```
***********************************************************
   44850 Tue Aug 27 12:59:20 2013
new/usr/src/uts/common/fs/zfs/dmu.c
4082 zfs receive gets EFBIG from dmu_tx_hold_free()
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Christopher Siden <christopher.siden@delphix.com>
Reviewed by: George Wilson <george.wilson@delphix.com>
***********************************************************
______unchanged_portion_omitted_

 665 int
 666 dmu_free_long_range(objset_t *os, uint64_t object,
 667     uint64_t offset, uint64_t length)
 668 {
 669         dnode_t *dn;
 670         int err;

 672         err = dnode_hold(os, object, FTAG, &dn);
 673         if (err != 0)
 674                 return (err);
 675         err = dmu_free_long_range_impl(os, dn, offset, length);

 677         /*
 678          * It is important to zero out the maxblkid when freeing the entire
 679          * file, so that (a) subsequent calls to dmu_free_long_range_impl()
 680          * will take the fast path, and (b) dnode_reallocate() can verify
 681          * that the entire file has been freed.
 682          */
 683         if (offset == 0 && length == DMU_OBJECT_END)
 684                 dn->dn_maxblkid = 0;

 686         dnode_rele(dn, FTAG);
 687         return (err);
 688 }
______unchanged_portion_omitted_
```

```
**********************************************************
   44556 Tue Aug 27 12:59:26 2013
new/usr/src/uts/common/fs/zfs/dmu_tx.c
4082 zfs receive gets EFBIG from dmu_tx_hold_free()
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Christopher Siden <christopher.siden@delphix.com>
Reviewed by: George Wilson <george.wilson@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 586 void
 587 dmu_tx_hold_free(dmu_tx_t *tx, uint64_t object, uint64_t off, uint64_t len)
 588 {
 589         dmu_tx_hold_t *txh;
 590         dnode_t *dn;
 591         int err;
 592         zio_t *zio;

 594         ASSERT(tx->tx_txg == 0);

 596         txh = dmu_tx_hold_object_impl(tx, tx->tx_objset,
 597             object, THT_FREE, off, len);
 598         if (txh == NULL)
 599                 return;
 600         dn = txh->txh_dnode;
 601         dmu_tx_count_dnode(txh);

 603         if (off >= (dn->dn_maxblkid+1) * dn->dn_datablksz)
 604                 return;
 605         if (len == DMU_OBJECT_END)
 606                 len = (dn->dn_maxblkid+1) * dn->dn_datablksz - off;


 608         /*
 609          * For i/o error checking, we read the first and last level-0
 610          * blocks if they are not aligned, and all the level-1 blocks.
 611          *
 612          * Note:  dbuf_free_range() assumes that we have not instantiated
 613          * any level-0 dbufs that will be completely freed.  Therefore we must
 614          * exercise care to not read or count the first and last blocks
 615          * if they are blocksize-aligned.
 616          */
 617         if (dn->dn_datablkshift == 0) {
 618                 if (off != 0 || len < dn->dn_datablksz)
 619                         dmu_tx_count_write(txh, 0, dn->dn_datablksz);
 620                         dmu_tx_count_write(txh, off, len);
 620         } else {
 621                 /* first block will be modified if it is not aligned */
 622                 if (!IS_P2ALIGNED(off, 1 << dn->dn_datablkshift))
 623                         dmu_tx_count_write(txh, off, 1);
 624                 /* last block will be modified if it is not aligned */
 625                 if (!IS_P2ALIGNED(off + len, 1 << dn->dn_datablkshift))
 626                         dmu_tx_count_write(txh, off+len, 1);
 627         }

 629         /*
 630          * Check level-1 blocks.
 631          */
 632         if (dn->dn_nlevels > 1) {
 633                 int shift = dn->dn_datablkshift + dn->dn_indblkshift -
 634                     SPA_BLKPTRSHIFT;
 635                 uint64_t start = off >> shift;
 636                 uint64_t end = (off + len) >> shift;

 638                 ASSERT(dn->dn_datablkshift != 0);
 639                 ASSERT(dn->dn_indblkshift != 0);
```



```
 641                 zio = zio_root(tx->tx_pool->dp_spa,
 642                     NULL, NULL, ZIO_FLAG_CANFAIL);
 643                 for (uint64_t i = start; i <= end; i++) {
 644                         uint64_t ibyte = i << shift;
 645                         err = dnode_next_offset(dn, 0, &ibyte, 2, 1, 0);
 646                         i = ibyte >> shift;
 647                         if (err == ESRCH)
 648                                 break;
 649                         if (err) {
 650                                 tx->tx_err = err;
 651                                 return;
 652                         }

 654                         err = dmu_tx_check_ioerr(zio, dn, 1, i);
 655                         if (err) {
 656                                 tx->tx_err = err;
 657                                 return;
 658                         }
 659                 }
 660                 err = zio_wait(zio);
 661                 if (err) {
 662                         tx->tx_err = err;
 663                         return;
 664                 }
 665         }

 667         dmu_tx_count_free(txh, off, len);
 668 }
______unchanged_portion_omitted_
```