```
**********************************************************
   74754 Thu Jan 10 23:20:01 2013
new/usr/src/uts/common/fs/zfs/dbuf.c
3469 dbuf_read_impl shows too much enthusiasm
Reviewed by: Bryan Cantrill <bryan@joyent.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 */

#include <sys/zfs_context.h>
#include <sys/dmu.h>
#include <sys/dmu_impl.h>
#include <sys/dbuf.h>
#include <sys/dmu_objset.h>
#include <sys/dsl_dataset.h>
#include <sys/dsl_dir.h>
#include <sys/dmu_tx.h>
#include <sys/spa.h>
#include <sys/zio.h>
#include <sys/dmu_zfetch.h>
#include <sys/sa.h>
#include <sys/sa_impl.h>

static void dbuf_destroy(dmu_buf_impl_t *db);
static int dbuf_undirty(dmu_buf_impl_t *db, dmu_tx_t *tx);
static void dbuf_write(dbuf_dirty_record_t *dr, arc_buf_t *data, dmu_tx_t *tx);

/*
 * Global data structures and functions for the dbuf cache.
 */
static kmem_cache_t *dbuf_cache;

/* ARGSUSED */
static int
dbuf_cons(void *vdb, void *unused, int kmflag)
{
        dmu_buf_impl_t *db = vdb;
        bzero(db, sizeof (dmu_buf_impl_t));

        mutex_init(&db->db_mtx, NULL, MUTEX_DEFAULT, NULL);
        cv_init(&db->db_changed, NULL, CV_DEFAULT, NULL);
        refcount_create(&db->db_holds);
```



```
        return (0);
}
_____unchanged_portion_omitted_

static void
dbuf_read_impl(dmu_buf_impl_t *db, zio_t *zio, uint32_t *flags)
{
        dnode_t *dn;
        spa_t *spa;
        zbookmark_t zb;
        uint32_t aflags = ARC_NOWAIT;
        arc_buf_t *pbuf;

        DB_DNODE_ENTER(db);
        dn = DB_DNODE(db);
        ASSERT(!refcount_is_zero(&db->db_holds));
        /* We need the struct_rwlock to prevent db_blkptr from changing. */
        ASSERT(RW_LOCK_HELD(&dn->dn_struct_rwlock));
        ASSERT(MUTEX_HELD(&db->db_mtx));
        ASSERT(db->db_state == DB_UNCACHED);
        ASSERT(db->db_buf == NULL);

        if (db->db_blkid == DMU_BONUS_BLKID) {
                int bonuslen = MIN(dn->dn_bonuslen, dn->dn_phys->dn_bonuslen);

                ASSERT3U(bonuslen, <=, db->db.db_size);
                db->db.db_data = zio_buf_alloc(DN_MAX_BONUSLEN);
                arc_space_consume(DN_MAX_BONUSLEN, ARC_SPACE_OTHER);
                if (bonuslen < DN_MAX_BONUSLEN)
                        bzero(db->db.db_data, DN_MAX_BONUSLEN);

                if (bonuslen) {
                        /*
                         * Absent byzantine on-disk corruption, we fully expect
                         * our bonuslen to be no more than DN_MAX_BONUSLEN --
                         * but we nonetheless explicitly clamp it on the bcopy()
                         * to prevent any on-disk corruption from becoming
                         * rampant in-kernel corruption.
                         */
                        bcopy(DN_BONUS(dn->dn_phys), db->db.db_data,
                            MIN(bonuslen, DN_MAX_BONUSLEN));
                }

                if (bonuslen)
                        bcopy(DN_BONUS(dn->dn_phys), db->db.db_data, bonuslen);
                DB_DNODE_EXIT(db);
                dbuf_update_data(db);
                db->db_state = DB_CACHED;
                mutex_exit(&db->db_mtx);
                return;
        }

        /*
         * Recheck BP_IS_HOLE() after dnode_block_freed() in case dnode_sync()
         * processes the delete record and clears the bp while we are waiting
         * for the dn_mtx (resulting in a "no" from block_freed).
         */
        if (db->db_blkptr == NULL || BP_IS_HOLE(db->db_blkptr) ||
            (db->db_level == 0 && (dnode_block_freed(dn, db->db_blkid) ||
            BP_IS_HOLE(db->db_blkptr)))) {
                arc_buf_contents_t type = DBUF_GET_BUFC_TYPE(db);

                dbuf_set_data(db, arc_buf_alloc(dn->dn_objset->os_spa,
                    db->db.db_size, db, type));
                DB_DNODE_EXIT(db);
                bzero(db->db.db_data, db->db.db_size);
```

```
570                 db->db_state = DB_CACHED;
571                 *flags |= DB_RF_CACHED;
572                 mutex_exit(&db->db_mtx);
573                 return;
574         }

576         spa = dn->dn_objset->os_spa;
577         DB_DNODE_EXIT(db);

579         db->db_state = DB_READ;
580         mutex_exit(&db->db_mtx);

582         if (DBUF_IS_L2CACHEABLE(db))
583                 aflags |= ARC_L2CACHE;

585         SET_BOOKMARK(&zb, db->db_objset->os_dsl_dataset ?
586             db->db_objset->os_dsl_dataset->ds_object : DMU_META_OBJSET,
587             db->db.db_object, db->db_level, db->db_blkid);

589         dbuf_add_ref(db, NULL);
590         /* ZIO_FLAG_CANFAIL callers have to check the parent zio's error */

592         if (db->db_parent)
593                 pbuf = db->db_parent->db_buf;
594         else
595                 pbuf = db->db_objset->os_phys_buf;

597         (void) dsl_read(zio, spa, db->db_blkptr, pbuf,
598             dbuf_read_done, db, ZIO_PRIORITY_SYNC_READ,
599             (*flags & DB_RF_CANFAIL) ? ZIO_FLAG_CANFAIL : ZIO_FLAG_MUSTSUCCEED,
600             &aflags, &zb);
601         if (aflags & ARC_CACHED)
602                 *flags |= DB_RF_CACHED;
603 }
```

*unchanged_portion_omitted_*