transition to folio i/o

Why struct folio?

Upstream will not accept a file system using struct page now that struct folio is the standard.

Kernel support: many APIs have switched to struct folio and using the older API getting more difficult as the in-kernel users are being converted.

Shaun Tancheff <shaun.tancheff@hpe.com>

Moving to folio order 0: DirectIO

DirectIO user pages when treated as folios need to retain offset

The main caveat being DIO where get_user_page* family hands a struct page treating that page as a zero-order folio does not work, because it isn't.

This needs tracking through osc_async_page, osc_page (cl_page), brw, bulk RPC

LU-17916

Moving to folio: Buffered I/O

Buffered I/O

The main challenge of struct page -> struct folio at order 0 is not particularly complex in that the zero order page and zero order folio can mostly be handled as interchangeable.

LU-17916

Moving to folio order 0: status

Aside from DIO the primary friction is just the amount of Lustre code that needs to be touched ... which is large enough that reviewing takes time and it frequently conflicts with other work in progress.

So tests are passing for page->folio and reviews are getting completed so this looks to be ready for 2.18.

LU-17916

Large folios (order > 0), like DIO

Naive method follow DIO creating one cl_page for each PAGE_SIZE of I/O mapping many cl_pages onto one folio.

This conflicts with the cache and makes pages within a folio in-consistent leading to lots of special case handling with cl_page.

In testing the performance gains just did not appear, likely due to excessive overhead taken up in by cl_page, et. al.

LU-19895

Large folios: large cl_page

The current approach is to push the large folio down thorough to the BRW and osc_extent. The primary advantage here is the cl_page logic remains much the same.

Not all I/O starts at a folio boundary, since cl_page is indexed on the folio index some requests for cache pages need to be adjusted and cl_page must pass down the offset and length through to the extent/brw layer.

LU-19895

Large folios: caveats

mmap() page faults are arrive in 'page size' chunks

places where PAGE_SIZE is assumed needs to be adjust for large folios

osc_extent grows in multiple pages chunks which affects grant handling

cl_page_list, osc_extents, brw_page (now brw_ext) all now track number of pages along with #of entries.

LU-19895

Large folios: brw, encryption

the brw copy-in, copy-out, t10 and encryption needs to walk a nested loop over the pages within the folio

encryption bounce folios are always backed by order 0 folios (pool folios) so a multi-page folio is backed by an array of order 0 folios, currently the array to hold the bounce is a slice of the same array holding the allocated bounce folios, rather than allocating a tertiary array.

LU-19895

Large folios: readahead, pcc

rename brw_page -> brw_ext since this is not mapping back to a single page

read ahead adjustments for entire folios

mmap() fault is page at a time however a large folio should fault in the entire folio

pcc backing file-system should align folio order with Lustre

LU-19895

Large folios: offset, length, flags

cl_page offset (pages from index) and length (number of pages used) needed for short writes.

oap_page_off / ops_from / ops_to can be larger than 1 page

for EC bp_count_diff and bp_off_diff are 16 bits should they be 32 bits?

brw adds flags for 'enc' and 'cache'

LU-19895

Thank You

Questions?