```
**********************************************************
   77297 Tue Sep  3 20:26:48 2013
new/usr/src/cmd/mdb/common/modules/zfs/zfs.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
______unchanged_portion_omitted_

typedef struct mdb_space_map_phys_t {
        uint64_t smp_alloc;
} mdb_space_map_phys_t;

typedef struct mdb_space_map {
        uint64_t sm_size;
        uint64_t sm_alloc;
        uintptr_t sm_phys;
} mdb_space_map_t;

typedef struct mdb_range_tree {
        uint64_t rt_space;
} mdb_range_tree_t;

typedef struct mdb_metaslab {
        uintptr_t ms_alloctree[TXG_SIZE];
        uintptr_t ms_freetree[TXG_SIZE];
        uintptr_t ms_tree;
        uintptr_t ms_sm;
        space_map_t ms_allocmap[TXG_SIZE];
        space_map_t ms_freemap[TXG_SIZE];
        space_map_t ms_map;
        space_map_obj_t ms_smo;
        space_map_obj_t ms_smo_syncing;
} mdb_metaslab_t;

typedef struct space_data {
        uint64_t ms_alloctree[TXG_SIZE];
        uint64_t ms_freetree[TXG_SIZE];
        uint64_t ms_tree;
        uint64_t ms_allocmap[TXG_SIZE];
        uint64_t ms_freemap[TXG_SIZE];
        uint64_t ms_map;
        uint64_t avail;
        uint64_t nowavail;
} space_data_t;

/* ARGSUSED */
static int
space_cb(uintptr_t addr, const void *unknown, void *arg)
{
        space_data_t *sd = arg;
        mdb_metaslab_t ms;
        mdb_range_tree_t rt;
        mdb_space_map_t sm;
        mdb_space_map_phys_t smp = { 0 };
        int i;

        if (mdb_ctf_vread(&ms, "metaslab_t", "mdb_metaslab_t",
            addr, 0) == -1)
        if (GETMEMB(addr, "metaslab", ms_allocmap, ms.ms_allocmap) ||
```



```
            GETMEMB(addr, "metaslab", ms_freemap, ms.ms_freemap) ||
            GETMEMB(addr, "metaslab", ms_map, ms.ms_map) ||
            GETMEMB(addr, "metaslab", ms_smo, ms.ms_smo) ||
            GETMEMB(addr, "metaslab", ms_smo_syncing, ms.ms_smo_syncing)) {
                return (WALK_ERR);

        for (i = 0; i < TXG_SIZE; i++) {

                if (mdb_ctf_vread(&rt, "range_tree_t",
                    "mdb_range_tree_t", ms.ms_alloctree[i], 0) == -1)
                sd->ms_alloctree[i] += rt.rt_space;

                if (mdb_ctf_vread(&rt, "range_tree_t",
                    "mdb_range_tree_t", ms.ms_freetree[i], 0) == -1)
                sd->ms_freetree[i] += rt.rt_space;
        }

        if (mdb_ctf_vread(&rt, "range_tree_t",
            "mdb_range_tree_t", ms.ms_tree, 0) == -1 ||
            mdb_ctf_vread(&sm, "space_map_t",
            "mdb_space_map_t", ms.ms_sm, 0) == -1)
                return (WALK_ERR);
        sd->ms_allocmap[0] += ms.ms_allocmap[0].sm_space;
        sd->ms_allocmap[1] += ms.ms_allocmap[1].sm_space;
        sd->ms_allocmap[2] += ms.ms_allocmap[2].sm_space;
        sd->ms_allocmap[3] += ms.ms_allocmap[3].sm_space;
        sd->ms_freemap[0] += ms.ms_freemap[0].sm_space;
        sd->ms_freemap[1] += ms.ms_freemap[1].sm_space;
        sd->ms_freemap[2] += ms.ms_freemap[2].sm_space;
        sd->ms_freemap[3] += ms.ms_freemap[3].sm_space;
        sd->ms_map += ms.ms_map.sm_space;
        sd->avail += ms.ms_map.sm_size - ms.ms_smo.smo_alloc;
        sd->nowavail += ms.ms_map.sm_size - ms.ms_smo_syncing.smo_alloc;

        if (sm.sm_phys != NULL) {
                (void) mdb_ctf_vread(&smp, "space_map_phys_t",
                    "mdb_space_map_phys_t", sm.sm_phys, 0);
        }

        sd->ms_tree += rt.rt_space;
        sd->avail += sm.sm_size - sm.sm_alloc;
        sd->nowavail += sm.sm_size - smp.smp_alloc;

        return (WALK_NEXT);
}

/*
 * ::spa_space [-b]
 *
 * Given a spa_t, print out it's on-disk space usage and in-core
 * estimates of future usage.  If -b is given, print space in bytes.
 * Otherwise print in megabytes.
 */
/* ARGSUSED */
static int
spa_space(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
{
        mdb_spa_t spa;
        uintptr_t dp_root_dir;
        mdb_dsl_dir_t dd;
        mdb_dsl_dir_phys_t dsp;
        uint64_t children;
        uintptr_t childaddr;
        space_data_t sd;
        int shift = 20;
        char *suffix = "M";
```

```
1551         int bytes = B_FALSE;

1553         if (mdb_getopts(argc, argv, 'b', MDB_OPT_SETBITS, TRUE, &bytes, NULL) !=
1554             argc)
1555                 return (DCMD_USAGE);
1556         if (!(flags & DCMD_ADDRSPEC))
1557                 return (DCMD_USAGE);

1559         if (bytes) {
1560                 shift = 0;
1561                 suffix = "";
1562         }

1564         if (GETMEMB(addr, "spa", spa_dsl_pool, spa.spa_dsl_pool) ||
1565             GETMEMB(addr, "spa", spa_root_vdev, spa.spa_root_vdev) ||
1566             GETMEMB(spa.spa_root_vdev, "vdev", vdev_children, children) ||
1567             GETMEMB(spa.spa_root_vdev, "vdev", vdev_child, childaddr) ||
1568             GETMEMB(spa.spa_dsl_pool, "dsl_pool",
1569             dp_root_dir, dp_root_dir) ||
1570             GETMEMB(dp_root_dir, "dsl_dir", dd_phys, dd.dd_phys) ||
1571             GETMEMB(dp_root_dir, "dsl_dir",
1572             dd_space_towrite, dd.dd_space_towrite) ||
1573             GETMEMB(dd.dd_phys, "dsl_dir_phys",
1574             dd_used_bytes, dsp.dd_used_bytes) ||
1575             GETMEMB(dd.dd_phys, "dsl_dir_phys",
1576             dd_compressed_bytes, dsp.dd_compressed_bytes) ||
1577             GETMEMB(dd.dd_phys, "dsl_dir_phys",
1578             dd_uncompressed_bytes, dsp.dd_uncompressed_bytes)) {
1579                 return (DCMD_ERR);
1580         }

1582         mdb_printf("dd_space_towrite = %llu%s %llu%s %llu%s %llu%s\n",
1583             dd.dd_space_towrite[0] >> shift, suffix,
1584             dd.dd_space_towrite[1] >> shift, suffix,
1585             dd.dd_space_towrite[2] >> shift, suffix,
1586             dd.dd_space_towrite[3] >> shift, suffix);

1588         mdb_printf("dd_phys.dd_used_bytes = %llu%s\n",
1589             dsp.dd_used_bytes >> shift, suffix);
1590         mdb_printf("dd_phys.dd_compressed_bytes = %llu%s\n",
1591             dsp.dd_compressed_bytes >> shift, suffix);
1592         mdb_printf("dd_phys.dd_uncompressed_bytes = %llu%s\n",
1593             dsp.dd_uncompressed_bytes >> shift, suffix);

1595         bzero(&sd, sizeof (sd));
1596         if (mdb_pwalk("metaslab", space_cb, &sd, addr) != 0) {
1597                 mdb_warn("can't walk metaslabs");
1598                 return (DCMD_ERR);
1599         }

1601         mdb_printf("ms_allocmap = %llu%s %llu%s %llu%s %llu%s\n",
1602             sd.ms_alloctree[0] >> shift, suffix,
1603             sd.ms_alloctree[1] >> shift, suffix,
1604             sd.ms_alloctree[2] >> shift, suffix,
1605             sd.ms_alloctree[3] >> shift, suffix);
1575             sd.ms_allocmap[0] >> shift, suffix,
1576             sd.ms_allocmap[1] >> shift, suffix,
1577             sd.ms_allocmap[2] >> shift, suffix,
1578             sd.ms_allocmap[3] >> shift, suffix);
1606         mdb_printf("ms_freemap = %llu%s %llu%s %llu%s %llu%s\n",
1607             sd.ms_freetree[0] >> shift, suffix,
1608             sd.ms_freetree[1] >> shift, suffix,
1609             sd.ms_freetree[2] >> shift, suffix,
1610             sd.ms_freetree[3] >> shift, suffix);
1611         mdb_printf("ms_tree = %llu%s\n", sd.ms_tree >> shift, suffix);
1580             sd.ms_freemap[0] >> shift, suffix,
```



```
1581             sd.ms_freemap[1] >> shift, suffix,
1582             sd.ms_freemap[2] >> shift, suffix,
1583             sd.ms_freemap[3] >> shift, suffix);
1584         mdb_printf("ms_map = %llu%s\n", sd.ms_map >> shift, suffix);
1612         mdb_printf("last synced avail = %llu%s\n", sd.avail >> shift, suffix);
1613         mdb_printf("current syncing avail = %llu%s\n",
1614             sd.nowavail >> shift, suffix);

1616         return (DCMD_OK);
1617 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   90879 Tue Sep  3 20:26:50 2013
new/usr/src/cmd/zdb/zdb.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 245 const char histo_stars[] = "****************************************";
 246 const int histo_width = sizeof (histo_stars) - 1;

 248 static void
 249 dump_histogram(const uint64_t *histo, int size, int offset)
 249 dump_histogram(const uint64_t *histo, int size)
 250 {
 251         int i;
 252         int minidx = size - 1;
 253         int maxidx = 0;
 254         uint64_t max = 0;

 256         for (i = 0; i < size; i++) {
 257                 if (histo[i] > max)
 258                         max = histo[i];
 259                 if (histo[i] > 0 && i > maxidx)
 260                         maxidx = i;
 261                 if (histo[i] > 0 && i < minidx)
 262                         minidx = i;
 263         }

 265         if (max < histo_width)
 266                 max = histo_width;

 268         for (i = minidx; i <= maxidx; i++) {
 269                 (void) printf("\t\t\t%3u: %6llu %s\n",
 270                     i + offset, (u_longlong_t)histo[i],
 270                     i, (u_longlong_t)histo[i],
 271                     &histo_stars[(max - histo[i]) * histo_width / max]);
 272         }
 273 }

 275 static void
 276 dump_zap_stats(objset_t *os, uint64_t object)
 277 {
 278         int error;
 279         zap_stats_t zs;

 281         error = zap_get_stats(os, object, &zs);
 282         if (error)
 283                 return;

 285         if (zs.zs_ptrtbl_len == 0) {
 286                 ASSERT(zs.zs_num_blocks == 1);
 287                 (void) printf("\tmicrozap: %llu bytes, %llu entries\n",
 288                     (u_longlong_t)zs.zs_blocksize,
 289                     (u_longlong_t)zs.zs_num_entries);
 290                 return;
 291         }

 293         (void) printf("\tFat ZAP stats:\n");
```



```
 295         (void) printf("\t\tPointer table:\n");
 296         (void) printf("\t\t\t%llu elements\n",
 297             (u_longlong_t)zs.zs_ptrtbl_len);
 298         (void) printf("\t\t\tzt_blk: %llu\n",
 299             (u_longlong_t)zs.zs_ptrtbl_zt_blk);
 300         (void) printf("\t\t\tzt_numblks: %llu\n",
 301             (u_longlong_t)zs.zs_ptrtbl_zt_numblks);
 302         (void) printf("\t\t\tzt_shift: %llu\n",
 303             (u_longlong_t)zs.zs_ptrtbl_zt_shift);
 304         (void) printf("\t\t\tzt_blks_copied: %llu\n",
 305             (u_longlong_t)zs.zs_ptrtbl_blks_copied);
 306         (void) printf("\t\t\tzt_nextblk: %llu\n",
 307             (u_longlong_t)zs.zs_ptrtbl_nextblk);

 309         (void) printf("\t\tZAP entries: %llu\n",
 310             (u_longlong_t)zs.zs_num_entries);
 311         (void) printf("\t\tLeaf blocks: %llu\n",
 312             (u_longlong_t)zs.zs_num_leafs);
 313         (void) printf("\t\tTotal blocks: %llu\n",
 314             (u_longlong_t)zs.zs_num_blocks);
 315         (void) printf("\t\tzap_block_type: 0x%llx\n",
 316             (u_longlong_t)zs.zs_block_type);
 317         (void) printf("\t\tzap_magic: 0x%llx\n",
 318             (u_longlong_t)zs.zs_magic);
 319         (void) printf("\t\tzap_salt: 0x%llx\n",
 320             (u_longlong_t)zs.zs_salt);

 322         (void) printf("\t\tLeafs with 2^n pointers:\n");
 323         dump_histogram(zs.zs_leafs_with_2n_pointers, ZAP_HISTOGRAM_SIZE, 0);
 323         dump_histogram(zs.zs_leafs_with_2n_pointers, ZAP_HISTOGRAM_SIZE);

 325         (void) printf("\t\tBlocks with n*5 entries:\n");
 326         dump_histogram(zs.zs_blocks_with_n5_entries, ZAP_HISTOGRAM_SIZE, 0);
 326         dump_histogram(zs.zs_blocks_with_n5_entries, ZAP_HISTOGRAM_SIZE);

 328         (void) printf("\t\tBlocks n/10 full:\n");
 329         dump_histogram(zs.zs_blocks_n_tenths_full, ZAP_HISTOGRAM_SIZE, 0);
 329         dump_histogram(zs.zs_blocks_n_tenths_full, ZAP_HISTOGRAM_SIZE);

 331         (void) printf("\t\tEntries with n chunks:\n");
 332         dump_histogram(zs.zs_entries_using_n_chunks, ZAP_HISTOGRAM_SIZE, 0);
 332         dump_histogram(zs.zs_entries_using_n_chunks, ZAP_HISTOGRAM_SIZE);

 334         (void) printf("\t\tBuckets with n entries:\n");
 335         dump_histogram(zs.zs_buckets_with_n_entries, ZAP_HISTOGRAM_SIZE, 0);
 335         dump_histogram(zs.zs_buckets_with_n_entries, ZAP_HISTOGRAM_SIZE);
 336 }
______unchanged_portion_omitted_

 524 int
 525 get_dtl_refcount(vdev_t *vd)
 526 {
 527         int refcount = 0;

 529         if (vd->vdev_ops->vdev_op_leaf) {
 530                 space_map_t *sm = vd->vdev_dtl_sm;

 532                 if (sm != NULL &&
 533                     sm->sm_dbuf->db_size == sizeof (space_map_phys_t))
 534                         return (1);
 535                 return (0);
 536         }

 538         for (int c = 0; c < vd->vdev_children; c++)
 539                 refcount += get_dtl_refcount(vd->vdev_child[c]);
```

```
        return (refcount);
}

int
get_metaslab_refcount(vdev_t *vd)
{
        int refcount = 0;

        if (vd->vdev_top == vd) {
                for (int m = 0; m < vd->vdev_ms_count; m++) {
                        space_map_t *sm = vd->vdev_ms[m]->ms_sm;

                        if (sm != NULL &&
                            sm->sm_dbuf->db_size == sizeof (space_map_phys_t))
                                refcount++;
                }
        }
        for (int c = 0; c < vd->vdev_children; c++)
                refcount += get_metaslab_refcount(vd->vdev_child[c]);

        return (refcount);
}

static int
verify_spacemap_refcounts(spa_t *spa)
{
        int expected_refcount, actual_refcount;

        expected_refcount = spa_feature_get_refcount(spa,
            &spa_feature_table[SPA_FEATURE_SPACEMAP_HISTOGRAM]);
        actual_refcount = get_dtl_refcount(spa->spa_root_vdev);
        actual_refcount += get_metaslab_refcount(spa->spa_root_vdev);

        if (expected_refcount != actual_refcount) {
                (void) printf("space map refcount mismatch: expected %d != "
                    "actual %d\n", expected_refcount, actual_refcount);
                return (2);
        }
        return (0);
}

static void
dump_spacemap(objset_t *os, space_map_t *sm)
dump_spacemap(objset_t *os, space_map_obj_t *smo, space_map_t *sm)
{
        uint64_t alloc, offset, entry;
        uint8_t mapshift = sm->sm_shift;
        uint64_t mapstart = sm->sm_start;
        char *ddata[] = { "ALLOC", "FREE", "CONDENSE", "INVALID",
                          "INVALID", "INVALID", "INVALID", "INVALID" };

        if (sm == NULL)
        if (smo->smo_object == 0)
                return;

        /*
         * Print out the freelist entries in both encoded and decoded form.
         */
        alloc = 0;
        for (offset = 0; offset < space_map_length(sm);
            offset += sizeof (entry)) {
                uint8_t mapshift = sm->sm_shift;

                VERIFY0(dmu_read(os, space_map_object(sm), offset,
        for (offset = 0; offset < smo->smo_objsize; offset += sizeof (entry)) {
                VERIFY3U(0, ==, dmu_read(os, smo->smo_object, offset,
```



```
                    sizeof (entry), &entry, DMU_READ_PREFETCH));
                if (SM_DEBUG_DECODE(entry)) {

                        (void) printf("\t    [%6llu] %s: txg %llu, pass %llu\n",
                            (u_longlong_t)(offset / sizeof (entry)),
                            ddata[SM_DEBUG_ACTION_DECODE(entry)],
                            (u_longlong_t)SM_DEBUG_TXG_DECODE(entry),
                            (u_longlong_t)SM_DEBUG_SYNCPASS_DECODE(entry));
                } else {
                        (void) printf("\t    [%6llu]    %c  range:"
                            " %010llx-%010llx  size: %06llx\n",
                            (u_longlong_t)(offset / sizeof (entry)),
                            SM_TYPE_DECODE(entry) == SM_ALLOC ? 'A' : 'F',
                            (u_longlong_t)((SM_OFFSET_DECODE(entry) <<
                            mapshift) + sm->sm_start),
                            mapshift) + mapstart),
                            (u_longlong_t)((SM_OFFSET_DECODE(entry) <<
                            mapshift) + sm->sm_start +
                            (SM_RUN_DECODE(entry) << mapshift)),
                            mapshift) + mapstart + (SM_RUN_DECODE(entry) <<
                            mapshift)),
                            (u_longlong_t)(SM_RUN_DECODE(entry) << mapshift));
                        if (SM_TYPE_DECODE(entry) == SM_ALLOC)
                                alloc += SM_RUN_DECODE(entry) << mapshift;
                        else
                                alloc -= SM_RUN_DECODE(entry) << mapshift;
                }
        }
        if (alloc != space_map_allocated(sm)) {
        if (alloc != smo->smo_alloc) {
                (void) printf("space_map_object alloc (%llu) INCONSISTENT "
                    "with space map summary (%llu)\n",
                    (u_longlong_t)space_map_allocated(sm), (u_longlong_t)alloc);
                    (u_longlong_t)smo->smo_alloc, (u_longlong_t)alloc);
        }
}

static void
dump_metaslab_stats(metaslab_t *msp)
{
        char maxbuf[32];
        range_tree_t *rt = msp->ms_tree;
        avl_tree_t *t = &msp->ms_size_tree;
        int free_pct = range_tree_space(rt) * 100 / msp->ms_size;
        space_map_t *sm = msp->ms_map;
        avl_tree_t *t = sm->sm_pp_root;
        int free_pct = sm->sm_space * 100 / sm->sm_size;

        zdb_nicenum(metaslab_block_maxsize(msp), maxbuf);
        zdb_nicenum(space_map_maxsize(sm), maxbuf);

        (void) printf("\t %25s %10lu   %7s  %6s   %4s %4d%%\n",
            "segments", avl_numnodes(t), "maxsize", maxbuf,
            "freepct", free_pct);
        (void) printf("\tIn-memory histogram:\n");
        dump_histogram(rt->rt_histogram, RANGE_TREE_HISTOGRAM_SIZE, 0);
}

static void
dump_metaslab(metaslab_t *msp)
{
        vdev_t *vd = msp->ms_group->mg_vd;
        spa_t *spa = vd->vdev_spa;
        space_map_t *sm = msp->ms_sm;
        space_map_t *sm = msp->ms_map;
        space_map_obj_t *smo = &msp->ms_smo;
```

```
655         char freebuf[32];

657         zdb_nicenum(msp->ms_size - space_map_allocated(sm), freebuf);
597         zdb_nicenum(sm->sm_size - smo->smo_alloc, freebuf);

659         (void) printf(
660             "\tmetaslab %6llu   offset %12llx   spacemap %6llu   free    %5s\n",
661             (u_longlong_t)msp->ms_id, (u_longlong_t)msp->ms_start,
662             (u_longlong_t)space_map_object(sm), freebuf);
601             (u_longlong_t)(sm->sm_start / sm->sm_size),
602             (u_longlong_t)sm->sm_start, (u_longlong_t)smo->smo_object, freebuf);

664         if (dump_opt['m'] > 2 && !dump_opt['L']) {
604         if (dump_opt['m'] > 1 && !dump_opt['L']) {
665                 mutex_enter(&msp->ms_lock);
666                 metaslab_load_wait(msp);
667                 if (!msp->ms_loaded) {
668                         VERIFY0(metaslab_load(msp));
669                         range_tree_stat_verify(msp->ms_tree);
670                 }
606                 space_map_load_wait(sm);
607                 if (!sm->sm_loaded)
608                         VERIFY(space_map_load(sm, zfs_metaslab_ops,
609                             SM_FREE, smo, spa->spa_meta_objset) == 0);
671                 dump_metaslab_stats(msp);
672                 metaslab_unload(msp);
611                 space_map_unload(sm);
673                 mutex_exit(&msp->ms_lock);
674         }

676         if (dump_opt['m'] > 1 && sm != NULL &&
677             spa_feature_is_active(spa,
678             &spa_feature_table[SPA_FEATURE_SPACEMAP_HISTOGRAM])) {
679                 /*
680                  * The space map histogram represents free space in chunks
681                  * of sm_shift (i.e. bucket 0 refers to 2^sm_shift).
682                  */
683                 (void) printf("\tOn-disk histogram:\n");
684                 dump_histogram(sm->sm_phys->smp_histogram,
685                     SPACE_MAP_HISTOGRAM_SIZE(sm), sm->sm_shift);
686         }
615         if (dump_opt['d'] > 5 || dump_opt['m'] > 2) {
616                 ASSERT(sm->sm_size == (1ULL << vd->vdev_ms_shift));

688         if (dump_opt['d'] > 5 || dump_opt['m'] > 3) {
689                 ASSERT(msp->ms_size == (1ULL << vd->vdev_ms_shift));

691                 mutex_enter(&msp->ms_lock);
692                 dump_spacemap(spa->spa_meta_objset, msp->ms_sm);
619                 dump_spacemap(spa->spa_meta_objset, smo, sm);
693                 mutex_exit(&msp->ms_lock);
694         }
695 }
```

_____***unchanged_portion_omitted_***

```
878 static void
879 dump_dtl_seg(void *arg, uint64_t start, uint64_t size)
806 dump_dtl_seg(space_map_t *sm, uint64_t start, uint64_t size)
880 {
881         char *prefix = arg;
808         char *prefix = (void *)sm;

883         (void) printf("%s [%llu,%llu) length %llu\n",
884             prefix,
885             (u_longlong_t)start,
886             (u_longlong_t)(start + size),
```



```
887             (u_longlong_t)(size));
888 }

890 static void
891 dump_dtl(vdev_t *vd, int indent)
892 {
893         spa_t *spa = vd->vdev_spa;
894         boolean_t required;
895         char *name[DTL_TYPES] = { "missing", "partial", "scrub", "outage" };
896         char prefix[256];

898         spa_vdev_state_enter(spa, SCL_NONE);
899         required = vdev_dtl_required(vd);
900         (void) spa_vdev_state_exit(spa, NULL, 0);

902         if (indent == 0)
903                 (void) printf("\nDirty time logs:\n\n");

905         (void) printf("\t%*s%s [%s]\n", indent, "",
906             vd->vdev_path ? vd->vdev_path :
907             vd->vdev_parent ? vd->vdev_ops->vdev_op_type : spa_name(spa),
908             required ? "DTL-required" : "DTL-expendable");

910         for (int t = 0; t < DTL_TYPES; t++) {
911                 range_tree_t *rt = vd->vdev_dtl[t];
912                 if (range_tree_space(rt) == 0)
838                 space_map_t *sm = &vd->vdev_dtl[t];
839                 if (sm->sm_space == 0)
913                         continue;
914                 (void) snprintf(prefix, sizeof (prefix), "\t%*s%s",
915                     indent + 2, "", name[t]);
916                 mutex_enter(rt->rt_lock);
917                 range_tree_walk(rt, dump_dtl_seg, prefix);
918                 mutex_exit(rt->rt_lock);
843                 mutex_enter(sm->sm_lock);
844                 space_map_walk(sm, dump_dtl_seg, (void *)prefix);
845                 mutex_exit(sm->sm_lock);
919                 if (dump_opt['d'] > 5 && vd->vdev_children == 0)
920                         dump_spacemap(spa->spa_meta_objset, vd->vdev_dtl_sm);
847                         dump_spacemap(spa->spa_meta_objset,
848                             &vd->vdev_dtl_smo, sm);
921         }

923         for (int c = 0; c < vd->vdev_children; c++)
924                 dump_dtl(vd->vdev_child[c], indent + 4);
925 }
```

_____***unchanged_portion_omitted_***

```
2295 static void
2296 zdb_leak(void *arg, uint64_t start, uint64_t size)
2224 zdb_leak(space_map_t *sm, uint64_t start, uint64_t size)
2297 {
2298         vdev_t *vd = arg;
2226         vdev_t *vd = sm->sm_ppd;

2300         (void) printf("leaked space: vdev %llu, offset 0x%llx, size %llu\n",
2301             (u_longlong_t)vd->vdev_id, (u_longlong_t)start, (u_longlong_t)size);
2302 }

2304 static metaslab_ops_t zdb_metaslab_ops = {
2232 /* ARGSUSED */
2233 static void
2234 zdb_space_map_load(space_map_t *sm)
2235 {
2236 }
```

```
2238 static void
2239 zdb_space_map_unload(space_map_t *sm)
2240 {
2241         space_map_vacate(sm, zdb_leak, sm);
2242 }

2244 /* ARGSUSED */
2245 static void
2246 zdb_space_map_claim(space_map_t *sm, uint64_t start, uint64_t size)
2247 {
2248 }

2250 static space_map_ops_t zdb_space_map_ops = {
2251         zdb_space_map_load,
2252         zdb_space_map_unload,
2305         NULL,   /* alloc */
2306         NULL    /* fragmented */
2254         zdb_space_map_claim,
2255         NULL,   /* free */
2256         NULL    /* maxsize */
2307 };
______unchanged_portion_omitted_

2349 static void
2350 zdb_leak_init(spa_t *spa, zdb_cb_t *zcb)
2351 {
2352         zcb->zcb_spa = spa;

2354         if (!dump_opt['L']) {
2355                 vdev_t *rvd = spa->spa_root_vdev;
2356                 for (int c = 0; c < rvd->vdev_children; c++) {
2357                         vdev_t *vd = rvd->vdev_child[c];
2358                         for (int m = 0; m < vd->vdev_ms_count; m++) {
2359                                 metaslab_t *msp = vd->vdev_ms[m];
2360                                 mutex_enter(&msp->ms_lock);
2361                                 metaslab_unload(msp);

2363                                 /*
2364                                  * For leak detection, we overload the metaslab
2365                                  * ms_tree to contain allocated segments
2366                                  * instead of free segments. As a result,
2367                                  * we can't use the normal metaslab_load/unload
2368                                  * interfaces.
2369                                  */
2370                                 if (msp->ms_sm != NULL) {
2371                                         msp->ms_ops = &zdb_metaslab_ops;
2372                                         VERIFY0(space_map_load(msp->ms_sm,
2373                                             msp->ms_tree, SM_ALLOC));
2374                                         msp->ms_loaded = B_TRUE;
2375                                 }
2311                                 space_map_unload(msp->ms_map);
2312                                 VERIFY(space_map_load(msp->ms_map,
2313                                     &zdb_space_map_ops, SM_ALLOC, &msp->ms_smo,
2314                                     spa->spa_meta_objset) == 0);
2315                                 msp->ms_map->sm_ppd = vd;
2376                                 mutex_exit(&msp->ms_lock);
2377                         }
2378                 }
2379         }

2381         spa_config_enter(spa, SCL_CONFIG, FTAG, RW_READER);

2383         zdb_ddt_leak_init(spa, zcb);

2385         spa_config_exit(spa, SCL_CONFIG, FTAG);
2386 }
```



```
2388 static void
2389 zdb_leak_fini(spa_t *spa)
2390 {
2391         if (!dump_opt['L']) {
2392                 vdev_t *rvd = spa->spa_root_vdev;
2393                 for (int c = 0; c < rvd->vdev_children; c++) {
2394                         vdev_t *vd = rvd->vdev_child[c];
2395                         for (int m = 0; m < vd->vdev_ms_count; m++) {
2396                                 metaslab_t *msp = vd->vdev_ms[m];
2397                                 mutex_enter(&msp->ms_lock);

2399                                 /*
2400                                  * The ms_tree has been overloaded to
2401                                  * contain allocated segments. Now that we
2402                                  * finished traversing all blocks, any
2403                                  * block that remains in the ms_tree
2404                                  * represents an allocated block that we
2405                                  * did not claim during the traversal.
2406                                  * Claimed blocks would have been removed
2407                                  * from the ms_tree.
2408                                  */
2409                                 range_tree_vacate(msp->ms_tree, zdb_leak, vd);
2410                                 msp->ms_loaded = B_FALSE;

2338                                 space_map_unload(msp->ms_map);
2412                                 mutex_exit(&msp->ms_lock);
2413                         }
2414                 }
2415         }
2416 }
______unchanged_portion_omitted_

2434 static int
2435 dump_block_stats(spa_t *spa)
2436 {
2437         zdb_cb_t zcb = { 0 };
2438         zdb_blkstats_t *zb, *tzb;
2439         uint64_t norm_alloc, norm_space, total_alloc, total_found;
2440         int flags = TRAVERSE_PRE | TRAVERSE_PREFETCH_METADATA | TRAVERSE_HARD;
2441         int leaks = 0;

2443         (void) printf("\nTraversing all blocks %s%s%s%s%s...\n\n",
2444             (dump_opt['c'] || !dump_opt['L']) ? "to verify " : "",
2445             (dump_opt['c'] == 1) ? "metadata " : "",
2446             dump_opt['c'] ? "checksums " : "",
2447             (dump_opt['c'] && !dump_opt['L']) ? "and verify " : "",
2448             !dump_opt['L'] ? "nothing leaked " : "");

2450         /*
2451          * Load all space maps as SM_ALLOC maps, then traverse the pool
2452          * claiming each block we discover.  If the pool is perfectly
2453          * consistent, the space maps will be empty when we're done.
2454          * Anything left over is a leak; any block we can't claim (because
2455          * it's not part of any space map) is a double allocation,
2456          * reference to a freed block, or an unclaimed log block.
2457          */
2458         zdb_leak_init(spa, &zcb);

2460         /*
2461          * If there's a deferred-free bplist, process that first.
2462          */
2463         (void) bpobj_iterate_nofree(&spa->spa_deferred_bpobj,
2464             count_block_cb, &zcb, NULL);
2465         if (spa_version(spa) >= SPA_VERSION_DEADLISTS) {
2466                 (void) bpobj_iterate_nofree(&spa->spa_dsl_pool->dp_free_bpobj,
```

```
2467                    count_block_cb, &zcb, NULL);
2468            }
2469            if (spa_feature_is_active(spa,
2470                &spa_feature_table[SPA_FEATURE_ASYNC_DESTROY])) {
2471                    VERIFY3U(0, ==, bptree_iterate(spa->spa_meta_objset,
2472                        spa->spa_dsl_pool->dp_bptree_obj, B_FALSE, count_block_cb,
2473                        &zcb, NULL));
2474            }

2476            if (dump_opt['c'] > 1)
2477                    flags |= TRAVERSE_PREFETCH_DATA;

2479            zcb.zcb_totalasize = metaslab_class_get_alloc(spa_normal_class(spa));
2480            zcb.zcb_start = zcb.zcb_lastprint = gethrtime();
2481            zcb.zcb_haderrors |= traverse_pool(spa, 0, flags, zdb_blkptr_cb, &zcb);

2483            /*
2484             * If we've traversed the data blocks then we need to wait for those
2485             * I/Os to complete. We leverage "The Godfather" zio to wait on
2486             * all async I/Os to complete.
2487             */
2488            if (dump_opt['c']) {
2489                    (void) zio_wait(spa->spa_async_zio_root);
2490                    spa->spa_async_zio_root = zio_root(spa, NULL, NULL,
2491                        ZIO_FLAG_CANFAIL | ZIO_FLAG_SPECULATIVE |
2492                        ZIO_FLAG_GODFATHER);
2493            }

2495            if (zcb.zcb_haderrors) {
2496                    (void) printf("\nError counts:\n\n");
2497                    (void) printf("\t%5s  %s\n", "errno", "count");
2498                    for (int e = 0; e < 256; e++) {
2499                            if (zcb.zcb_errors[e] != 0) {
2500                                    (void) printf("\t%5d  %llu\n",
2501                                        e, (u_longlong_t)zcb.zcb_errors[e]);
2502                            }
2503                    }
2504            }

2506            /*
2507             * Report any leaked segments.
2508             */
2509            zdb_leak_fini(spa);

2511            tzb = &zcb.zcb_type[ZB_TOTAL][ZDB_OT_TOTAL];

2513            norm_alloc = metaslab_class_get_alloc(spa_normal_class(spa));
2514            norm_space = metaslab_class_get_space(spa_normal_class(spa));

2516            total_alloc = norm_alloc + metaslab_class_get_alloc(spa_log_class(spa));
2517            total_found = tzb->zb_asize - zcb.zcb_dedup_asize;

2519            if (total_found == total_alloc) {
2520                    if (!dump_opt['L'])
2521                            (void) printf("\n\tNo leaks (block sum matches space"
2522                                " maps exactly)\n");
2523            } else {
2524                    (void) printf("block traversal size %llu != alloc %llu "
2525                        "(%s %lld)\n",
2526                        (u_longlong_t)total_found,
2527                        (u_longlong_t)total_alloc,
2528                        (dump_opt['L']) ? "unreachable" : "leaked",
2529                        (longlong_t)(total_alloc - total_found));
2530                    leaks = 1;
2531            }
```



```
2533            if (tzb->zb_count == 0)
2534                    return (2);

2536            (void) printf("\n");
2537            (void) printf("\tbp count:      %10llu\n",
2538                (u_longlong_t)tzb->zb_count);
2539            (void) printf("\tganged count:  %10llu\n",
2540                (longlong_t)tzb->zb_gangs);
2541            (void) printf("\tbp logical:    %10llu      avg: %6llu\n",
2542                (u_longlong_t)tzb->zb_lsize,
2543                (u_longlong_t)(tzb->zb_lsize / tzb->zb_count));
2544            (void) printf("\tbp physical:   %10llu      avg:"
2545                " %6llu     compression: %6.2f\n",
2546                (u_longlong_t)tzb->zb_psize,
2547                (u_longlong_t)(tzb->zb_psize / tzb->zb_count),
2548                (double)tzb->zb_lsize / tzb->zb_psize);
2549            (void) printf("\tbp allocated:  %10llu      avg:"
2550                " %6llu     compression: %6.2f\n",
2551                (u_longlong_t)tzb->zb_asize,
2552                (u_longlong_t)(tzb->zb_asize / tzb->zb_count),
2553                (double)tzb->zb_lsize / tzb->zb_asize);
2554            (void) printf("\tbp deduped:    %10llu    ref>1:"
2555                " %6llu   deduplication: %6.2f\n",
2556                (u_longlong_t)zcb.zcb_dedup_asize,
2557                (u_longlong_t)zcb.zcb_dedup_blocks,
2558                (double)zcb.zcb_dedup_asize / tzb->zb_asize + 1.0);
2559            (void) printf("\tSPA allocated: %10llu     used: %5.2f%%\n",
2560                (u_longlong_t)norm_alloc, 100.0 * norm_alloc / norm_space);

2562            if (tzb->zb_ditto_samevdev != 0) {
2563                    (void) printf("\tDittoed blocks on same vdev: %llu\n",
2564                        (longlong_t)tzb->zb_ditto_samevdev);
2565            }

2567            if (dump_opt['b'] >= 2) {
2568                    int l, t, level;
2569                    (void) printf("\nBlocks\tLSIZE\tPSIZE\tASIZE"
2570                        "\t  avg\t comp\t%%Total\tType\n");

2572                    for (t = 0; t <= ZDB_OT_TOTAL; t++) {
2573                            char csize[32], lsize[32], psize[32], asize[32];
2574                            char avg[32], gang[32];
2575                            char *typename;

2577                            if (t < DMU_OT_NUMTYPES)
2578                                    typename = dmu_ot[t].ot_name;
2579                            else
2580                                    typename = zdb_ot_extname[t - DMU_OT_NUMTYPES];

2582                            if (zcb.zcb_type[ZB_TOTAL][t].zb_asize == 0) {
2583                                    (void) printf("%6s\t%5s\t%5s\t%5s\t%5s"
2584                                        "\t%5s\t%5s\t%6s\t%s\n",
2585                                        "-",
2586                                        "-",
2587                                        "-",
2588                                        "-",
2589                                        "-",
2590                                        "-",
2591                                        "-",
2592                                        typename);
2593                                    continue;
2594                            }

2596                            for (l = ZB_TOTAL - 1; l >= -1; l--) {
2597                                    level = (l == -1 ? ZB_TOTAL : l);
2598                                    zb = &zcb.zcb_type[level][t];
```

```
2600                                if (zb->zb_asize == 0)
2601                                        continue;

2603                                if (dump_opt['b'] < 3 && level != ZB_TOTAL)
2604                                        continue;

2606                                if (level == 0 && zb->zb_asize ==
2607                                    zcb.zcb_type[ZB_TOTAL][t].zb_asize)
2608                                        continue;

2610                                zdb_nicenum(zb->zb_count, csize);
2611                                zdb_nicenum(zb->zb_lsize, lsize);
2612                                zdb_nicenum(zb->zb_psize, psize);
2613                                zdb_nicenum(zb->zb_asize, asize);
2614                                zdb_nicenum(zb->zb_asize / zb->zb_count, avg);
2615                                zdb_nicenum(zb->zb_gangs, gang);

2617                                (void) printf("%6s\t%5s\t%5s\t%5s\t%5s"
2618                                    "\t%5.2f\t%6.2f\t",
2619                                    csize, lsize, psize, asize, avg,
2620                                    (double)zb->zb_lsize / zb->zb_psize,
2621                                    100.0 * zb->zb_asize / tzb->zb_asize);

2623                                if (level == ZB_TOTAL)
2624                                        (void) printf("%s\n", typename);
2625                                else
2626                                        (void) printf("    L%d %s\n",
2627                                            level, typename);

2629                                if (dump_opt['b'] >= 3 && zb->zb_gangs > 0) {
2630                                        (void) printf("\t number of ganged "
2631                                            "blocks: %s\n", gang);
2632                                }

2634                                if (dump_opt['b'] >= 4) {
2635                                        (void) printf("psize "
2636                                            "(in 512-byte sectors): "
2637                                            "number of blocks\n");
2638                                        dump_histogram(zb->zb_psize_histogram,
2639                                            PSIZE_HISTO_SIZE, 0);
2566                                            PSIZE_HISTO_SIZE);
2640                                }
2641                        }
2642                }
2643        }

2645        (void) printf("\n");

2647        if (leaks)
2648                return (2);

2650        if (zcb.zcb_haderrors)
2651                return (3);

2653        return (0);
2654 }
```

*_____unchanged_portion_omitted_*

```
2757 static void
2758 dump_zpool(spa_t *spa)
2759 {
2760        dsl_pool_t *dp = spa_get_dsl(spa);
2761        int rc = 0;

2763        if (dump_opt['S']) {
```



```
2764                dump_simulated_ddt(spa);
2765                return;
2766        }

2768        if (!dump_opt['e'] && dump_opt['C'] > 1) {
2769                (void) printf("\nCached configuration:\n");
2770                dump_nvlist(spa->spa_config, 8);
2771        }

2773        if (dump_opt['C'])
2774                dump_config(spa);

2776        if (dump_opt['u'])
2777                dump_uberblock(&spa->spa_uberblock, "\nUberblock:\n", "\n");

2779        if (dump_opt['D'])
2780                dump_all_ddts(spa);

2782        if (dump_opt['d'] > 2 || dump_opt['m'])
2783                dump_metaslabs(spa);

2785        if (dump_opt['d'] || dump_opt['i']) {
2786                dump_dir(dp->dp_meta_objset);
2787                if (dump_opt['d'] >= 3) {
2788                        dump_bpobj(&spa->spa_deferred_bpobj,
2789                            "Deferred frees", 0);
2790                        if (spa_version(spa) >= SPA_VERSION_DEADLISTS) {
2791                                dump_bpobj(&spa->spa_dsl_pool->dp_free_bpobj,
2792                                    "Pool snapshot frees", 0);
2793                        }

2795                        if (spa_feature_is_active(spa,
2796                            &spa_feature_table[SPA_FEATURE_ASYNC_DESTROY])) {
2797                                dump_bptree(spa->spa_meta_objset,
2798                                    spa->spa_dsl_pool->dp_bptree_obj,
2799                                    "Pool dataset frees");
2800                        }
2801                        dump_dtl(spa->spa_root_vdev, 0);
2802                }
2803                (void) dmu_objset_find(spa_name(spa), dump_one_dir,
2804                    NULL, DS_FIND_SNAPSHOTS | DS_FIND_CHILDREN);
2805        }
2806        if (dump_opt['b'] || dump_opt['c'])
2807                rc = dump_block_stats(spa);

2809        if (rc == 0)
2810                rc = verify_spacemap_refcounts(spa);

2812        if (dump_opt['s'])
2813                show_pool_stats(spa);

2815        if (dump_opt['h'])
2816                dump_history(spa);

2818        if (rc != 0)
2819                exit(rc);
2820 }
```

*_____unchanged_portion_omitted_*

```
***********************************************************
  160708 Tue Sep  3 20:26:52 2013
new/usr/src/cmd/ztest/ztest.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
***********************************************************
______unchanged_portion_omitted_

5320 static void *
5321 ztest_deadman_thread(void *arg)
5322 {
5323         ztest_shared_t *zs = arg;
5324         spa_t *spa = ztest_spa;
5325         hrtime_t delta, total = 0;

5327         for (;;) {
5328                 delta = zs->zs_thread_stop - zs->zs_thread_start +
5329                     MSEC2NSEC(zfs_deadman_synctime_ms);

5331                 (void) poll(NULL, 0, (int)NSEC2MSEC(delta));

5333                 /*
5334                  * If the pool is suspended then fail immediately. Otherwise,
5335                  * check to see if the pool is making any progress. If
5336                  * vdev_deadman() discovers that there hasn't been any recent
5337                  * I/Os then it will end up aborting the tests.
5338                  */
5339                 if (spa_suspended(spa) || spa->spa_root_vdev == NULL) {
5339                 if (spa_suspended(spa)) {
5340                         fatal(0, "aborting test after %llu seconds because "
5341                             "pool has transitioned to a suspended state.",
5342                             zfs_deadman_synctime_ms / 1000);
5343                         return (NULL);
5344                 }
5345                 vdev_deadman(spa->spa_root_vdev);

5347                 total += zfs_deadman_synctime_ms/1000;
5348                 (void) printf("ztest has been running for %lld seconds\n",
5349                     total);
5350         }
5351 }
______unchanged_portion_omitted_
```

```
**********************************************************
    4629 Tue Sep  3 20:26:53 2013
new/usr/src/common/zfs/zfeature_common.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 */

#ifdef _KERNEL
#include <sys/systm.h>
#else
#include <errno.h>
#include <string.h>
#endif
#include <sys/debug.h>
#include <sys/fs/zfs.h>
#include <sys/inttypes.h>
#include <sys/types.h>
#include "zfeature_common.h"

/*
 * Set to disable all feature checks while opening pools, allowing pools with
 * unsupported features to be opened. Set for testing only.
 */
boolean_t zfeature_checks_disable = B_FALSE;

zfeature_info_t spa_feature_table[SPA_FEATURES];

/*
 * Valid characters for feature guids. This list is mainly for aesthetic
 * purposes and could be expanded in the future. There are different allowed
 * characters in the guids reverse dns portion (before the colon) and its
 * short name (after the colon).
```



```
 */
static int
valid_char(char c, boolean_t after_colon)
{
        return ((c >= 'a' && c <= 'z') ||
            (c >= '0' && c <= '9') ||
            c == (after_colon ? '_' : '.'));
}
_____unchanged_portion_omitted_

void
zpool_feature_init(void)
{
        zfeature_register(SPA_FEATURE_ASYNC_DESTROY,
            "com.delphix:async_destroy", "async_destroy",
            "Destroy filesystems asynchronously.", B_TRUE, B_FALSE, NULL);
        zfeature_register(SPA_FEATURE_EMPTY_BPOBJ,
            "com.delphix:empty_bpobj", "empty_bpobj",
            "Snapshots use less space.", B_TRUE, B_FALSE, NULL);
        zfeature_register(SPA_FEATURE_LZ4_COMPRESS,
            "org.illumos:lz4_compress", "lz4_compress",
            "LZ4 compression algorithm support.", B_FALSE, B_FALSE, NULL);
        zfeature_register(SPA_FEATURE_MULTI_VDEV_CRASH_DUMP,
            "com.joyent:multi_vdev_crash_dump", "multi_vdev_crash_dump",
            "Crash dumps to multiple vdev pools.", B_FALSE, B_FALSE, NULL);
        zfeature_register(SPA_FEATURE_SPACEMAP_HISTOGRAM,
            "com.delphix:spacemap_histogram", "spacemap_histogram",
            "Spacemaps maintain space histograms.", B_TRUE, B_FALSE, NULL);
}
_____unchanged_portion_omitted_
```

****
2304 Tue Sep 3 20:26:54 2013
new/usr/src/common/zfs/zfeature_common.h
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 */

#ifndef _ZFEATURE_COMMON_H
#define _ZFEATURE_COMMON_H

#include <sys/fs/zfs.h>
#include <sys/inttypes.h>
#include <sys/types.h>

#ifdef  __cplusplus
extern "C" {
#endif

struct zfeature_info;

typedef struct zfeature_info {
        const char *fi_uname;   /* User-facing feature name */
        const char *fi_guid;    /* On-disk feature identifier */
        const char *fi_desc;    /* Feature description */
        boolean_t fi_can_readonly; /* Can open pool readonly w/o support? */
        boolean_t fi_mos;       /* Is the feature necessary to read the MOS? */
        struct zfeature_info **fi_depends; /* array; null terminated */
} zfeature_info_t;

typedef int (zfeature_func_t)(zfeature_info_t *fi, void *arg);

#define ZFS_FEATURE_DEBUG
```



```
enum spa_feature {
        SPA_FEATURE_ASYNC_DESTROY,
        SPA_FEATURE_EMPTY_BPOBJ,
        SPA_FEATURE_LZ4_COMPRESS,
        SPA_FEATURE_MULTI_VDEV_CRASH_DUMP,
        SPA_FEATURE_SPACEMAP_HISTOGRAM,
        SPA_FEATURES
} spa_feature_t;
```

_____unchanged_portion_omitted_

```
**********************************************************
    9200 Tue Sep  3 20:26:55 2013
new/usr/src/man/man5/zpool-features.5
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
```

```
'\" te
.\" Copyright (c) 2012 by Delphix. All rights reserved.
.\" Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
.\" Copyright (c) 2013, Joyent, Inc. All rights reserved.
.\" 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]
.TH ZPOOL-FEATURES 5 "Aug 27, 2013"
.TH ZPOOL-FEATURES 5 "Mar 16, 2012"
.SH NAME
zpool\-features \- ZFS pool feature descriptions
.SH DESCRIPTION
.sp
.LP
ZFS pool on\-disk format versions are specified via "features" which replace
the old on\-disk format numbers (the last supported on\-disk format number is
28). To enable a feature on a pool use the \fBupgrade\fR subcommand of the
\fBzpool\fR(1M) command, or set the \fBfeature@\fR\fIfeature_name\fR property
to \fBenabled\fR.
.sp
.LP
The pool format does not affect file system version compatibility or the ability
to send file systems between pools.
.sp
.LP
Since most features can be enabled independently of each other the on\-disk
format of the pool is specified by the set of all features marked as
\fBactive\fR on the pool. If the pool was created by another software version
this set may include unsupported features.
.SS "Identifying features"
.sp
.LP
Every feature has a guid of the form \fIcom.example:feature_name\fR. The reverse
DNS name ensures that the feature's guid is unique across all ZFS
implementations. When unsupported features are encountered on a pool they will
be identified by their guids. Refer to the documentation for the ZFS
implementation that created the pool for information about those features.
.sp
.LP
Each supported feature also has a short name. By convention a feature's short
name is the portion of its guid which follows the ':' (e.g.
\fIcom.example:feature_name\fR would have the short name \fIfeature_name\fR),
however a feature's short name may differ across ZFS implementations if
following the convention would result in name conflicts.
```



```
.SS "Feature states"
.sp
.LP
Features can be in one of three states:
.sp
.ne 2
.na
\fB\fBactive\fR\fR
.ad
.RS 12n
This feature's on\-disk format changes are in effect on the pool. Support for
this feature is required to import the pool in read\-write mode. If this
feature is not read-only compatible, support is also required to import the pool
in read\-only mode (see "Read\-only compatibility").
.RE

.sp
.ne 2
.na
\fB\fBenabled\fR\fR
.ad
.RS 12n
An administrator has marked this feature as enabled on the pool, but the
feature's on\-disk format changes have not been made yet. The pool can still be
imported by software that does not support this feature, but changes may be made
to the on\-disk format at any time which will move the feature to the
\fBactive\fR state. Some features may support returning to the \fBenabled\fR
state after becoming \fBactive\fR. See feature\-specific documentation for
details.
.RE

.sp
.ne 2
.na
\fBdisabled\fR
.ad
.RS 12n
This feature's on\-disk format changes have not been made and will not be made
unless an administrator moves the feature to the \fBenabled\fR state. Features
cannot be disabled once they have been enabled.
.RE

.sp
.LP
The state of supported features is exposed through pool properties of the form
\fIfeature@short_name\fR.
.SS "Read\-only compatibility"
.sp
.LP
Some features may make on\-disk format changes that do not interfere with other
software's ability to read from the pool. These features are referred to as
"read\-only compatible". If all unsupported features on a pool are read\-only
compatible, the pool can be imported in read\-only mode by setting the
\fBreadonly\fR property during import (see \fBzpool\fR(1M) for details on
importing pools).
.SS "Unsupported features"
.sp
.LP
For each unsupported feature enabled on an imported pool a pool property
named \fIunsupported@feature_guid\fR will indicate why the import was allowed
despite the unsupported feature. Possible values for this property are:

.sp
.ne 2
.na
\fB\fBinactive\fR\fR
```

```
.ad
.RS 12n
The feature is in the \fBenabled\fR state and therefore the pool's on\-disk
format is still compatible with software that does not support this feature.
.RE

.sp
.ne 2
.na
\fB\fBreadonly\fR\fR
.ad
.RS 12n
The feature is read\-only compatible and the pool has been imported in
read\-only mode.
.RE

.SS "Feature dependencies"
.sp
.LP
Some features depend on other features being enabled in order to function
properly. Enabling a feature will automatically enable any features it
depends on.
.SH FEATURES
.sp
.LP
The following features are supported on this system:
.sp
.ne 2
.na
\fB\fBasync_destroy\fR\fR
.ad
.RS 4n
.TS
l l .
GUID    com.delphix:async_destroy
READ\-ONLY COMPATIBLE   yes
DEPENDENCIES    none
.TE

Destroying a file system requires traversing all of its data in order to
return its used space to the pool. Without \fBasync_destroy\fR the file system
is not fully removed until all space has been reclaimed. If the destroy
operation is interrupted by a reboot or power outage the next attempt to open
the pool will need to complete the destroy operation synchronously.

When \fBasync_destroy\fR is enabled the file system's data will be reclaimed
by a background process, allowing the destroy operation to complete without
traversing the entire file system. The background process is able to resume
interrupted destroys after the pool has been opened, eliminating the need
to finish interrupted destroys as part of the open operation. The amount
of space remaining to be reclaimed by the background process is available
through the \fBfreeing\fR property.

This feature is only \fBactive\fR while \fBfreeing\fR is non\-zero.
.RE

.sp
.ne 2
.na
\fB\fBempty_bpobj\fR\fR
.ad
.RS 4n
.TS
l l .
GUID    com.delphix:empty_bpobj
READ\-ONLY COMPATIBLE   yes
```



```
DEPENDENCIES    none
.TE

This feature increases the performance of creating and using a large
number of snapshots of a single filesystem or volume, and also reduces
the disk space required.

When there are many snapshots, each snapshot uses many Block Pointer
Objects (bpobj's) to track blocks associated with that snapshot.
However, in common use cases, most of these bpobj's are empty.  This
feature allows us to create each bpobj on-demand, thus eliminating the
empty bpobjs.

This feature is \fBactive\fR while there are any filesystems, volumes,
or snapshots which were created after enabling this feature.
.RE

.sp
.ne 2
.na
\fB\fBlz4_compress\fR\fR
.ad
.RS 4n
.TS
l l .
GUID    org.illumos:lz4_compress
READ\-ONLY COMPATIBLE   no
DEPENDENCIES    none
.TE

\fBlz4\fR is a high-performance real-time compression algorithm that
features significantly faster compression and decompression as well as a
higher compression ratio than the older \fBlzjb\fR compression.
Typically, \fBlz4\fR compression is approximately 50% faster on
compressible data and 200% faster on incompressible data than
\fBlzjb\fR. It is also approximately 80% faster on decompression, while
giving approximately 10% better compression ratio.

When the \fBlz4_compress\fR feature is set to \fBenabled\fR, the
administrator can turn on \fBlz4\fR compression on any dataset on the
pool using the \fBzfs\fR(1M) command. Please note that doing so will
immediately activate the \fBlz4_compress\fR feature on the underlying
pool (even before any data is written). Since this feature is not
read-only compatible, this operation will render the pool unimportable
on systems without support for the \fBlz4_compress\fR feature. At the
moment, this operation cannot be reversed. Booting off of
\fBlz4\fR-compressed root pools is supported.
.RE

.sp
.ne 2
.na
\fB\fBspacemap_histogram\fR\fR
.ad
.RS 4n
.TS
l l .
GUID    com.delphix:spacemap_histogram
READ\-ONLY COMPATIBLE   yes
DEPENDENCIES    none
.TE

This features allows ZFS to maintain more information about how free space
is organized within the pool. If this feature is \fBenabled\fR, ZFS will
set this feature to \fBactive\fR when a new space map object is created or
an existing space map is upgraded to the new format. Once the feature is
```

```
\fBactive\fR, it will remain in that state until the pool is destroyed.
.RE

.sp
.ne 2
.na
\fB\fBmulti_vdev_crash_dump\fR\fR
.ad
.RS 4n
.TS
l l .
GUID    com.joyent:multi_vdev_crash_dump
READ\-ONLY COMPATIBLE   no
DEPENDENCIES    none
.TE

This feature allows a dump device to be configured with a pool comprised
of multiple vdevs.  Those vdevs may be arranged in any mirrored or raidz
configuration.

When the \fBmulti_vdev_crash_dump\fR feature is set to \fBenabled\fR,
the administrator can use the \fBdumpadm\fR(1M) command to configure a
dump device on a pool comprised of multiple vdevs.

.SH "SEE ALSO"
\fBzpool\fR(1M)
```

```
**********************************************************
   43700 Tue Sep  3 20:26:56 2013
new/usr/src/uts/common/Makefile.files
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
# Copyright (c) 2013 by Delphix. All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
#

#
# This Makefile defines all file modules for the directory uts/common
# and its children. These are the source files which may be considered
# common to all SunOS systems.

i386_CORE_OBJS += \
                atomic.o        \
                avintr.o        \
                pic.o

sparc_CORE_OBJS +=

COMMON_CORE_OBJS +=             \
                beep.o          \
                bitset.o        \
                bp_map.o        \
                brand.o         \
                cpucaps.o       \
                cmt.o           \
                cmt_policy.o    \
                cpu.o           \
                cpu_event.o     \
                cpu_intr.o      \
                cpu_pm.o        \
```



```
                cpupart.o       \
                cap_util.o      \
                disp.o          \
                group.o         \
                kstat_fr.o      \
                iscsiboot_prop.o        \
                lgrp.o          \
                lgrp_topo.o     \
                mmapobj.o       \
                mutex.o         \
                page_lock.o     \
                page_retire.o   \
                panic.o         \
                param.o         \
                pg.o            \
                pghw.o          \
                putnext.o       \
                rctl_proc.o     \
                rwlock.o        \
                seg_kmem.o      \
                softint.o       \
                string.o        \
                strtol.o        \
                strtoul.o       \
                strtoll.o       \
                strtoull.o      \
                thread_intr.o   \
                vm_page.o       \
                vm_pagelist.o   \
                zlib_obj.o      \
                clock_tick.o

CORE_OBJS +=    $(COMMON_CORE_OBJS) $($(MACH)_CORE_OBJS)

ZLIB_OBJS =     zutil.o zmod.o zmod_subr.o \
                adler32.o crc32.o deflate.o inffast.o \
                inflate.o inftrees.o trees.o

GENUNIX_OBJS += \
                access.o        \
                acl.o           \
                acl_common.o    \
                adjtime.o       \
                alarm.o         \
                aio_subr.o      \
                auditsys.o      \
                audit_core.o    \
                audit_zone.o    \
                audit_memory.o  \
                autoconf.o      \
                avl.o           \
                bdev_dsort.o    \
                bio.o           \
                bitmap.o        \
                blabel.o        \
                brandsys.o      \
                bz2blocksort.o  \
                bz2compress.o   \
                bz2decompress.o \
                bz2randtable.o  \
                bz2bzlib.o      \
                bz2crctable.o   \
                bz2huffman.o    \
                callb.o         \
                callout.o       \
                chdir.o         \
```

```
119                 chmod.o         \
120                 chown.o         \
121                 cladm.o         \
122                 class.o         \
123                 clock.o         \
124                 clock_highres.o \
125                 clock_realtime.o\
126                 close.o         \
127                 compress.o      \
128                 condvar.o       \
129                 conf.o          \
130                 console.o       \
131                 contract.o      \
132                 copyops.o       \
133                 core.o          \
134                 corectl.o       \
135                 cred.o          \
136                 cs_stubs.o      \
137                 dacf.o          \
138                 dacf_clnt.o     \
139                 damap.o \
140                 cyclic.o        \
141                 ddi.o           \
142                 ddifm.o         \
143                 ddi_hp_impl.o   \
144                 ddi_hp_ndi.o    \
145                 ddi_intr.o      \
146                 ddi_intr_impl.o \
147                 ddi_intr_irm.o  \
148                 ddi_nodeid.o    \
149                 ddi_periodic.o  \
150                 devcfg.o        \
151                 devcache.o      \
152                 device.o        \
153                 devid.o         \
154                 devid_cache.o   \
155                 devid_scsi.o    \
156                 devid_smp.o     \
157                 devpolicy.o     \
158                 disp_lock.o     \
159                 dnlc.o          \
160                 driver.o        \
161                 dumpsubr.o      \
162                 driver_lyr.o    \
163                 dtrace_subr.o   \
164                 errorq.o        \
165                 etheraddr.o     \
166                 evchannels.o    \
167                 exacct.o        \
168                 exacct_core.o   \
169                 exec.o          \
170                 exit.o          \
171                 fbio.o          \
172                 fcntl.o         \
173                 fdbuffer.o      \
174                 fdsync.o        \
175                 fem.o           \
176                 ffs.o           \
177                 fio.o           \
178                 flock.o         \
179                 fm.o            \
180                 fork.o          \
181                 vpm.o           \
182                 fs_reparse.o    \
183                 fs_subr.o       \
184                 fsflush.o       \
```



```
185                 ftrace.o        \
186                 getcwd.o        \
187                 getdents.o      \
188                 getloadavg.o    \
189                 getpagesizes.o  \
190                 getpid.o        \
191                 gfs.o           \
192                 rusagesys.o     \
193                 gid.o           \
194                 groups.o        \
195                 grow.o          \
196                 hat_refmod.o    \
197                 id32.o          \
198                 id_space.o      \
199                 inet_ntop.o     \
200                 instance.o      \
201                 ioctl.o         \
202                 ip_cksum.o      \
203                 issetugid.o     \
204                 ippconf.o       \
205                 kcpc.o          \
206                 kdi.o           \
207                 kiconv.o        \
208                 klpd.o          \
209                 kmem.o          \
210                 ksyms_snapshot.o        \
211                 l_strplumb.o    \
212                 labelsys.o      \
213                 link.o          \
214                 list.o          \
215                 lockstat_subr.o \
216                 log_sysevent.o  \
217                 logsubr.o       \
218                 lookup.o        \
219                 lseek.o         \
220                 ltos.o          \
221                 lwp.o           \
222                 lwp_create.o    \
223                 lwp_info.o      \
224                 lwp_self.o      \
225                 lwp_sobj.o      \
226                 lwp_timer.o     \
227                 lwpsys.o        \
228                 main.o          \
229                 mmapobjsys.o    \
230                 memcntl.o       \
231                 memstr.o        \
232                 lgrpsys.o       \
233                 mkdir.o         \
234                 mknod.o         \
235                 mount.o         \
236                 move.o          \
237                 msacct.o        \
238                 multidata.o     \
239                 nbmlock.o       \
240                 ndifm.o         \
241                 nice.o          \
242                 netstack.o      \
243                 ntptime.o       \
244                 nvpair.o        \
245                 nvpair_alloc_system.o   \
246                 nvpair_alloc_fixed.o    \
247                 fnvpair.o       \
248                 octet.o         \
249                 open.o          \
250                 p_online.o      \
```

```
251                 pathconf.o      \
252                 pathname.o      \
253                 pause.o         \
254                 serializer.o    \
255                 pci_intr_lib.o  \
256                 pci_cap.o       \
257                 pcifm.o         \
258                 pgrp.o          \
259                 pgrpsys.o       \
260                 pid.o           \
261                 pkp_hash.o      \
262                 policy.o        \
263                 poll.o          \
264                 pool.o          \
265                 pool_pset.o     \
266                 port_subr.o     \
267                 ppriv.o         \
268                 printf.o        \
269                 priocntl.o      \
270                 priv.o          \
271                 priv_const.o    \
272                 proc.o          \
273                 procset.o       \
274                 processor_bind.o        \
275                 processor_info.o        \
276                 profil.o        \
277                 project.o       \
278                 qsort.o         \
279                 rctl.o          \
280                 rctlsys.o       \
281                 readlink.o      \
282                 refstr.o        \
283                 rename.o        \
284                 resolvepath.o   \
285                 retire_store.o  \
286                 process.o       \
287                 rlimit.o        \
288                 rmap.o          \
289                 rw.o            \
290                 rwstlock.o      \
291                 sad_conf.o      \
292                 sid.o           \
293                 sidsys.o        \
294                 sched.o         \
295                 schedctl.o      \
296                 sctp_crc32.o    \
297                 seg_dev.o       \
298                 seg_kp.o        \
299                 seg_kpm.o       \
300                 seg_map.o       \
301                 seg_vn.o        \
302                 seg_spt.o       \
303                 semaphore.o     \
304                 sendfile.o      \
305                 session.o       \
306                 share.o         \
307                 shuttle.o       \
308                 sig.o           \
309                 sigaction.o     \
310                 sigaltstack.o   \
311                 signotify.o     \
312                 sigpending.o    \
313                 sigprocmask.o   \
314                 sigqueue.o      \
315                 sigsendset.o    \
316                 sigsuspend.o    \
```



```
317                 sigtimedwait.o  \
318                 sleepq.o        \
319                 sock_conf.o     \
320                 space.o         \
321                 sscanf.o        \
322                 stat.o          \
323                 statfs.o        \
324                 statvfs.o       \
325                 stol.o          \
326                 str_conf.o      \
327                 strcalls.o      \
328                 stream.o        \
329                 streamio.o      \
330                 strext.o        \
331                 strsubr.o       \
332                 strsun.o        \
333                 subr.o          \
334                 sunddi.o        \
335                 sunmdi.o        \
336                 sunndi.o        \
337                 sunpci.o        \
338                 sunpm.o         \
339                 sundlpi.o       \
340                 suntpi.o        \
341                 swap_subr.o     \
342                 swap_vnops.o    \
343                 symlink.o       \
344                 sync.o          \
345                 sysclass.o      \
346                 sysconfig.o     \
347                 sysent.o        \
348                 sysfs.o         \
349                 systeminfo.o    \
350                 task.o          \
351                 taskq.o         \
352                 tasksys.o       \
353                 time.o          \
354                 timer.o         \
355                 times.o         \
356                 timers.o        \
357                 thread.o        \
358                 tlabel.o        \
359                 tnf_res.o       \
360                 turnstile.o     \
361                 tty_common.o    \
362                 u8_textprep.o   \
363                 uadmin.o        \
364                 uconv.o         \
365                 ucredsys.o      \
366                 uid.o           \
367                 umask.o         \
368                 umount.o        \
369                 uname.o         \
370                 unix_bb.o       \
371                 unlink.o        \
372                 urw.o           \
373                 utime.o         \
374                 utssys.o        \
375                 uucopy.o        \
376                 vfs.o           \
377                 vfs_conf.o      \
378                 vmem.o          \
379                 vm_anon.o       \
380                 vm_as.o         \
381                 vm_meter.o      \
382                 vm_pageout.o    \
```

```
383                 vm_pvn.o        \
384                 vm_rm.o         \
385                 vm_seg.o        \
386                 vm_subr.o       \
387                 vm_swap.o       \
388                 vm_usage.o      \
389                 vnode.o         \
390                 vuid_queue.o    \
391                 vuid_store.o    \
392                 waitq.o         \
393                 watchpoint.o    \
394                 yield.o         \
395                 scsi_confdata.o \
396                 xattr.o         \
397                 xattr_common.o  \
398                 xdr_mblk.o      \
399                 xdr_mem.o       \
400                 xdr.o           \
401                 xdr_array.o     \
402                 xdr_refer.o     \
403                 xhat.o          \
404                 zone.o

406 #
407 #       Stubs for the stand-alone linker/loader
408 #
409 sparc_GENSTUBS_OBJS =   \
410         kobj_stubs.o

412 i386_GENSTUBS_OBJS =

414 COMMON_GENSTUBS_OBJS =

416 GENSTUBS_OBJS += $(COMMON_GENSTUBS_OBJS) $($(MACH)_GENSTUBS_OBJS)

418 #
419 #       DTrace and DTrace Providers
420 #
421 DTRACE_OBJS += dtrace.o dtrace_isa.o dtrace_asm.o

423 SDT_OBJS += sdt_subr.o

425 PROFILE_OBJS += profile.o

427 SYSTRACE_OBJS += systrace.o

429 LOCKSTAT_OBJS += lockstat.o

431 FASTTRAP_OBJS += fasttrap.o fasttrap_isa.o

433 DCPC_OBJS += dcpc.o

435 #
436 #       Driver (pseudo-driver) Modules
437 #
438 IPP_OBJS +=     ippctl.o

440 AUDIO_OBJS += audio_client.o audio_ddi.o audio_engine.o \
441             audio_fltdata.o audio_format.o audio_ctrl.o \
442             audio_grc3.o audio_output.o audio_input.o \
443             audio_oss.o audio_sun.o

445 AUDIOEMU10K_OBJS += audioemu10k.o

447 AUDIOENS_OBJS += audioens.o
```



```
449 AUDIOVIA823X_OBJS += audiovia823x.o

451 AUDIOVIA97_OBJS += audiovia97.o

453 AUDIO1575_OBJS += audio1575.o

455 AUDIO810_OBJS += audio810.o

457 AUDIOCMI_OBJS += audiocmi.o

459 AUDIOCMIHD_OBJS += audiocmihd.o

461 AUDIOHD_OBJS += audiohd.o

463 AUDIOIXP_OBJS += audioixp.o

465 AUDIOLS_OBJS += audiols.o

467 AUDIOP16X_OBJS += audiop16x.o

469 AUDIOPCI_OBJS += audiopci.o

471 AUDIOSOLO_OBJS += audiosolo.o

473 AUDIOTS_OBJS += audiots.o

475 AC97_OBJS += ac97.o ac97_ad.o ac97_alc.o ac97_cmi.o

477 BLKDEV_OBJS += blkdev.o

479 CARDBUS_OBJS += cardbus.o cardbus_hp.o cardbus_cfg.o

481 CONSKBD_OBJS += conskbd.o

483 CONSMS_OBJS +=  consms.o

485 OLDPTY_OBJS +=  tty_ptyconf.o

487 PTC_OBJS +=     tty_pty.o

489 PTSL_OBJS +=    tty_pts.o

491 PTM_OBJS +=     ptm.o

493 MII_OBJS +=     mii.o mii_cicada.o mii_natsemi.o mii_intel.o mii_qualsemi.o \
494                 mii_marvell.o mii_realtek.o mii_other.o

496 PTS_OBJS +=     pts.o

498 PTY_OBJS +=     ptms_conf.o

500 SAD_OBJS +=     sad.o

502 MD4_OBJS +=     md4.o md4_mod.o

504 MD5_OBJS +=     md5.o md5_mod.o

506 SHA1_OBJS +=    sha1.o sha1_mod.o

508 SHA2_OBJS +=    sha2.o sha2_mod.o

510 IPGPC_OBJS +=   classifierddi.o classifier.o filters.o trie.o table.o \
511                 ba_table.o

513 DSCPMK_OBJS +=  dscpmk.o dscpmkddi.o
```

```
515 DLCOSMK_OBJS += dlcosmk.o dlcosmkddi.o

517 FLOWACCT_OBJS +=        flowacctddi.o flowacct.o

519 TOKENMT_OBJS += tokenmt.o tokenmtddi.o

521 TSWTCL_OBJS +=  tswtcl.o tswtclddi.o

523 ARP_OBJS +=     arpddi.o

525 ICMP_OBJS +=    icmpddi.o

527 ICMP6_OBJS +=   icmp6ddi.o

529 RTS_OBJS +=     rtsddi.o

531 IP_ICMP_OBJS =  icmp.o icmp_opt_data.o
532 IP_RTS_OBJS =   rts.o rts_opt_data.o
533 IP_TCP_OBJS =   tcp.o tcp_fusion.o tcp_opt_data.o tcp_sack.o tcp_stats.o \
534                 tcp_misc.o tcp_timers.o tcp_time_wait.o tcp_tpi.o tcp_output.o \
535                 tcp_input.o tcp_socket.o tcp_bind.o tcp_cluster.o tcp_tunables.o
536 IP_UDP_OBJS =   udp.o udp_opt_data.o udp_tunables.o udp_stats.o
537 IP_SCTP_OBJS =  sctp.o sctp_opt_data.o sctp_output.o \
538                 sctp_init.o sctp_input.o sctp_cookie.o \
539                 sctp_conn.o sctp_error.o sctp_snmp.o \
540                 sctp_tunables.o sctp_shutdown.o sctp_common.o \
541                 sctp_timer.o sctp_heartbeat.o sctp_hash.o \
542                 sctp_bind.o sctp_notify.o sctp_asconf.o \
543                 sctp_addr.o tn_ipopt.o tnet.o ip_netinfo.o \
544                 sctp_misc.o
545 IP_ILB_OBJS =   ilb.o ilb_nat.o ilb_conn.o ilb_alg_hash.o ilb_alg_rr.o

547 IP_OBJS +=      igmp.o ipmp.o ip.o ip6.o ip6_asp.o ip6_if.o ip6_ire.o \
548                 ip6_rts.o ip_if.o ip_ire.o ip_listutils.o ip_mroute.o \
549                 ip_multi.o ip2mac.o ip_ndp.o ip_rts.o ip_srcid.o \
550                 ipddi.o ipdrop.o mi.o nd.o tunables.o optcom.o snmpcom.o \
551                 ipsec_loader.o spd.o ipclassifier.o inet_common.o ip_squeue.o \
552                 squeue.o ip_sadb.o ip_ftable.o proto_set.o radix.o ip_dummy.o \
553                 ip_helper_stream.o ip_tunables.o \
554                 ip_output.o ip_input.o ip6_input.o ip6_output.o ip_arp.o \
555                 conn_opt.o ip_attr.o ip_dce.o \
556                 $(IP_ICMP_OBJS) \
557                 $(IP_RTS_OBJS) \
558                 $(IP_TCP_OBJS) \
559                 $(IP_UDP_OBJS) \
560                 $(IP_SCTP_OBJS) \
561                 $(IP_ILB_OBJS)

563 IP6_OBJS +=     ip6ddi.o

565 HOOK_OBJS +=    hook.o

567 NETI_OBJS +=    neti_impl.o neti_mod.o neti_stack.o

569 KEYSOCK_OBJS += keysockddi.o keysock.o keysock_opt_data.o

571 IPNET_OBJS +=   ipnet.o ipnet_bpf.o

573 SPDSOCK_OBJS += spdsockddi.o spdsock.o spdsock_opt_data.o

575 IPSECESP_OBJS += ipsecespddi.o ipsecesp.o

577 IPSECAH_OBJS += ipsecahddi.o ipsecah.o sadb.o

579 SPPP_OBJS +=    sppp.o sppp_dlpi.o sppp_mod.o s_common.o
```



```
581 SPPPTUN_OBJS += sppptun.o sppptun_mod.o

583 SPPPASYN_OBJS += spppasyn.o spppasyn_mod.o

585 SPPPCOMP_OBJS += spppcomp.o spppcomp_mod.o deflate.o bsd-comp.o vjcompress.o \
586                 zlib.o

588 TCP_OBJS +=     tcpddi.o

590 TCP6_OBJS +=    tcp6ddi.o

592 NCA_OBJS +=     ncaddi.o

594 SDP_SOCK_MOD_OBJS += sockmod_sdp.o socksdp.o socksdpsubr.o

596 SCTP_SOCK_MOD_OBJS += sockmod_sctp.o socksctp.o socksctpsubr.o

598 PFP_SOCK_MOD_OBJS += sockmod_pfp.o

600 RDS_SOCK_MOD_OBJS += sockmod_rds.o

602 RDS_OBJS +=     rdsddi.o rdssubr.o rds_opt.o rds_ioctl.o

604 RDSIB_OBJS +=   rdsib.o rdsib_ib.o rdsib_cm.o rdsib_ep.o rdsib_buf.o \
605                 rdsib_debug.o rdsib_sc.o

607 RDSV3_OBJS +=   af_rds.o rdsv3_ddi.o bind.o loop.o threads.o connection.o \
608                 transport.o cong.o sysctl.o message.o rds_recv.o send.o \
609                 stats.o info.o page.o rdma_transport.o ib_ring.o ib_rdma.o \
610                 ib_recv.o ib.o ib_send.o ib_sysctl.o ib_stats.o ib_cm.o \
611                 rdsv3_sc.o rdsv3_debug.o rdsv3_impl.o rdma.o rdsv3_af_thr.o

613 ISER_OBJS +=    iser.o iser_cm.o iser_cq.o iser_ib.o iser_idm.o \
614                 iser_resource.o iser_xfer.o

616 UDP_OBJS +=     udpddi.o

618 UDP6_OBJS +=    udp6ddi.o

620 SY_OBJS +=      gentty.o

622 TCO_OBJS +=     ticots.o

624 TCOO_OBJS +=    ticotsord.o

626 TCL_OBJS +=     ticlts.o

628 TL_OBJS +=      tl.o

630 DUMP_OBJS +=    dump.o

632 BPF_OBJS +=     bpf.o bpf_filter.o bpf_mod.o bpf_dlt.o bpf_mac.o

634 CLONE_OBJS +=   clone.o

636 CN_OBJS +=      cons.o

638 DLD_OBJS +=     dld_drv.o dld_proto.o dld_str.o dld_flow.o

640 DLS_OBJS +=     dls.o dls_link.o dls_mod.o dls_stat.o dls_mgmt.o

642 GLD_OBJS +=     gld.o gldutil.o

644 MAC_OBJS +=     mac.o  mac_bcast.o mac_client.o mac_datapath_setup.o mac_flow.o
645                 mac_hio.o mac_mod.o mac_ndd.o mac_provider.o mac_sched.o \
646                 mac_protect.o mac_soft_ring.o mac_stat.o mac_util.o
```

```
MAC_6TO4_OBJS +=        mac_6to4.o

MAC_ETHER_OBJS +=       mac_ether.o

MAC_IPV4_OBJS +=        mac_ipv4.o

MAC_IPV6_OBJS +=        mac_ipv6.o

MAC_WIFI_OBJS +=        mac_wifi.o

MAC_IB_OBJS +=          mac_ib.o

IPTUN_OBJS +=   iptun_dev.o iptun_ctl.o iptun.o

AGGR_OBJS +=    aggr_dev.o aggr_ctl.o aggr_grp.o aggr_port.o \
                aggr_send.o aggr_recv.o aggr_lacp.o

SOFTMAC_OBJS += softmac_main.o softmac_ctl.o softmac_capab.o \
                softmac_dev.o softmac_stat.o softmac_pkt.o softmac_fp.o

NET80211_OBJS += net80211.o net80211_proto.o net80211_input.o \
                 net80211_output.o net80211_node.o net80211_crypto.o \
                 net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
                 net80211_crypto_tkip.o net80211_crypto_ccmp.o  \
                 net80211_ht.o

VNIC_OBJS +=    vnic_ctl.o vnic_dev.o

SIMNET_OBJS +=  simnet.o

IB_OBJS +=      ibnex.o ibnex_ioctl.o ibnex_hca.o

IBCM_OBJS +=    ibcm_impl.o ibcm_sm.o ibcm_ti.o ibcm_utils.o ibcm_path.o \
                ibcm_arp.o ibcm_arp_link.o

IBDM_OBJS +=    ibdm.o

IBDMA_OBJS +=   ibdma.o

IBMF_OBJS +=    ibmf.o ibmf_impl.o ibmf_dr.o ibmf_wqe.o ibmf_ud_dest.o ibmf_mod.
                ibmf_send.o ibmf_recv.o ibmf_handlers.o ibmf_trans.o \
                ibmf_timers.o ibmf_msg.o ibmf_utils.o ibmf_rmpp.o \
                ibmf_saa.o ibmf_saa_impl.o ibmf_saa_utils.o ibmf_saa_events.o

IBTL_OBJS +=    ibtl_impl.o ibtl_util.o ibtl_mem.o ibtl_handlers.o ibtl_qp.o \
                ibtl_cq.o ibtl_wr.o ibtl_hca.o ibtl_chan.o ibtl_cm.o \
                ibtl_mcg.o ibtl_ibnex.o ibtl_srq.o ibtl_part.o

TAVOR_OBJS +=   tavor.o tavor_agents.o tavor_cfg.o tavor_ci.o tavor_cmd.o \
                tavor_cq.o tavor_event.o tavor_ioctl.o tavor_misc.o \
                tavor_mr.o tavor_qp.o tavor_qpmod.o tavor_rsrc.o \
                tavor_srq.o tavor_stats.o tavor_umap.o tavor_wr.o

HERMON_OBJS +=  hermon.o hermon_agents.o hermon_cfg.o hermon_ci.o hermon_cmd.o \
                hermon_cq.o hermon_event.o hermon_ioctl.o hermon_misc.o \
                hermon_mr.o hermon_qp.o hermon_qpmod.o hermon_rsrc.o \
                hermon_srq.o hermon_stats.o hermon_umap.o hermon_wr.o \
                hermon_fcoib.o hermon_fm.o

DAPLT_OBJS +=   daplt.o

SOL_OFS_OBJS += sol_cma.o sol_ib_cma.o sol_uobj.o \
                sol_ofs_debug_util.o sol_ofs_gen_util.o \
                sol_kverbs.o
```



```
SOL_UCMA_OBJS +=        sol_ucma.o

SOL_UVERBS_OBJS +=      sol_uverbs.o sol_uverbs_comp.o sol_uverbs_event.o \
                        sol_uverbs_hca.o sol_uverbs_qp.o

SOL_UMAD_OBJS += sol_umad.o

KSTAT_OBJS +=   kstat.o

KSYMS_OBJS +=   ksyms.o

INSTANCE_OBJS += inst_sync.o

IWSCN_OBJS +=   iwscons.o

LOFI_OBJS +=    lofi.o LzmaDec.o

FSSNAP_OBJS +=  fssnap.o

FSSNAPIF_OBJS += fssnap_if.o

MM_OBJS +=      mem.o

PHYSMEM_OBJS += physmem.o

OPTIONS_OBJS += options.o

WINLOCK_OBJS += winlockio.o

PM_OBJS +=      pm.o
SRN_OBJS +=     srn.o

PSEUDO_OBJS +=  pseudonex.o

RAMDISK_OBJS += ramdisk.o

LLC1_OBJS += llc1.o

USBKBM_OBJS += usbkbm.o

USBWCM_OBJS += usbwcm.o

BOFI_OBJS += bofi.o

HID_OBJS += hid.o

HWA_RC_OBJS += hwarc.o

USBSKEL_OBJS += usbskel.o

USBVC_OBJS += usbvc.o usbvc_v4l2.o

HIDPARSER_OBJS += hidparser.o

USB_AC_OBJS += usb_ac.o

USB_AS_OBJS += usb_as.o

USB_AH_OBJS += usb_ah.o

USBMS_OBJS += usbms.o

USBPRN_OBJS += usbprn.o

UGEN_OBJS += ugen.o
```

```
779 USBSER_OBJS += usbser.o usbser_rseq.o

781 USBSACM_OBJS += usbsacm.o

783 USBSER_KEYSPAN_OBJS += usbser_keyspan.o keyspan_dsd.o keyspan_pipe.o

785 USBS49_FW_OBJS += keyspan_49fw.o

787 USBSPRL_OBJS += usbser_pl2303.o pl2303_dsd.o

789 WUSB_CA_OBJS += wusb_ca.o

791 USBFTDI_OBJS += usbser_uftdi.o uftdi_dsd.o

793 USBECM_OBJS += usbecm.o

795 WC_OBJS += wscons.o vcons.o

797 VCONS_CONF_OBJS += vcons_conf.o

799 SCSI_OBJS +=    scsi_capabilities.o scsi_confsubr.o scsi_control.o \
800                 scsi_data.o scsi_fm.o scsi_hba.o scsi_reset_notify.o \
801                 scsi_resource.o scsi_subr.o scsi_transport.o scsi_watch.o \
802                 smp_transport.o

804 SCSI_VHCI_OBJS +=               scsi_vhci.o mpapi_impl.o scsi_vhci_tpgs.o

806 SCSI_VHCI_F_SYM_OBJS +=         sym.o

808 SCSI_VHCI_F_TPGS_OBJS +=        tpgs.o

810 SCSI_VHCI_F_ASYM_SUN_OBJS +=    asym_sun.o

812 SCSI_VHCI_F_SYM_HDS_OBJS +=     sym_hds.o

814 SCSI_VHCI_F_TAPE_OBJS +=        tape.o

816 SCSI_VHCI_F_TPGS_TAPE_OBJS +=   tpgs_tape.o

818 SGEN_OBJS +=    sgen.o

820 SMP_OBJS +=     smp.o

822 SATA_OBJS +=    sata.o

824 USBA_OBJS +=    hcdi.o  usba.o  usbai.o hubdi.o parser.o genconsole.o \
825                 usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
826                 usba_devdb.o usba10_calls.o usba_ugen.o whcdi.o wa.o
827 USBA_WITHOUT_WUSB_OBJS +=       hcdi.o  usba.o  usbai.o hubdi.o parser.o gencons
828                 usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
829                 usba_devdb.o usba10_calls.o usba_ugen.o

831 USBA10_OBJS +=  usba10.o

833 RSM_OBJS +=     rsm.o   rsmka_pathmanager.o     rsmka_util.o

835 RSMOPS_OBJS +=  rsmops.o

837 S1394_OBJS +=   t1394.o t1394_errmsg.o s1394.o s1394_addr.o s1394_asynch.o \
838                 s1394_bus_reset.o s1394_cmp.o s1394_csr.o s1394_dev_disc.o \
839                 s1394_fa.o s1394_fcp.o \
840                 s1394_hotplug.o s1394_isoch.o s1394_misc.o h1394.o nx1394.o

842 HCI1394_OBJS += hci1394.o hci1394_async.o hci1394_attach.o hci1394_buf.o \
843                 hci1394_csr.o hci1394_detach.o hci1394_extern.o \
844                 hci1394_ioctl.o hci1394_isoch.o hci1394_isr.o \
```



```
845                 hci1394_ixl_comp.o hci1394_ixl_isr.o hci1394_ixl_misc.o \
846                 hci1394_ixl_update.o hci1394_misc.o hci1394_ohci.o \
847                 hci1394_q.o hci1394_s1394if.o hci1394_tlabel.o \
848                 hci1394_tlist.o hci1394_vendor.o

850 AV1394_OBJS +=  av1394.o av1394_as.o av1394_async.o av1394_cfgrom.o \
851                 av1394_cmp.o av1394_fcp.o av1394_isoch.o av1394_isoch_chan.o \
852                 av1394_isoch_recv.o av1394_isoch_xmit.o av1394_list.o \
853                 av1394_queue.o

855 DCAM1394_OBJS += dcam.o dcam_frame.o dcam_param.o dcam_reg.o \
856                 dcam_ring_buff.o

858 SCSA1394_OBJS += hba.o sbp2_driver.o sbp2_bus.o

860 SBP2_OBJS += cfgrom.o sbp2.o

862 PMODEM_OBJS += pmodem.o pmodem_cis.o cis.o cis_callout.o cis_handlers.o cis_para

864 DSW_OBJS +=     dsw.o dsw_dev.o ii_tree.o

866 NCALL_OBJS +=   ncall.o \
867                 ncall_stub.o

869 RDC_OBJS +=     rdc.o \
870                 rdc_dev.o \
871                 rdc_io.o \
872                 rdc_clnt.o \
873                 rdc_prot_xdr.o \
874                 rdc_svc.o \
875                 rdc_bitmap.o \
876                 rdc_health.o \
877                 rdc_subr.o \
878                 rdc_diskq.o

880 RDCSRV_OBJS +=  rdcsrv.o

882 RDCSTUB_OBJS += rdc_stub.o

884 SDBC_OBJS +=    sd_bcache.o \
885                 sd_bio.o \
886                 sd_conf.o \
887                 sd_ft.o \
888                 sd_hash.o \
889                 sd_io.o \
890                 sd_misc.o \
891                 sd_pcu.o \
892                 sd_tdaemon.o \
893                 sd_trace.o \
894                 sd_iob_impl0.o \
895                 sd_iob_impl1.o \
896                 sd_iob_impl2.o \
897                 sd_iob_impl3.o \
898                 sd_iob_impl4.o \
899                 sd_iob_impl5.o \
900                 sd_iob_impl6.o \
901                 sd_iob_impl7.o \
902                 safestore.o \
903                 safestore_ram.o

905 NSCTL_OBJS +=   nsctl.o \
906                 nsc_cache.o \
907                 nsc_disk.o \
908                 nsc_dev.o \
909                 nsc_freeze.o \
910                 nsc_gen.o \
```

```
911                 nsc_mem.o \
912                 nsc_ncallio.o \
913                 nsc_power.o \
914                 nsc_resv.o \
915                 nsc_rmspin.o \
916                 nsc_solaris.o \
917                 nsc_trap.o \
918                 nsc_list.o
919 UNISTAT_OBJS += spuni.o \
920                 spcs_s_k.o

922 NSKERN_OBJS +=  nsc_ddi.o \
923                 nsc_proc.o \
924                 nsc_raw.o \
925                 nsc_thread.o \
926                 nskernd.o

928 SV_OBJS +=      sv.o

930 PMCS_OBJS += pmcs_attach.o pmcs_ds.o pmcs_intr.o pmcs_nvram.o pmcs_sata.o \
931                 pmcs_scsa.o pmcs_smhba.o pmcs_subr.o pmcs_fwlog.o

933 PMCS8001FW_C_OBJS +=    pmcs_fw_hdr.o
934 PMCS8001FW_OBJS +=              $(PMCS8001FW_C_OBJS) SPCBoot.o ila.o firmware.o

936 #
937 #       Build up defines and paths.

939 ST_OBJS +=      st.o    st_conf.o

941 EMLXS_OBJS +=   emlxs_clock.o emlxs_dfc.o emlxs_dhchap.o emlxs_diag.o \
942                 emlxs_download.o emlxs_dump.o emlxs_els.o emlxs_event.o \
943                 emlxs_fcf.o emlxs_fcp.o emlxs_fct.o emlxs_hba.o emlxs_ip.o \
944                 emlxs_mbox.o emlxs_mem.o emlxs_msg.o emlxs_node.o \
945                 emlxs_pkt.o emlxs_sli3.o emlxs_sli4.o emlxs_solaris.o \
946                 emlxs_thread.o

948 EMLXS_FW_OBJS +=        emlxs_fw.o

950 OCE_OBJS +=     oce_buf.o oce_fm.o oce_gld.o oce_hw.o oce_intr.o oce_main.o \
951                 oce_mbx.o oce_mq.o oce_queue.o oce_rx.o oce_stat.o oce_tx.o \
952                 oce_utils.o

954 FCT_OBJS += discovery.o fct.o

956 QLT_OBJS += 2400.o 2500.o 8100.o qlt.o qlt_dma.o

958 SRPT_OBJS += srpt_mod.o srpt_ch.o srpt_cm.o srpt_ioc.o srpt_stp.o

960 FCOE_OBJS += fcoe.o fcoe_eth.o fcoe_fc.o

962 FCOET_OBJS += fcoet.o fcoet_eth.o fcoet_fc.o

964 FCOEI_OBJS += fcoei.o fcoei_eth.o fcoei_lv.o

966 ISCSIT_SHARED_OBJS += \
967                 iscsit_common.o

969 ISCSIT_OBJS +=  $(ISCSIT_SHARED_OBJS) \
970                 iscsit.o iscsit_tgt.o iscsit_sess.o iscsit_login.o \
971                 iscsit_text.o iscsit_isns.o iscsit_radiusauth.o \
972                 iscsit_radiuspacket.o iscsit_auth.o iscsit_authclient.o

974 PPPT_OBJS +=    alua_ic_if.o pppt.o pppt_msg.o pppt_tgt.o

976 STMF_OBJS += lun_map.o stmf.o
```



```
978 STMF_SBD_OBJS += sbd.o sbd_scsi.o sbd_pgr.o sbd_zvol.o

980 SYSMSG_OBJS +=  sysmsg.o

982 SES_OBJS +=     ses.o ses_sen.o ses_safte.o ses_ses.o

984 TNF_OBJS +=     tnf_buf.o       tnf_trace.o     tnf_writer.o    trace_init.o \
985                 trace_funcs.o   tnf_probe.o     tnf.o

987 LOGINDMUX_OBJS += logindmux.o

989 DEVINFO_OBJS += devinfo.o

991 DEVPOLL_OBJS += devpoll.o

993 DEVPOOL_OBJS += devpool.o

995 I8042_OBJS +=   i8042.o

997 KB8042_OBJS +=  \
998                 at_keyprocess.o \
999                 kb8042.o        \
1000                kb8042_keytables.o

1002 MOUSE8042_OBJS += mouse8042.o

1004 FDC_OBJS +=     fdc.o

1006 ASY_OBJS +=     asy.o

1008 ECPP_OBJS +=    ecpp.o

1010 VUIDM3P_OBJS += vuidmice.o vuidm3p.o

1012 VUIDM4P_OBJS += vuidmice.o vuidm4p.o

1014 VUIDM5P_OBJS += vuidmice.o vuidm5p.o

1016 VUIDPS2_OBJS += vuidmice.o vuidps2.o

1018 HPCSVC_OBJS += hpcsvc.o

1020 PCIE_MISC_OBJS += pcie.o pcie_fault.o pcie_hp.o pciehpc.o pcishpc.o pcie_pwr.o p

1022 PCIHPNEXUS_OBJS += pcihp.o

1024 OPENEEPR_OBJS += openprom.o

1026 RANDOM_OBJS += random.o

1028 PSHOT_OBJS += pshot.o

1030 GEN_DRV_OBJS += gen_drv.o

1032 TCLIENT_OBJS += tclient.o

1034 TPHCI_OBJS += tphci.o

1036 TVHCI_OBJS += tvhci.o

1038 EMUL64_OBJS += emul64.o emul64_bsd.o

1040 FCP_OBJS += fcp.o

1042 FCIP_OBJS += fcip.o
```

```
1044 FCSM_OBJS += fcsm.o

1046 FCTL_OBJS += fctl.o

1048 FP_OBJS += fp.o

1050 QLC_OBJS += ql_api.o ql_debug.o ql_hba_fru.o ql_init.o ql_iocb.o ql_ioctl.o \
1051         ql_isr.o ql_mbx.o ql_nx.o ql_xioctl.o ql_fw_table.o

1053 QLC_FW_2200_OBJS += ql_fw_2200.o

1055 QLC_FW_2300_OBJS += ql_fw_2300.o

1057 QLC_FW_2400_OBJS += ql_fw_2400.o

1059 QLC_FW_2500_OBJS += ql_fw_2500.o

1061 QLC_FW_6322_OBJS += ql_fw_6322.o

1063 QLC_FW_8100_OBJS += ql_fw_8100.o

1065 QLGE_OBJS += qlge.o qlge_dbg.o qlge_flash.o qlge_fm.o qlge_gld.o qlge_mpi.o

1067 ZCONS_OBJS += zcons.o

1069 NV_SATA_OBJS += nv_sata.o

1071 SI3124_OBJS += si3124.o

1073 AHCI_OBJS += ahci.o

1075 PCIIDE_OBJS += pci-ide.o

1077 PCEPP_OBJS += pcepp.o

1079 CPC_OBJS += cpc.o

1081 CPUID_OBJS += cpuid_drv.o

1083 SYSEVENT_OBJS += sysevent.o

1085 BL_OBJS += bl.o

1087 DRM_OBJS += drm_sunmod.o drm_kstat.o drm_agpsupport.o \
1088             drm_auth.o drm_bufs.o drm_context.o drm_dma.o \
1089             drm_drawable.o drm_drv.o drm_fops.o drm_ioctl.o drm_irq.o \
1090             drm_lock.o drm_memory.o drm_msg.o drm_pci.o drm_scatter.o \
1091             drm_cache.o drm_gem.o drm_mm.o ati_pcigart.o

1093 FM_OBJS += devfm.o devfm_machdep.o

1095 RTLS_OBJS +=    rtls.o

1097 #
1098 #                       exec modules
1099 #
1100 AOUTEXEC_OBJS +=aout.o

1102 ELFEXEC_OBJS += elf.o elf_notes.o old_notes.o

1104 INTPEXEC_OBJS +=intp.o

1106 SHBINEXEC_OBJS +=shbin.o

1108 JAVAEXEC_OBJS +=java.o
```



```
1110 #
1111 #                       file system modules
1112 #
1113 AUTOFS_OBJS +=  auto_vfsops.o auto_vnops.o auto_subr.o auto_xdr.o auto_sys.o

1115 CACHEFS_OBJS += cachefs_cnode.o         cachefs_cod.o \
1116                 cachefs_dir.o           cachefs_dlog.o  cachefs_filegrp.o \
1117                 cachefs_fscache.o       cachefs_ioctl.o cachefs_log.o \
1118                 cachefs_module.o \
1119                 cachefs_noopc.o         cachefs_resource.o \
1120                 cachefs_strict.o \
1121                 cachefs_subr.o          cachefs_vfsops.o \
1122                 cachefs_vnops.o

1124 DCFS_OBJS +=    dc_vnops.o

1126 DEVFS_OBJS +=   devfs_subr.o    devfs_vfsops.o  devfs_vnops.o

1128 DEV_OBJS  +=    sdev_subr.o     sdev_vfsops.o   sdev_vnops.o    \
1129                 sdev_ptsops.o   sdev_zvolops.o  sdev_comm.o     \
1130                 sdev_profile.o  sdev_ncache.o   sdev_netops.o   \
1131                 sdev_ipnetops.o \
1132                 sdev_vtops.o

1134 CTFS_OBJS +=    ctfs_all.o ctfs_cdir.o ctfs_ctl.o ctfs_event.o \
1135                 ctfs_latest.o ctfs_root.o ctfs_sym.o ctfs_tdir.o ctfs_tmpl.o

1137 OBJFS_OBJS +=   objfs_vfs.o     objfs_root.o    objfs_common.o \
1138                 objfs_odir.o    objfs_data.o

1140 FDFS_OBJS +=    fdops.o

1142 FIFO_OBJS +=    fifosubr.o      fifovnops.o

1144 PIPE_OBJS +=    pipe.o

1146 HSFS_OBJS +=    hsfs_node.o     hsfs_subr.o     hsfs_vfsops.o   hsfs_vnops.o \
1147                 hsfs_susp.o     hsfs_rrip.o     hsfs_susp_subr.o

1149 LOFS_OBJS +=    lofs_subr.o     lofs_vfsops.o   lofs_vnops.o

1151 NAMEFS_OBJS +=  namevfs.o       namevno.o

1153 NFS_OBJS +=     nfs_client.o    nfs_common.o    nfs_dump.o \
1154                 nfs_subr.o      nfs_vfsops.o    nfs_vnops.o \
1155                 nfs_xdr.o       nfs_sys.o       nfs_strerror.o \
1156                 nfs3_vfsops.o   nfs3_vnops.o    nfs3_xdr.o \
1157                 nfs_acl_vnops.o nfs_acl_xdr.o   nfs4_vfsops.o \
1158                 nfs4_vnops.o    nfs4_xdr.o      nfs4_idmap.o \
1159                 nfs4_shadow.o   nfs4_subr.o \
1160                 nfs4_attr.o     nfs4_rnode.o    nfs4_client.o \
1161                 nfs4_acache.o   nfs4_common.o   nfs4_client_state.o \
1162                 nfs4_callback.o nfs4_recovery.o nfs4_client_secinfo.o \
1163                 nfs4_client_debug.o     nfs_stats.o \
1164                 nfs4_acl.o      nfs4_stub_vnops.o       nfs_cmd.o

1166 NFSSRV_OBJS +=  nfs_server.o    nfs_srv.o       nfs3_srv.o \
1167                 nfs_acl_srv.o   nfs_auth.o      nfs_auth_xdr.o \
1168                 nfs_export.o    nfs_log.o       nfs_log_xdr.o \
1169                 nfs4_srv.o      nfs4_state.o    nfs4_srv_attr.o \
1170                 nfs4_srv_ns.o   nfs4_db.o       nfs4_srv_deleg.o \
1171                 nfs4_deleg_ops.o nfs4_srv_readdir.o nfs4_dispatch.o

1173 SMBSRV_SHARED_OBJS += \
1174                 smb_inet.o \
```

```
1175                    smb_match.o \
1176                    smb_msgbuf.o \
1177                    smb_oem.o \
1178                    smb_string.o \
1179                    smb_utf8.o \
1180                    smb_door_legacy.o \
1181                    smb_xdr.o \
1182                    smb_token.o \
1183                    smb_token_xdr.o \
1184                    smb_sid.o \
1185                    smb_native.o \
1186                    smb_netbios_util.o

1188 SMBSRV_OBJS +=     $(SMBSRV_SHARED_OBJS)                   \
1189                    smb_acl.o                               \
1190                    smb_alloc.o                             \
1191                    smb_close.o                             \
1192                    smb_common_open.o                       \
1193                    smb_common_transact.o                   \
1194                    smb_create.o                            \
1195                    smb_delete.o                            \
1196                    smb_directory.o                         \
1197                    smb_dispatch.o                          \
1198                    smb_echo.o                              \
1199                    smb_fem.o                               \
1200                    smb_find.o                              \
1201                    smb_flush.o                             \
1202                    smb_fsinfo.o                            \
1203                    smb_fsops.o                             \
1204                    smb_init.o                              \
1205                    smb_kdoor.o                             \
1206                    smb_kshare.o                            \
1207                    smb_kutil.o                             \
1208                    smb_lock.o                              \
1209                    smb_lock_byte_range.o                   \
1210                    smb_locking_andx.o                      \
1211                    smb_logoff_andx.o                       \
1212                    smb_mangle_name.o                       \
1213                    smb_mbuf_marshaling.o                   \
1214                    smb_mbuf_util.o                         \
1215                    smb_negotiate.o                         \
1216                    smb_net.o                               \
1217                    smb_node.o                              \
1218                    smb_nt_cancel.o                         \
1219                    smb_nt_create_andx.o                    \
1220                    smb_nt_transact_create.o                \
1221                    smb_nt_transact_ioctl.o                 \
1222                    smb_nt_transact_notify_change.o         \
1223                    smb_nt_transact_quota.o                 \
1224                    smb_nt_transact_security.o              \
1225                    smb_odir.o                              \
1226                    smb_ofile.o                             \
1227                    smb_open_andx.o                         \
1228                    smb_opipe.o                             \
1229                    smb_oplock.o                            \
1230                    smb_pathname.o                          \
1231                    smb_print.o                             \
1232                    smb_process_exit.o                      \
1233                    smb_query_fileinfo.o                    \
1234                    smb_read.o                              \
1235                    smb_rename.o                            \
1236                    smb_sd.o                                \
1237                    smb_seek.o                              \
1238                    smb_server.o                            \
1239                    smb_session.o                           \
1240                    smb_session_setup_andx.o                \
```



```
1241                    smb_set_fileinfo.o                      \
1242                    smb_signing.o                           \
1243                    smb_tree.o                              \
1244                    smb_trans2_create_directory.o           \
1245                    smb_trans2_dfs.o                        \
1246                    smb_trans2_find.o                       \
1247                    smb_tree_connect.o                      \
1248                    smb_unlock_byte_range.o                 \
1249                    smb_user.o                              \
1250                    smb_vfs.o                               \
1251                    smb_vops.o                              \
1252                    smb_vss.o                               \
1253                    smb_write.o                             \
1254                    smb_write_raw.o

1256 PCFS_OBJS +=      pc_alloc.o      pc_dir.o        pc_node.o       pc_subr.o \
1257                    pc_vfsops.o     pc_vnops.o

1259 PROC_OBJS +=      prcontrol.o     prioctl.o       prsubr.o        prusrio.o \
1260                    prvfsops.o      prvnops.o

1262 MNTFS_OBJS +=     mntvfsops.o     mntvnops.o

1264 SHAREFS_OBJS +=   sharetab.o      sharefs_vfsops.o        sharefs_vnops.o

1266 SPEC_OBJS +=      specsubr.o      specvfsops.o    specvnops.o

1268 SOCK_OBJS +=      socksubr.o      sockvfsops.o    sockparams.o    \
1269                    socksyscalls.o  socktpi.o       sockstr.o \
1270                    sockcommon_vnops.o      sockcommon_subr.o \
1271                    sockcommon_sops.o       sockcommon.o    \
1272                    sock_notsupp.o  socknotify.o \
1273                    nl7c.o          nl7curi.o       nl7chttp.o      nl7clogd.o \
1274                    nl7cnca.o       sodirect.o      sockfilter.o

1276 TMPFS_OBJS +=     tmp_dir.o       tmp_subr.o      tmp_tnode.o     tmp_vfsops.o \
1277                    tmp_vnops.o

1279 UDFS_OBJS +=      udf_alloc.o     udf_bmap.o      udf_dir.o       \
1280                    udf_inode.o     udf_subr.o      udf_vfsops.o    \
1281                    udf_vnops.o

1283 UFS_OBJS +=       ufs_alloc.o     ufs_bmap.o      ufs_dir.o       ufs_xattr.o \
1284                    ufs_inode.o     ufs_subr.o      ufs_tables.o    ufs_vfsops.o \
1285                    ufs_vnops.o     quota.o         quotacalls.o    quota_ufs.o \
1286                    ufs_filio.o     ufs_lockfs.o    ufs_thread.o    ufs_trans.o \
1287                    ufs_acl.o       ufs_panic.o     ufs_directio.o  ufs_log.o \
1288                    ufs_extvnops.o  ufs_snap.o      lufs.o          lufs_thread.o \
1289                    lufs_log.o      lufs_map.o      lufs_top.o      lufs_debug.o
1290 VSCAN_OBJS +=     vscan_drv.o     vscan_svc.o vscan_door.o

1292 NSMB_OBJS +=      smb_conn.o      smb_dev.o       smb_iod.o       smb_pass.o \
1293                    smb_rq.o        smb_sign.o      smb_smb.o       smb_subrs.o \
1294                    smb_time.o      smb_tran.o      smb_trantcp.o   smb_usr.o \
1295                    subr_mchain.o

1297 SMBFS_COMMON_OBJS += smbfs_ntacl.o
1298 SMBFS_OBJS +=     smbfs_vfsops.o  smbfs_vnops.o   smbfs_node.o    \
1299                    smbfs_acl.o     smbfs_client.o  smbfs_smb.o     \
1300                    smbfs_subr.o    smbfs_subr2.o   \
1301                    smbfs_rwlock.o  smbfs_xattr.o   \
1302                    $(SMBFS_COMMON_OBJS)


1305 #
1306 #                       LVM modules
```

```
1307 #
1308 MD_OBJS += md.o md_error.o md_ioctl.o md_mddb.o md_names.o \
1309         md_med.o md_rename.o md_subr.o

1311 MD_COMMON_OBJS = md_convert.o md_crc.o md_revchk.o

1313 MD_DERIVED_OBJS = metamed_xdr.o meta_basic_xdr.o

1315 SOFTPART_OBJS += sp.o sp_ioctl.o

1317 STRIPE_OBJS += stripe.o stripe_ioctl.o

1319 HOTSPARES_OBJS += hotspares.o

1321 RAID_OBJS += raid.o raid_ioctl.o raid_replay.o raid_resync.o raid_hotspare.o

1323 MIRROR_OBJS += mirror.o mirror_ioctl.o mirror_resync.o

1325 NOTIFY_OBJS += md_notify.o

1327 TRANS_OBJS += mdtrans.o trans_ioctl.o trans_log.o

1329 ZFS_COMMON_OBJS +=              \
1330         arc.o                   \
1331         bplist.o                \
1332         bpobj.o                 \
1333         bptree.o                \
1334         dbuf.o                  \
1335         ddt.o                   \
1336         ddt_zap.o               \
1337         dmu.o                   \
1338         dmu_diff.o              \
1339         dmu_send.o              \
1340         dmu_object.o            \
1341         dmu_objset.o            \
1342         dmu_traverse.o          \
1343         dmu_tx.o                \
1344         dnode.o                 \
1345         dnode_sync.o            \
1346         dsl_dir.o               \
1347         dsl_dataset.o           \
1348         dsl_deadlist.o          \
1349         dsl_destroy.o           \
1350         dsl_pool.o              \
1351         dsl_synctask.o          \
1352         dsl_userhold.o          \
1353         dmu_zfetch.o            \
1354         dsl_deleg.o             \
1355         dsl_prop.o              \
1356         dsl_scan.o              \
1357         zfeature.o              \
1358         gzip.o                  \
1359         lz4.o                   \
1360         lzjb.o                  \
1361         metaslab.o              \
1362         range_tree.o            \
1363         refcount.o              \
1364         rrwlock.o               \
1365         sa.o                    \
1366         sha256.o                \
1367         spa.o                   \
1368         spa_config.o            \
1369         spa_errlog.o            \
1370         spa_history.o           \
1371         spa_misc.o              \
1372         space_map.o             \
```



```
1373         space_reftree.o         \
1374         txg.o                   \
1375         uberblock.o             \
1376         unique.o                \
1377         vdev.o                  \
1378         vdev_cache.o            \
1379         vdev_file.o             \
1380         vdev_label.o            \
1381         vdev_mirror.o           \
1382         vdev_missing.o          \
1383         vdev_queue.o            \
1384         vdev_raidz.o            \
1385         vdev_root.o             \
1386         zap.o                   \
1387         zap_leaf.o              \
1388         zap_micro.o             \
1389         zfs_byteswap.o          \
1390         zfs_debug.o             \
1391         zfs_fm.o                \
1392         zfs_fuid.o              \
1393         zfs_sa.o                \
1394         zfs_znode.o             \
1395         zil.o                   \
1396         zio.o                   \
1397         zio_checksum.o          \
1398         zio_compress.o          \
1399         zio_inject.o            \
1400         zle.o                   \
1401         zrlock.o

1403 ZFS_SHARED_OBJS +=              \
1404         zfeature_common.o       \
1405         zfs_comutil.o           \
1406         zfs_deleg.o             \
1407         zfs_fletcher.o          \
1408         zfs_namecheck.o         \
1409         zfs_prop.o              \
1410         zpool_prop.o            \
1411         zprop_common.o

1413 ZFS_OBJS +=                     \
1414         $(ZFS_COMMON_OBJS)      \
1415         $(ZFS_SHARED_OBJS)      \
1416         vdev_disk.o             \
1417         zfs_acl.o               \
1418         zfs_ctldir.o            \
1419         zfs_dir.o               \
1420         zfs_ioctl.o             \
1421         zfs_log.o               \
1422         zfs_onexit.o            \
1423         zfs_replay.o            \
1424         zfs_rlock.o             \
1425         zfs_vfsops.o            \
1426         zfs_vnops.o             \
1427         zvol.o

1429 ZUT_OBJS +=                     \
1430         zut.o

1432 #
1433 #                       streams modules
1434 #
1435 BUFMOD_OBJS     +=      bufmod.o

1437 CONNLD_OBJS +=  connld.o
```

```
1439 DEDUMP_OBJS +=  dedump.o

1441 DRCOMPAT_OBJS +=        drcompat.o

1443 LDLINUX_OBJS += ldlinux.o

1445 LDTERM_OBJS +=  ldterm.o uwidth.o

1447 PCKT_OBJS +=    pckt.o

1449 PFMOD_OBJS +=   pfmod.o

1451 PTEM_OBJS +=    ptem.o

1453 REDIRMOD_OBJS += strredirm.o

1455 TIMOD_OBJS +=   timod.o

1457 TIRDWR_OBJS +=  tirdwr.o

1459 TTCOMPAT_OBJS +=ttcompat.o

1461 LOG_OBJS +=     log.o

1463 PIPEMOD_OBJS += pipemod.o

1465 RPCMOD_OBJS +=  rpcmod.o        clnt_cots.o     clnt_clts.o \
1466                 clnt_gen.o      clnt_perr.o     mt_rpcinit.o    rpc_calmsg.o \
1467                 rpc_prot.o      rpc_sztypes.o   rpc_subr.o      rpcb_prot.o \
1468                 svc.o           svc_clts.o      svc_gen.o       svc_cots.o \
1469                 rpcsys.o        xdr_sizeof.o    clnt_rdma.o     svc_rdma.o \
1470                 xdr_rdma.o      rdma_subr.o     xdrrdma_sizeof.o

1472 KLMMOD_OBJS +=  klmmod.o \
1473                 nlm_impl.o \
1474                 nlm_rpc_handle.o \
1475                 nlm_dispatch.o \
1476                 nlm_rpc_svc.o \
1477                 nlm_client.o \
1478                 nlm_service.o \
1479                 nlm_prot_clnt.o \
1480                 nlm_prot_xdr.o \
1481                 nlm_rpc_clnt.o \
1482                 nsm_addr_clnt.o \
1483                 nsm_addr_xdr.o \
1484                 sm_inter_clnt.o \
1485                 sm_inter_xdr.o

1487 KLMOPS_OBJS +=  klmops.o

1489 TLIMOD_OBJS +=  tlimod.o        t_kalloc.o      t_kbind.o       t_kclose.o \
1490                 t_kconnect.o    t_kfree.o       t_kgtstate.o    t_kopen.o \
1491                 t_krcvudat.o    t_ksndudat.o    t_kspoll.o      t_kunbind.o \
1492                 t_kutil.o

1494 RLMOD_OBJS += rlmod.o

1496 TELMOD_OBJS += telmod.o

1498 CRYPTMOD_OBJS += cryptmod.o

1500 KB_OBJS +=      kbd.o           keytables.o

1502 #
1503 #                       ID mapping module
1504 #
```



```
1505 IDMAP_OBJS +=   idmap_mod.o     idmap_kapi.o    idmap_xdr.o     idmap_cache.o

1507 #
1508 #                       scheduling class modules
1509 #
1510 SDC_OBJS +=             sysdc.o

1512 RT_OBJS +=              rt.o
1513 RT_DPTBL_OBJS +=        rt_dptbl.o

1515 TS_OBJS +=              ts.o
1516 TS_DPTBL_OBJS +=        ts_dptbl.o

1518 IA_OBJS +=              ia.o

1520 FSS_OBJS +=             fss.o

1522 FX_OBJS +=              fx.o
1523 FX_DPTBL_OBJS +=        fx_dptbl.o

1525 #
1526 #                       Inter-Process Communication (IPC) modules
1527 #
1528 IPC_OBJS +=     ipc.o

1530 IPCMSG_OBJS +=  msg.o

1532 IPCSEM_OBJS +=  sem.o

1534 IPCSHM_OBJS +=  shm.o

1536 #
1537 #                       bignum module
1538 #
1539 COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o

1541 BIGNUM_OBJS += $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)

1543 #
1544 #                       kernel cryptographic framework
1545 #
1546 KCF_OBJS +=     kcf.o kcf_callprov.o kcf_cbufcall.o kcf_cipher.o kcf_crypto.o \
1547                 kcf_cryptoadm.o kcf_ctxops.o kcf_digest.o kcf_dual.o \
1548                 kcf_keys.o kcf_mac.o kcf_mech_tabs.o kcf_miscapi.o \
1549                 kcf_object.o kcf_policy.o kcf_prov_lib.o kcf_prov_tabs.o \
1550                 kcf_sched.o kcf_session.o kcf_sign.o kcf_spi.o kcf_verify.o \
1551                 kcf_random.o modes.o ecb.o cbc.o ctr.o ccm.o gcm.o \
1552                 fips_random.o

1554 CRYPTOADM_OBJS += cryptoadm.o

1556 CRYPTO_OBJS +=  crypto.o

1558 DPROV_OBJS +=   dprov.o

1560 DCA_OBJS +=     dca.o dca_3des.o dca_debug.o dca_dsa.o dca_kstat.o dca_rng.o \
1561                 dca_rsa.o

1563 AESPROV_OBJS += aes.o aes_impl.o aes_modes.o

1565 ARCFOURPROV_OBJS += arcfour.o arcfour_crypt.o

1567 BLOWFISHPROV_OBJS += blowfish.o blowfish_impl.o

1569 ECCPROV_OBJS += ecc.o ec.o ec2_163.o ec2_mont.o ecdecode.o ecl_mult.o \
1570                 ecp_384.o ecp_jac.o ec2_193.o ecl.o ecp_192.o ecp_521.o \
```

```
1571                 ecp_jm.o ec2_233.o ecl_curve.o ecp_224.o ecp_aff.o \
1572                 ecp_mont.o ec2_aff.o ec_naf.o ecl_gf.o ecp_256.o mp_gf2m.o \
1573                 mpi.o mplogic.o mpmontg.o mpprime.o oid.o \
1574                 secitem.o ec2_test.o ecp_test.o

1576 RSAPROV_OBJS += rsa.o rsa_impl.o pkcs1.o

1578 SWRANDPROV_OBJS += swrand.o

1580 #
1581 #                       kernel SSL
1582 #
1583 KSSL_OBJS +=    kssl.o kssl_ioctl.o

1585 KSSL_SOCKFIL_MOD_OBJS += ksslfilter.o ksslapi.o ksslrec.o

1587 #
1588 #                       misc. modules
1589 #

1591 C2AUDIT_OBJS += adr.o audit.o audit_event.o audit_io.o \
1592                 audit_path.o audit_start.o audit_syscalls.o audit_token.o \
1593                 audit_mem.o

1595 PCIC_OBJS +=    pcic.o

1597 RPCSEC_OBJS +=  secmod.o        sec_clnt.o      sec_svc.o       sec_gen.o \
1598                 auth_des.o      auth_kern.o     auth_none.o     auth_loopb.o\
1599                 authdesprt.o    authdesubr.o    authu_prot.o \
1600                 key_call.o      key_prot.o      svc_authu.o     svcauthdes.o

1602 RPCSEC_GSS_OBJS +=      rpcsec_gssmod.o rpcsec_gss.o rpcsec_gss_misc.o \
1603                 rpcsec_gss_utils.o svc_rpcsec_gss.o

1605 CONSCONFIG_OBJS += consconfig.o

1607 CONSCONFIG_DACF_OBJS  += consconfig_dacf.o consplat.o

1609 TEM_OBJS += tem.o tem_safe.o 6x10.o 7x14.o 12x22.o

1611 KBTRANS_OBJS +=                        \
1612                 kbtrans.o               \
1613                 kbtrans_keytables.o     \
1614                 kbtrans_polled.o        \
1615                 kbtrans_streams.o       \
1616                 usb_keytables.o

1618 KGSSD_OBJS +=   gssd_clnt_stubs.o gssd_handle.o gssd_prot.o \
1619                 gss_display_name.o gss_release_name.o gss_import_name.o \
1620                 gss_release_buffer.o gss_release_oid_set.o gen_oids.o gssdmod.o

1622 KGSSD_DERIVED_OBJS = gssd_xdr.o

1624 KGSS_DUMMY_OBJS += dmech.o

1626 KSOCKET_OBJS += ksocket.o ksocket_mod.o

1628 CRYPTO= cksumtypes.o decrypt.o encrypt.o encrypt_length.o etypes.o \
1629         nfold.o verify_checksum.o prng.o block_size.o make_checksum.o\
1630         checksum_length.o hmac.o default_state.o mandatory_sumtype.o

1632 # crypto/des
1633 CRYPTO_DES= f_cbc.o f_cksum.o f_parity.o weak_key.o d3_cbc.o ef_crypto.o

1635 CRYPTO_DK= checksum.o derive.o dk_decrypt.o dk_encrypt.o
```



```
1637 CRYPTO_ARCFOUR= k5_arcfour.o

1639 # crypto/enc_provider
1640 CRYPTO_ENC= des.o des3.o arcfour_provider.o aes_provider.o

1642 # crypto/hash_provider
1643 CRYPTO_HASH= hash_kef_generic.o hash_kmd5.o hash_crc32.o hash_ksha1.o

1645 # crypto/keyhash_provider
1646 CRYPTO_KEYHASH= descbc.o k5_kmd5des.o k_hmac_md5.o

1648 # crypto/crc32
1649 CRYPTO_CRC32= crc32.o

1651 # crypto/old
1652 CRYPTO_OLD= old_decrypt.o old_encrypt.o

1654 # crypto/raw
1655 CRYPTO_RAW= raw_decrypt.o raw_encrypt.o

1657 K5_KRB= kfree.o copy_key.o \
1658         parse.o init_ctx.o \
1659         ser_adata.o ser_addr.o \
1660         ser_auth.o ser_cksum.o \
1661         ser_key.o ser_princ.o \
1662         serialize.o unparse.o \
1663         ser_actx.o

1665 K5_OS=  timeofday.o toffset.o \
1666         init_os_ctx.o c_ustime.o

1668 SEAL=   seal.o unseal.o

1670 MECH=   delete_sec_context.o \
1671         import_sec_context.o \
1672         gssapi_krb5.o \
1673         k5seal.o k5unseal.o k5sealv3.o \
1674         ser_sctx.o \
1675         sign.o \
1676         util_crypt.o  \
1677         util_validate.o  util_ordering.o  \
1678         util_seqnum.o util_set.o util_seed.o \
1679         wrap_size_limit.o verify.o



1683 MECH_GEN= util_token.o


1686 KGSS_KRB5_OBJS += krb5mech.o \
1687         $(MECH) $(SEAL) $(MECH_GEN) \
1688         $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
1689         $(CRYPTO_ENC) $(CRYPTO_HASH) \
1690         $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
1691         $(CRYPTO_OLD) \
1692         $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

1694 DES_OBJS +=     des_crypt.o des_impl.o des_ks.o des_soft.o

1696 DLBOOT_OBJS +=  bootparam_xdr.o nfs_dlinet.o scan.o

1698 KRTLD_OBJS +=   kobj_bootflags.o getoptstr.o \
1699                 kobj.o kobj_kdi.o kobj_lm.o kobj_subr.o

1701 MOD_OBJS +=     modctl.o modsubr.o modsysfile.o modconf.o modhash.o
```

```
1703 STRPLUMB_OBJS += strplumb.o

1705 CPR_OBJS +=     cpr_driver.o cpr_dump.o \
1706                 cpr_main.o cpr_misc.o cpr_mod.o cpr_stat.o \
1707                 cpr_uthread.o

1709 PROF_OBJS +=    prf.o

1711 SE_OBJS += se_driver.o

1713 SYSACCT_OBJS += acct.o

1715 ACCTCTL_OBJS += acctctl.o

1717 EXACCTSYS_OBJS += exacctsys.o

1719 KAIO_OBJS += aio.o

1721 PCMCIA_OBJS += pcmcia.o cs.o cis.o cis_callout.o cis_handlers.o cis_params.o

1723 BUSRA_OBJS += busra.o

1725 PCS_OBJS += pcs.o

1727 PCAN_OBJS += pcan.o

1729 PCATA_OBJS += pcide.o pcdisk.o pclabel.o pcata.o

1731 PCSER_OBJS += pcser.o pcser_cis.o

1733 PCWL_OBJS += pcwl.o

1735 PSET_OBJS +=    pset.o

1737 OHCI_OBJS += ohci.o ohci_hub.o ohci_polled.o

1739 UHCI_OBJS += uhci.o uhciutil.o uhcitgt.o uhcihub.o uhcipolled.o

1741 EHCI_OBJS += ehci.o ehci_hub.o ehci_xfer.o ehci_intr.o ehci_util.o ehci_polled.o

1743 HUBD_OBJS += hubd.o

1745 USB_MID_OBJS += usb_mid.o

1747 USB_IA_OBJS += usb_ia.o

1749 UWBA_OBJS += uwba.o uwbai.o

1751 SCSA2USB_OBJS += scsa2usb.o usb_ms_bulkonly.o usb_ms_cbi.o

1753 HWAHC_OBJS += hwahc.o hwahc_util.o

1755 WUSB_DF_OBJS += wusb_df.o
1756 WUSB_FWMOD_OBJS += wusb_fwmod.o

1758 IPF_OBJS += ip_fil_solaris.o fil.o solaris.o ip_state.o ip_frag.o ip_nat.o \
1759             ip_proxy.o ip_auth.o ip_pool.o ip_htable.o ip_lookup.o \
1760             ip_log.o misc.o ip_compat.o ip_nat6.o drand48.o

1762 IBD_OBJS +=     ibd.o ibd_cm.o

1764 EIBNX_OBJS +=   enx_main.o enx_hdlrs.o enx_ibt.o enx_log.o enx_fip.o \
1765                 enx_misc.o enx_q.o enx_ctl.o

1767 EOIB_OBJS +=    eib_adm.o eib_chan.o eib_cmn.o eib_ctl.o eib_data.o \
1768                 eib_fip.o eib_ibt.o eib_log.o eib_mac.o eib_main.o \
```



```
1769                 eib_rsrc.o eib_svc.o eib_vnic.o

1771 DLPISTUB_OBJS += dlpistub.o

1773 SDP_OBJS +=     sdpddi.o

1775 TRILL_OBJS +=   trill.o

1777 CTF_OBJS += ctf_create.o ctf_decl.o ctf_error.o ctf_hash.o ctf_labels.o \
1778         ctf_lookup.o ctf_open.o ctf_types.o ctf_util.o ctf_subr.o ctf_mod.o

1780 SMBIOS_OBJS += smb_error.o smb_info.o smb_open.o smb_subr.o smb_dev.o

1782 RPCIB_OBJS += rpcib.o

1784 KMDB_OBJS += kdrv.o

1786 AFE_OBJS += afe.o

1788 BGE_OBJS += bge_main2.o bge_chip2.o bge_kstats.o bge_log.o bge_ndd.o \
1789                 bge_atomic.o bge_mii.o bge_send.o bge_recv2.o bge_mii_5906.o

1791 DMFE_OBJS += dmfe_log.o dmfe_main.o dmfe_mii.o

1793 EFE_OBJS += efe.o

1795 ELXL_OBJS += elxl.o

1797 HME_OBJS += hme.o

1799 IXGB_OBJS += ixgb.o ixgb_atomic.o ixgb_chip.o ixgb_gld.o ixgb_kstats.o \
1800                 ixgb_log.o ixgb_ndd.o ixgb_rx.o ixgb_tx.o ixgb_xmii.o

1802 NGE_OBJS += nge_main.o nge_atomic.o nge_chip.o nge_ndd.o nge_kstats.o \
1803                 nge_log.o nge_rx.o nge_tx.o nge_xmii.o

1805 PCN_OBJS += pcn.o

1807 RGE_OBJS += rge_main.o rge_chip.o rge_ndd.o rge_kstats.o rge_log.o rge_rxtx.o

1809 URTW_OBJS += urtw.o

1811 ARN_OBJS += arn_hw.o arn_eeprom.o arn_mac.o arn_calib.o arn_ani.o arn_phy.o arn_
1812                 arn_main.o arn_recv.o arn_xmit.o arn_rc.o

1814 ATH_OBJS += ath_aux.o ath_main.o ath_osdep.o ath_rate.o

1816 ATU_OBJS += atu.o

1818 IPW_OBJS += ipw2100_hw.o ipw2100.o

1820 IWI_OBJS += ipw2200_hw.o ipw2200.o

1822 IWH_OBJS += iwh.o

1824 IWK_OBJS += iwk2.o

1826 IWP_OBJS += iwp.o

1828 MWL_OBJS += mwl.o

1830 MWLFW_OBJS += mwlfw_mode.o

1832 WPI_OBJS += wpi.o

1834 RAL_OBJS += rt2560.o ral_rate.o
```

```
1836 RUM_OBJS += rum.o

1838 RWD_OBJS += rt2661.o

1840 RWN_OBJS += rt2860.o

1842 UATH_OBJS += uath.o

1844 UATHFW_OBJS += uathfw_mod.o

1846 URAL_OBJS += ural.o

1848 RTW_OBJS += rtw.o smc93cx6.o rtwphy.o rtwphyio.o

1850 ZYD_OBJS += zyd.o zyd_usb.o zyd_hw.o zyd_fw.o

1852 MXFE_OBJS += mxfe.o

1854 MPTSAS_OBJS += mptsas.o mptsas_impl.o mptsas_init.o mptsas_raid.o mptsas_smhba.o

1856 SFE_OBJS += sfe.o sfe_util.o

1858 BFE_OBJS += bfe.o

1860 BRIDGE_OBJS += bridge.o

1862 IDM_SHARED_OBJS += base64.o

1864 IDM_OBJS +=     $(IDM_SHARED_OBJS) \
1865                 idm.o idm_impl.o idm_text.o idm_conn_sm.o idm_so.o

1867 VR_OBJS += vr.o

1869 ATGE_OBJS += atge_main.o atge_l1e.o atge_mii.o atge_l1.o atge_l1c.o

1871 YGE_OBJS = yge.o

1873 #
1874 #       Build up defines and paths.
1875 #
1876 LINT_DEFS       += -Dunix

1878 #
1879 #       This duality can be removed when the native and target compilers
1880 #       are the same (or at least recognize the same command line syntax!)
1881 #       It is a bug in the current compilation system that the assember
1882 #       can't process the -Y I, flag.
1883 #
1884 NATIVE_INC_PATH += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1885 AS_INC_PATH     += $(INC_PATH) -I$(UTSBASE)/common
1886 INCLUDE_PATH    += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common

1888 PCIEB_OBJS += pcieb.o

1890 #       Chelsio N110 10G NIC driver module
1891 #
1892 CH_OBJS = ch.o glue.o pe.o sge.o

1894 CH_COM_OBJS =   ch_mac.o ch_subr.o cspi.o espi.o ixf1010.o mc3.o mc4.o mc5.o \
1895                 mv88e1xxx.o mv88x201x.o my3126.o pm3393.o tp.o ulp.o \
1896                 vsc7321.o vsc7326.o xpak.o

1898 #
1899 #       Chelsio Terminator 4 10G NIC nexus driver module
1900 #
```



```
1901 CXGBE_FW_OBJS  =        t4_fw.o t4_cfg.o
1902 CXGBE_COM_OBJS =        t4_hw.o common.o
1903 CXGBE_NEX_OBJS =        t4_nexus.o t4_sge.o t4_mac.o t4_ioctl.o shared.o \
1904                         t4_l2t.o adapter.o osdep.o

1906 #
1907 #       Chelsio Terminator 4 10G NIC driver module
1908 #
1909 CXGBE_OBJS =    cxgbe.o

1911 #
1912 #       PCI strings file
1913 #
1914 PCI_STRING_OBJS = pci_strings.o

1916 NET_DACF_OBJS += net_dacf.o

1918 #
1919 #       Xframe 10G NIC driver module
1920 #
1921 XGE_OBJS = xge.o xgell.o

1923 XGE_HAL_OBJS =  xgehal-channel.o xgehal-fifo.o xgehal-ring.o  xgehal-config.o \
1924                 xgehal-driver.o  xgehal-mm.o xgehal-stats.o  xgehal-device.o \
1925                 xge-queue.o  xgehal-mgmt.o xgehal-mgmtaux.o

1927 #
1928 #       e1000g module
1929 #
1930 E1000G_OBJS +=  e1000_80003es2lan.o e1000_82540.o e1000_82541.o e1000_82542.o \
1931                 e1000_82543.o e1000_82571.o e1000_api.o e1000_ich8lan.o \
1932                 e1000_mac.o e1000_manage.o e1000_nvm.o e1000_osdep.o \
1933                 e1000_phy.o e1000g_debug.o e1000g_main.o e1000g_alloc.o \
1934                 e1000g_tx.o e1000g_rx.o e1000g_stat.o

1936 #
1937 #       Intel 82575 1G NIC driver module
1938 #
1939 IGB_OBJS =      igb_82575.o igb_api.o igb_mac.o igb_manage.o \
1940                 igb_nvm.o igb_osdep.o igb_phy.o igb_buf.o \
1941                 igb_debug.o igb_gld.o igb_log.o igb_main.o \
1942                 igb_rx.o igb_stat.o igb_tx.o

1944 #
1945 #       Intel Pro/100 NIC driver module
1946 #
1947 IPRB_OBJS =     iprb.o

1949 #
1950 #       Intel 10GbE PCIE NIC driver module
1951 #
1952 IXGBE_OBJS =    ixgbe_82598.o ixgbe_82599.o ixgbe_api.o         \
1953                 ixgbe_common.o ixgbe_phy.o                      \
1954                 ixgbe_buf.o ixgbe_debug.o ixgbe_gld.o           \
1955                 ixgbe_log.o ixgbe_main.o                        \
1956                 ixgbe_osdep.o ixgbe_rx.o ixgbe_stat.o           \
1957                 ixgbe_tx.o  ixgbe_x540.o ixgbe_mbx.o

1959 #
1960 #       NIU 10G/1G driver module
1961 #
1962 NXGE_OBJS =     nxge_mac.o nxge_ipp.o nxge_rxdma.o              \
1963                 nxge_txdma.o nxge_txc.o nxge_main.o             \
1964                 nxge_hw.o nxge_fzc.o nxge_virtual.o             \
1965                 nxge_send.o nxge_classify.o nxge_fflp.o         \
1966                 nxge_fflp_hash.o nxge_ndd.o nxge_kstats.o       \
```

```
1967                    nxge_zcp.o nxge_fm.o nxge_espc.o nxge_hv.o      \
1968                    nxge_hio.o nxge_hio_guest.o nxge_intr.o

1970 NXGE_NPI_OBJS = \
1971                    npi.o npi_mac.o npi_ipp.o                       \
1972                    npi_txdma.o npi_rxdma.o npi_txc.o               \
1973                    npi_zcp.o npi_espc.o npi_fflp.o                 \
1974                    npi_vir.o

1976 NXGE_HCALL_OBJS =       \
1977                    nxge_hcall.o

1979 #
1980 # Virtio modules
1981 #

1983 # Virtio core
1984 VIRTIO_OBJS = virtio.o

1986 # Virtio block driver
1987 VIOBLK_OBJS = vioblk.o

1989 #
1990 #       kiconv modules
1991 #
1992 KICONV_EMEA_OBJS += kiconv_emea.o

1994 KICONV_JA_OBJS += kiconv_ja.o

1996 KICONV_KO_OBJS += kiconv_cck_common.o kiconv_ko.o

1998 KICONV_SC_OBJS += kiconv_cck_common.o kiconv_sc.o

2000 KICONV_TC_OBJS += kiconv_cck_common.o kiconv_tc.o

2002 #
2003 #       AAC module
2004 #
2005 AAC_OBJS = aac.o aac_ioctl.o

2007 #
2008 #       sdcard modules
2009 #
2010 SDA_OBJS =      sda_cmd.o sda_host.o sda_init.o sda_mem.o sda_mod.o sda_slot.o
2011 SDHOST_OBJS =   sdhost.o

2013 #
2014 #       hxge 10G driver module
2015 #
2016 HXGE_OBJS =     hxge_main.o hxge_vmac.o hxge_send.o              \
2017                    hxge_txdma.o hxge_rxdma.o hxge_virtual.o        \
2018                    hxge_fm.o hxge_fzc.o hxge_hw.o hxge_kstats.o    \
2019                    hxge_ndd.o hxge_pfc.o                           \
2020                    hpi.o hpi_vmac.o hpi_rxdma.o hpi_txdma.o        \
2021                    hpi_vir.o hpi_pfc.o

2023 #
2024 #       MEGARAID_SAS module
2025 #
2026 MEGA_SAS_OBJS = megaraid_sas.o

2028 #
2029 #       MR_SAS module
2030 #
2031 MR_SAS_OBJS = ld_pd_map.o mr_sas.o mr_sas_tbolt.o mr_sas_list.o
```



```
2033 #
2034 #       ISCSI_INITIATOR module
2035 #
2036 ISCSI_INITIATOR_OBJS =  chap.o iscsi_io.o iscsi_thread.o        \
2037                         iscsi_ioctl.o iscsid.o iscsi.o          \
2038                         iscsi_login.o isns_client.o iscsiAuthClient.o   \
2039                         iscsi_lun.o iscsiAuthClientGlue.o       \
2040                         iscsi_net.o nvfile.o iscsi_cmd.o        \
2041                         iscsi_queue.o persistent.o iscsi_conn.o \
2042                         iscsi_sess.o radius_auth.o iscsi_crc.o  \
2043                         iscsi_stats.o radius_packet.o iscsi_doorclt.o   \
2044                         iscsi_targetparam.o utils.o kifconf.o

2046 #
2047 #       ntxn 10Gb/1Gb NIC driver module
2048 #
2049 NTXN_OBJS =     unm_nic_init.o unm_gem.o unm_nic_hw.o unm_ndd.o \
2050                         unm_nic_main.o unm_nic_isr.o unm_nic_ctx.o niu.o

2052 #
2053 #       Myricom 10Gb NIC driver module
2054 #
2055 MYRI10GE_OBJS = myri10ge.o myri10ge_lro.o

2057 #       nulldriver module
2058 #
2059 NULLDRIVER_OBJS =       nulldriver.o

2061 TPM_OBJS =      tpm.o tpm_hcall.o
```

```
**********************************************************
   56685 Tue Sep  3 20:26:58 2013
new/usr/src/uts/common/fs/zfs/dnode.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
______unchanged_portion_omitted_

1311 /*
1312  * Try to change the block size for the indicated dnode.  This can only
1313  * succeed if there are no blocks allocated or dirty beyond first block
1314  */
1315 int
1316 dnode_set_blksz(dnode_t *dn, uint64_t size, int ibs, dmu_tx_t *tx)
1317 {
1318         dmu_buf_impl_t *db, *db_next;
1319         int err;

1321         if (size == 0)
1322                 size = SPA_MINBLOCKSIZE;
1323         if (size > SPA_MAXBLOCKSIZE)
1324                 size = SPA_MAXBLOCKSIZE;
1325         else
1326                 size = P2ROUNDUP(size, SPA_MINBLOCKSIZE);

1328         if (ibs == dn->dn_indblkshift)
1329                 ibs = 0;

1331         if (size >> SPA_MINBLOCKSHIFT == dn->dn_datablkszsec && ibs == 0)
1332                 return (0);

1334         rw_enter(&dn->dn_struct_rwlock, RW_WRITER);

1336         /* Check for any allocated blocks beyond the first */
1337         if (dn->dn_maxblkid != 0)
1337         if (dn->dn_phys->dn_maxblkid != 0)
1338                 goto fail;

1340         mutex_enter(&dn->dn_dbufs_mtx);
1341         for (db = list_head(&dn->dn_dbufs); db; db = db_next) {
1342                 db_next = list_next(&dn->dn_dbufs, db);

1344                 if (db->db_blkid != 0 && db->db_blkid != DMU_BONUS_BLKID &&
1345                     db->db_blkid != DMU_SPILL_BLKID) {
1346                         mutex_exit(&dn->dn_dbufs_mtx);
1347                         goto fail;
1348                 }
1349         }
1350         mutex_exit(&dn->dn_dbufs_mtx);

1352         if (ibs && dn->dn_nlevels != 1)
1353                 goto fail;

1355         /* resize the old block */
1356         err = dbuf_hold_impl(dn, 0, 0, TRUE, FTAG, &db);
1357         if (err == 0)
1358                 dbuf_new_size(db, size, tx);
1359         else if (err != ENOENT)
1360                 goto fail;
```



```
1362         dnode_setdblksz(dn, size);
1363         dnode_setdirty(dn, tx);
1364         dn->dn_next_blksz[tx->tx_txg&TXG_MASK] = size;
1365         if (ibs) {
1366                 dn->dn_indblkshift = ibs;
1367                 dn->dn_next_indblkshift[tx->tx_txg&TXG_MASK] = ibs;
1368         }
1369         /* rele after we have fixed the blocksize in the dnode */
1370         if (db)
1371                 dbuf_rele(db, FTAG);

1373         rw_exit(&dn->dn_struct_rwlock);
1374         return (0);

1376 fail:
1377         rw_exit(&dn->dn_struct_rwlock);
1378         return (SET_ERROR(ENOTSUP));
1379 }
______unchanged_portion_omitted_
```

```
**********************************************************
   62230 Tue Sep  3 20:26:59 2013
new/usr/src/uts/common/fs/zfs/metaslab.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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 (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
 */

#include <sys/zfs_context.h>
#include <sys/dmu.h>
#include <sys/dmu_tx.h>
#include <sys/space_map.h>
#include <sys/metaslab_impl.h>
#include <sys/vdev_impl.h>
#include <sys/zio.h>
#include <sys/spa_impl.h>

/*
 * Allow allocations to switch to gang blocks quickly. We do this to
 * avoid having to load lots of space_maps in a given txg. There are,
 * however, some cases where we want to avoid "fast" ganging and instead
 * we want to do an exhaustive search of all metaslabs on this device.
 * Currently we don't allow any gang, zil, or dump device related allocations
 * to "fast" gang.
 */
#define CAN_FASTGANG(flags) \
        (!((flags) & (METASLAB_GANG_CHILD | METASLAB_GANG_HEADER | \
        METASLAB_GANG_AVOID)))

#define METASLAB_WEIGHT_PRIMARY         (1ULL << 63)
#define METASLAB_WEIGHT_SECONDARY       (1ULL << 62)
#define METASLAB_ACTIVE_MASK            \
        (METASLAB_WEIGHT_PRIMARY | METASLAB_WEIGHT_SECONDARY)

uint64_t metaslab_aliquot = 512ULL << 10;
```



```
uint64_t metaslab_gang_bang = SPA_MAXBLOCKSIZE + 1;     /* force gang blocks */

/*
 * The in-core space map representation is more compact than its on-disk form.
 * The zfs_condense_pct determines how much more compact the in-core
 * space_map representation must be before we compact it on-disk.
 * Values should be greater than or equal to 100.
 */
int zfs_condense_pct = 200;

/*
 * This value defines the number of allowed allocation failures per vdev.
 * If a device reaches this threshold in a given txg then we consider skipping
 * allocations on that device. The value of zfs_mg_alloc_failures is computed
 * in zio_init() unless it has been overridden in /etc/system.
 */
int zfs_mg_alloc_failures = 0;

/*
 * The zfs_mg_noalloc_threshold defines which metaslab groups should
 * be eligible for allocation. The value is defined as a percentage of
 * a free space. Metaslab groups that have more free space than
 * zfs_mg_noalloc_threshold are always eligible for allocations. Once
 * a metaslab group's free space is less than or equal to the
 * zfs_mg_noalloc_threshold the allocator will avoid allocating to that
 * group unless all groups in the pool have reached zfs_mg_noalloc_threshold.
 * Once all groups in the pool reach zfs_mg_noalloc_threshold then all
 * groups are allowed to accept allocations. Gang blocks are always
 * eligible to allocate on any metaslab group. The default value of 0 means
 * no metaslab group will be excluded based on this criterion.
 */
int zfs_mg_noalloc_threshold = 0;

/*
 * When set will load all metaslabs when pool is first opened.
 * Metaslab debugging: when set, keeps all space maps in core to verify frees.
 */
int metaslab_debug_load = 0;
static int metaslab_debug = 0;

/*
 * When set will prevent metaslabs from being unloaded.
 */
int metaslab_debug_unload = 0;

/*
 * Minimum size which forces the dynamic allocator to change
 * it's allocation strategy.  Once the space map cannot satisfy
 * an allocation of this size then it switches to using more
 * aggressive strategy (i.e search by size rather than offset).
 */
uint64_t metaslab_df_alloc_threshold = SPA_MAXBLOCKSIZE;

/*
 * The minimum free space, in percent, which must be available
 * in a space map to continue allocations in a first-fit fashion.
 * Once the space_map's free space drops below this level we dynamically
 * switch to using best-fit allocations.
 */
int metaslab_df_free_pct = 4;

/*
 * A metaslab is considered "free" if it contains a contiguous
 * segment which is greater than metaslab_min_alloc_size.
 */
uint64_t metaslab_min_alloc_size = DMU_MAX_ACCESS;
```

```
 119 /*
 120  * Percentage of all cpus that can be used by the metaslab taskq.
 109  * Max number of space_maps to prefetch.
 121  */
 122 int metaslab_load_pct = 50;
 111 int metaslab_prefetch_limit = SPA_DVAS_PER_BP;

 124 /*
 125  * Determines how many txgs a metaslab may remain loaded without having any
 126  * allocations from it. As long as a metaslab continues to be used we will
 127  * keep it loaded.
 114  * Percentage bonus multiplier for metaslabs that are in the bonus area.
 128  */
 129 int metaslab_unload_delay = TXG_SIZE * 2;
 116 int metaslab_smo_bonus_pct = 150;

 131 /*
 132  * Should we be willing to write data to degraded vdevs?
 133  */
 134 boolean_t zfs_write_to_degraded = B_FALSE;

 136 /*
 137  * Max number of metaslabs per group to preload.
 138  */
 139 int metaslab_preload_limit = SPA_DVAS_PER_BP;

 141 /*
 142  * Enable/disable preloading of metaslab.
 143  */
 144 boolean_t metaslab_preload_enabled = B_TRUE;

 146 /*
 147  * Enable/disable additional weight factor for each metaslab.
 148  */
 149 boolean_t metaslab_weight_factor_enable = B_FALSE;


 152 /*
 153  * ==========================================================================
 154  * Metaslab classes
 155  * ==========================================================================
 156  */
 157 metaslab_class_t *
 158 metaslab_class_create(spa_t *spa, metaslab_ops_t *ops)
 129 metaslab_class_create(spa_t *spa, space_map_ops_t *ops)
 159 {
 160         metaslab_class_t *mc;

 162         mc = kmem_zalloc(sizeof (metaslab_class_t), KM_SLEEP);

 164         mc->mc_spa = spa;
 165         mc->mc_rotor = NULL;
 166         mc->mc_ops = ops;

 168         return (mc);
 169 }
_____unchanged_portion_omitted_

 243 /*
 244  * ==========================================================================
 245  * Metaslab groups
 246  * ==========================================================================
 247  */
 248 static int
 249 metaslab_compare(const void *x1, const void *x2)
```



```
 250 {
 251         const metaslab_t *m1 = x1;
 252         const metaslab_t *m2 = x2;

 254         if (m1->ms_weight < m2->ms_weight)
 255                 return (1);
 256         if (m1->ms_weight > m2->ms_weight)
 257                 return (-1);

 259         /*
 260          * If the weights are identical, use the offset to force uniqueness.
 261          */
 262         if (m1->ms_start < m2->ms_start)
 233         if (m1->ms_map->sm_start < m2->ms_map->sm_start)
 263                 return (-1);
 264         if (m1->ms_start > m2->ms_start)
 235         if (m1->ms_map->sm_start > m2->ms_map->sm_start)
 265                 return (1);

 267         ASSERT3P(m1, ==, m2);

 269         return (0);
 270 }
_____unchanged_portion_omitted_

 319 metaslab_group_t *
 320 metaslab_group_create(metaslab_class_t *mc, vdev_t *vd)
 321 {
 322         metaslab_group_t *mg;

 324         mg = kmem_zalloc(sizeof (metaslab_group_t), KM_SLEEP);
 325         mutex_init(&mg->mg_lock, NULL, MUTEX_DEFAULT, NULL);
 326         avl_create(&mg->mg_metaslab_tree, metaslab_compare,
 327             sizeof (metaslab_t), offsetof(struct metaslab, ms_group_node));
 328         mg->mg_vd = vd;
 329         mg->mg_class = mc;
 330         mg->mg_activation_count = 0;

 332         mg->mg_taskq = taskq_create("metaslab_group_tasksq", metaslab_load_pct,
 333             minclsyspri, 10, INT_MAX, TASKQ_THREADS_CPU_PCT);

 335         return (mg);
 336 }
_____unchanged_portion_omitted_

 387 void
 388 metaslab_group_passivate(metaslab_group_t *mg)
 389 {
 390         metaslab_class_t *mc = mg->mg_class;
 391         metaslab_group_t *mgprev, *mgnext;

 393         ASSERT(spa_config_held(mc->mc_spa, SCL_ALLOC, RW_WRITER));

 395         if (--mg->mg_activation_count != 0) {
 396                 ASSERT(mc->mc_rotor != mg);
 397                 ASSERT(mg->mg_prev == NULL);
 398                 ASSERT(mg->mg_next == NULL);
 399                 ASSERT(mg->mg_activation_count < 0);
 400                 return;
 401         }

 403         taskq_wait(mg->mg_taskq);

 405         mgprev = mg->mg_prev;
 406         mgnext = mg->mg_next;
```

```
        if (mg == mgnext) {
                mc->mc_rotor = NULL;
        } else {
                mc->mc_rotor = mgnext;
                mgprev->mg_next = mgnext;
                mgnext->mg_prev = mgprev;
        }

        mg->mg_prev = NULL;
        mg->mg_next = NULL;
}
```

*unchanged_portion_omitted_*

```
/*
 * ==========================================================================
 * Range tree callbacks
 * Common allocator routines
 * ==========================================================================
 */

/*
 * Comparison function for the private size-ordered tree. Tree is sorted
 * by size, larger sizes at the end of the tree.
 */
static int
metaslab_rangesize_compare(const void *x1, const void *x2)
metaslab_segsize_compare(const void *x1, const void *x2)
{
        const range_seg_t *r1 = x1;
        const range_seg_t *r2 = x2;
        uint64_t rs_size1 = r1->rs_end - r1->rs_start;
        uint64_t rs_size2 = r2->rs_end - r2->rs_start;
        const space_seg_t *s1 = x1;
        const space_seg_t *s2 = x2;
        uint64_t ss_size1 = s1->ss_end - s1->ss_start;
        uint64_t ss_size2 = s2->ss_end - s2->ss_start;

        if (rs_size1 < rs_size2)
        if (ss_size1 < ss_size2)
                return (-1);
        if (rs_size1 > rs_size2)
        if (ss_size1 > ss_size2)
                return (1);

        if (r1->rs_start < r2->rs_start)
        if (s1->ss_start < s2->ss_start)
                return (-1);

        if (r1->rs_start > r2->rs_start)
        if (s1->ss_start > s2->ss_start)
                return (1);

        return (0);
}

/*
 * Create any block allocator specific components. The current allocators
 * rely on using both a size-ordered range_tree_t and an array of uint64_t's.
 * This is a helper function that can be used by the allocator to find
 * a suitable block to allocate. This will search the specified AVL
 * tree looking for a block that matches the specified criteria.
 */
static void
metaslab_rt_create(range_tree_t *rt, void *arg)
static uint64_t
metaslab_block_picker(avl_tree_t *t, uint64_t *cursor, uint64_t size,
```



```
    uint64_t align)
{
        metaslab_t *msp = arg;
        space_seg_t *ss, ssearch;
        avl_index_t where;

        ASSERT3P(rt->rt_arg, ==, msp);
        ASSERT(msp->ms_tree == NULL);
        ssearch.ss_start = *cursor;
        ssearch.ss_end = *cursor + size;

        avl_create(&msp->ms_size_tree, metaslab_rangesize_compare,
            sizeof (range_seg_t), offsetof(range_seg_t, rs_pp_node));
}
        ss = avl_find(t, &ssearch, &where);
        if (ss == NULL)
                ss = avl_nearest(t, where, AVL_AFTER);

/*
 * Destroy the block allocator specific components.
        while (ss != NULL) {
                uint64_t offset = P2ROUNDUP(ss->ss_start, align);

                if (offset + size <= ss->ss_end) {
                        *cursor = offset + size;
                        return (offset);
                }
                ss = AVL_NEXT(t, ss);
        }

        /*
         * If we know we've searched the whole map (*cursor == 0), give up.
         * Otherwise, reset the cursor to the beginning and try again.
 */
        if (*cursor == 0)
                return (-1ULL);

        *cursor = 0;
        return (metaslab_block_picker(t, cursor, size, align));
}

static void
metaslab_rt_destroy(range_tree_t *rt, void *arg)
metaslab_pp_load(space_map_t *sm)
{
        metaslab_t *msp = arg;
        space_seg_t *ss;

        ASSERT3P(rt->rt_arg, ==, msp);
        ASSERT3P(msp->ms_tree, ==, rt);
        ASSERT0(avl_numnodes(&msp->ms_size_tree));
        ASSERT(sm->sm_ppd == NULL);
        sm->sm_ppd = kmem_zalloc(64 * sizeof (uint64_t), KM_SLEEP);

        avl_destroy(&msp->ms_size_tree);
        sm->sm_pp_root = kmem_alloc(sizeof (avl_tree_t), KM_SLEEP);
        avl_create(sm->sm_pp_root, metaslab_segsize_compare,
            sizeof (space_seg_t), offsetof(struct space_seg, ss_pp_node));

        for (ss = avl_first(&sm->sm_root); ss; ss = AVL_NEXT(&sm->sm_root, ss))
                avl_add(sm->sm_pp_root, ss);
}

static void
metaslab_rt_add(range_tree_t *rt, range_seg_t *rs, void *arg)
metaslab_pp_unload(space_map_t *sm)
```

```
 547 {
 548         metaslab_t *msp = arg;
 533         void *cookie = NULL;

 550         ASSERT3P(rt->rt_arg, ==, msp);
 551         ASSERT3P(msp->ms_tree, ==, rt);
 552         VERIFY(!msp->ms_condensing);
 553         avl_add(&msp->ms_size_tree, rs);
 535         kmem_free(sm->sm_ppd, 64 * sizeof (uint64_t));
 536         sm->sm_ppd = NULL;

 538         while (avl_destroy_nodes(sm->sm_pp_root, &cookie) != NULL) {
 539                 /* tear down the tree */
 540         }

 542         avl_destroy(sm->sm_pp_root);
 543         kmem_free(sm->sm_pp_root, sizeof (avl_tree_t));
 544         sm->sm_pp_root = NULL;
 554 }

 547 /* ARGSUSED */
 556 static void
 557 metaslab_rt_remove(range_tree_t *rt, range_seg_t *rs, void *arg)
 549 metaslab_pp_claim(space_map_t *sm, uint64_t start, uint64_t size)
 558 {
 559         metaslab_t *msp = arg;

 561         ASSERT3P(rt->rt_arg, ==, msp);
 562         ASSERT3P(msp->ms_tree, ==, rt);
 563         VERIFY(!msp->ms_condensing);
 564         avl_remove(&msp->ms_size_tree, rs);
 551         /* No need to update cursor */
 565 }

 554 /* ARGSUSED */
 567 static void
 568 metaslab_rt_vacate(range_tree_t *rt, void *arg)
 556 metaslab_pp_free(space_map_t *sm, uint64_t start, uint64_t size)
 569 {
 570         metaslab_t *msp = arg;

 572         ASSERT3P(rt->rt_arg, ==, msp);
 573         ASSERT3P(msp->ms_tree, ==, rt);

 575         /*
 576          * Normally one would walk the tree freeing nodes along the way.
 577          * Since the nodes are shared with the range trees we can avoid
 578          * walking all nodes and just reinitialize the avl tree. The nodes
 579          * will be freed by the range tree, so we don't want to free them here.
 580          */
 581         avl_create(&msp->ms_size_tree, metaslab_rangesize_compare,
 582             sizeof (range_seg_t), offsetof(range_seg_t, rs_pp_node));
 558         /* No need to update cursor */
 583 }

 585 static range_tree_ops_t metaslab_rt_ops = {
 586         metaslab_rt_create,
 587         metaslab_rt_destroy,
 588         metaslab_rt_add,
 589         metaslab_rt_remove,
 590         metaslab_rt_vacate
 591 };

 593 /*
 594  * ==========================================================================
 595  * Metaslab block operations
```



```
 596  * ==========================================================================
 597  */

 599 /*
 600  * Return the maximum contiguous segment within the metaslab.
 601  */
 602 uint64_t
 603 metaslab_block_maxsize(metaslab_t *msp)
 565 metaslab_pp_maxsize(space_map_t *sm)
 604 {
 605         avl_tree_t *t = &msp->ms_size_tree;
 606         range_seg_t *rs;
 567         avl_tree_t *t = sm->sm_pp_root;
 568         space_seg_t *ss;

 608         if (t == NULL || (rs = avl_last(t)) == NULL)
 570         if (t == NULL || (ss = avl_last(t)) == NULL)
 609                 return (0ULL);

 611         return (rs->rs_end - rs->rs_start);
 573         return (ss->ss_end - ss->ss_start);
 612 }

 614 uint64_t
 615 metaslab_block_alloc(metaslab_t *msp, uint64_t size)
 616 {
 617         uint64_t start;
 618         range_tree_t *rt = msp->ms_tree;

 620         VERIFY(!msp->ms_condensing);

 622         start = msp->ms_ops->msop_alloc(msp, size);
 623         if (start != -1ULL) {
 624                 vdev_t *vd = msp->ms_group->mg_vd;

 626                 VERIFY0(P2PHASE(start, 1ULL << vd->vdev_ashift));
 627                 VERIFY0(P2PHASE(size, 1ULL << vd->vdev_ashift));
 628                 VERIFY3U(range_tree_space(rt) - size, <=, msp->ms_size);
 629                 range_tree_remove(rt, start, size);
 630         }
 631         return (start);
 632 }

 634 /*
 635  * ==========================================================================
 636  * Common allocator routines
 637  * ==========================================================================
 638  */

 640 /*
 641  * This is a helper function that can be used by the allocator to find
 642  * a suitable block to allocate. This will search the specified AVL
 643  * tree looking for a block that matches the specified criteria.
 644  */
 645 static uint64_t
 646 metaslab_block_picker(avl_tree_t *t, uint64_t *cursor, uint64_t size,
 647     uint64_t align)
 648 {
 649         range_seg_t *rs, rsearch;
 650         avl_index_t where;

 652         rsearch.rs_start = *cursor;
 653         rsearch.rs_end = *cursor + size;

 655         rs = avl_find(t, &rsearch, &where);
 656         if (rs == NULL)
```

```
657                 rs = avl_nearest(t, where, AVL_AFTER);

659         while (rs != NULL) {
660                 uint64_t offset = P2ROUNDUP(rs->rs_start, align);

662                 if (offset + size <= rs->rs_end) {
663                         *cursor = offset + size;
664                         return (offset);
665                 }
666                 rs = AVL_NEXT(t, rs);
667         }

669         /*
670          * If we know we've searched the whole map (*cursor == 0), give up.
671          * Otherwise, reset the cursor to the beginning and try again.
672          */
673         if (*cursor == 0)
674                 return (-1ULL);

676         *cursor = 0;
677         return (metaslab_block_picker(t, cursor, size, align));
678 }

680 /*
681  * ==========================================================================
682  * The first-fit block allocator
683  * ==========================================================================
684  */
685 static uint64_t
686 metaslab_ff_alloc(metaslab_t *msp, uint64_t size)
582 metaslab_ff_alloc(space_map_t *sm, uint64_t size)
687 {
688         /*
689          * Find the largest power of 2 block size that evenly divides the
690          * requested size. This is used to try to allocate blocks with similar
691          * alignment from the same area of the metaslab (i.e. same cursor
692          * bucket) but it does not guarantee that other allocations sizes
693          * may exist in the same region.
694          */
584         avl_tree_t *t = &sm->sm_root;
695         uint64_t align = size & -size;
696         uint64_t *cursor = &msp->ms_lbas[highbit(align) - 1];
697         avl_tree_t *t = &msp->ms_tree->rt_root;
586         uint64_t *cursor = (uint64_t *)sm->sm_ppd + highbit(align) - 1;

699         return (metaslab_block_picker(t, cursor, size, align));
700 }

702 /* ARGSUSED */
703 static boolean_t
704 metaslab_ff_fragmented(metaslab_t *msp)
592 boolean_t
593 metaslab_ff_fragmented(space_map_t *sm)
705 {
706         return (B_TRUE);
707 }

709 static metaslab_ops_t metaslab_ff_ops = {
598 static space_map_ops_t metaslab_ff_ops = {
599         metaslab_pp_load,
600         metaslab_pp_unload,
710         metaslab_ff_alloc,
602         metaslab_pp_claim,
603         metaslab_pp_free,
604         metaslab_pp_maxsize,
711         metaslab_ff_fragmented
```



```
712 };

714 /*
715  * ==========================================================================
716  * Dynamic block allocator -
717  * Uses the first fit allocation scheme until space get low and then
718  * adjusts to a best fit allocation method. Uses metaslab_df_alloc_threshold
719  * and metaslab_df_free_pct to determine when to switch the allocation scheme.
720  * ==========================================================================
721  */
722 static uint64_t
723 metaslab_df_alloc(metaslab_t *msp, uint64_t size)
617 metaslab_df_alloc(space_map_t *sm, uint64_t size)
724 {
725         /*
726          * Find the largest power of 2 block size that evenly divides the
727          * requested size. This is used to try to allocate blocks with similar
728          * alignment from the same area of the metaslab (i.e. same cursor
729          * bucket) but it does not guarantee that other allocations sizes
730          * may exist in the same region.
731          */
619         avl_tree_t *t = &sm->sm_root;
732         uint64_t align = size & -size;
733         uint64_t *cursor = &msp->ms_lbas[highbit(align) - 1];
734         range_tree_t *rt = msp->ms_tree;
735         avl_tree_t *t = &rt->rt_root;
736         uint64_t max_size = metaslab_block_maxsize(msp);
737         int free_pct = range_tree_space(rt) * 100 / msp->ms_size;
621         uint64_t *cursor = (uint64_t *)sm->sm_ppd + highbit(align) - 1;
622         uint64_t max_size = metaslab_pp_maxsize(sm);
623         int free_pct = sm->sm_space * 100 / sm->sm_size;

739         ASSERT(MUTEX_HELD(&msp->ms_lock));
740         ASSERT3U(avl_numnodes(t), ==, avl_numnodes(&msp->ms_size_tree));
625         ASSERT(MUTEX_HELD(sm->sm_lock));
626         ASSERT3U(avl_numnodes(&sm->sm_root), ==, avl_numnodes(sm->sm_pp_root));

742         if (max_size < size)
743                 return (-1ULL);

745         /*
746          * If we're running low on space switch to using the size
747          * sorted AVL tree (best-fit).
748          */
749         if (max_size < metaslab_df_alloc_threshold ||
750             free_pct < metaslab_df_free_pct) {
751                 t = &msp->ms_size_tree;
637                 t = sm->sm_pp_root;
752                 *cursor = 0;
753         }

755         return (metaslab_block_picker(t, cursor, size, 1ULL));
756 }

758 static boolean_t
759 metaslab_df_fragmented(metaslab_t *msp)
645 metaslab_df_fragmented(space_map_t *sm)
760 {
761         range_tree_t *rt = msp->ms_tree;
762         uint64_t max_size = metaslab_block_maxsize(msp);
763         int free_pct = range_tree_space(rt) * 100 / msp->ms_size;
647         uint64_t max_size = metaslab_pp_maxsize(sm);
648         int free_pct = sm->sm_space * 100 / sm->sm_size;

765         if (max_size >= metaslab_df_alloc_threshold &&
766             free_pct >= metaslab_df_free_pct)
```

```
767                 return (B_FALSE);

769         return (B_TRUE);
770 }

772 static metaslab_ops_t metaslab_df_ops = {
657 static space_map_ops_t metaslab_df_ops = {
658         metaslab_pp_load,
659         metaslab_pp_unload,
773         metaslab_df_alloc,
661         metaslab_pp_claim,
662         metaslab_pp_free,
663         metaslab_pp_maxsize,
774         metaslab_df_fragmented
775 };

777 /*
778  * ==========================================================================
779  * Cursor fit block allocator -
780  * Select the largest region in the metaslab, set the cursor to the beginning
781  * of the range and the cursor_end to the end of the range. As allocations
782  * are made advance the cursor. Continue allocating from the cursor until
783  * the range is exhausted and then find a new range.
669  * Other experimental allocators
784  * ==========================================================================
785  */
786 static uint64_t
787 metaslab_cf_alloc(metaslab_t *msp, uint64_t size)
673 metaslab_cdf_alloc(space_map_t *sm, uint64_t size)
788 {
789         range_tree_t *rt = msp->ms_tree;
790         avl_tree_t *t = &msp->ms_size_tree;
791         uint64_t *cursor = &msp->ms_lbas[0];
792         uint64_t *cursor_end = &msp->ms_lbas[1];
675         avl_tree_t *t = &sm->sm_root;
676         uint64_t *cursor = (uint64_t *)sm->sm_ppd;
677         uint64_t *extent_end = (uint64_t *)sm->sm_ppd + 1;
678         uint64_t max_size = metaslab_pp_maxsize(sm);
679         uint64_t rsize = size;
793         uint64_t offset = 0;

795         ASSERT(MUTEX_HELD(&msp->ms_lock));
796         ASSERT3U(avl_numnodes(t), ==, avl_numnodes(&rt->rt_root));
682         ASSERT(MUTEX_HELD(sm->sm_lock));
683         ASSERT3U(avl_numnodes(&sm->sm_root), ==, avl_numnodes(sm->sm_pp_root));

798         ASSERT3U(*cursor_end, >=, *cursor);

800         if ((*cursor + size) > *cursor_end) {
801                 range_seg_t *rs;

803                 rs = avl_last(&msp->ms_size_tree);
804                 if (rs == NULL || (rs->rs_end - rs->rs_start) < size)
685         if (max_size < size)
805                         return (-1ULL);

807                 *cursor = rs->rs_start;
808                 *cursor_end = rs->rs_end;
809         }
688         ASSERT3U(*extent_end, >=, *cursor);

811         offset = *cursor;
812         *cursor += size;
690         /*
691          * If we're running low on space switch to using the size
692          * sorted AVL tree (best-fit).
```



```
693          */
694         if ((*cursor + size) > *extent_end) {

696                 t = sm->sm_pp_root;
697                 *cursor = *extent_end = 0;

699                 if (max_size > 2 * SPA_MAXBLOCKSIZE)
700                         rsize = MIN(metaslab_min_alloc_size, max_size);
701                 offset = metaslab_block_picker(t, extent_end, rsize, 1ULL);
702                 if (offset != -1)
703                         *cursor = offset + size;
704         } else {
705                 offset = metaslab_block_picker(t, cursor, rsize, 1ULL);
706         }
707         ASSERT3U(*cursor, <=, *extent_end);
814         return (offset);
815 }

817 static boolean_t
818 metaslab_cf_fragmented(metaslab_t *msp)
712 metaslab_cdf_fragmented(space_map_t *sm)
819 {
820         return (metaslab_block_maxsize(msp) < metaslab_min_alloc_size);
714         uint64_t max_size = metaslab_pp_maxsize(sm);

716         if (max_size > (metaslab_min_alloc_size * 10))
717                 return (B_FALSE);
718         return (B_TRUE);
821 }

823 static metaslab_ops_t metaslab_cf_ops = {
824         metaslab_cf_alloc,
825         metaslab_cf_fragmented
721 static space_map_ops_t metaslab_cdf_ops = {
722         metaslab_pp_load,
723         metaslab_pp_unload,
724         metaslab_cdf_alloc,
725         metaslab_pp_claim,
726         metaslab_pp_free,
727         metaslab_pp_maxsize,
728         metaslab_cdf_fragmented
826 };

828 /*
829  * ==========================================================================
830  * New dynamic fit allocator -
831  * Select a region that is large enough to allocate 2^metaslab_ndf_clump_shift
832  * contiguous blocks. If no region is found then just use the largest segment
833  * that remains.
834  * ==========================================================================
835  */

837 /*
838  * Determines desired number of contiguous blocks (2^metaslab_ndf_clump_shift)
839  * to request from the allocator.
840  */
841 uint64_t metaslab_ndf_clump_shift = 4;

843 static uint64_t
844 metaslab_ndf_alloc(metaslab_t *msp, uint64_t size)
734 metaslab_ndf_alloc(space_map_t *sm, uint64_t size)
845 {
846         avl_tree_t *t = &msp->ms_tree->rt_root;
736         avl_tree_t *t = &sm->sm_root;
847         avl_index_t where;
848         range_seg_t *rs, rsearch;
```

```
738         space_seg_t *ss, ssearch;
849         uint64_t hbit = highbit(size);
850         uint64_t *cursor = &msp->ms_lbas[hbit - 1];
851         uint64_t max_size = metaslab_block_maxsize(msp);
740         uint64_t *cursor = (uint64_t *)sm->sm_ppd + hbit - 1;
741         uint64_t max_size = metaslab_pp_maxsize(sm);

853         ASSERT(MUTEX_HELD(&msp->ms_lock));
854         ASSERT3U(avl_numnodes(t), ==, avl_numnodes(&msp->ms_size_tree));
743         ASSERT(MUTEX_HELD(sm->sm_lock));
744         ASSERT3U(avl_numnodes(&sm->sm_root), ==, avl_numnodes(sm->sm_pp_root));

856         if (max_size < size)
857                 return (-1ULL);

859         rsearch.rs_start = *cursor;
860         rsearch.rs_end = *cursor + size;
749         ssearch.ss_start = *cursor;
750         ssearch.ss_end = *cursor + size;

862         rs = avl_find(t, &rsearch, &where);
863         if (rs == NULL || (rs->rs_end - rs->rs_start) < size) {
864                 t = &msp->ms_size_tree;
752         ss = avl_find(t, &ssearch, &where);
753         if (ss == NULL || (ss->ss_start + size > ss->ss_end)) {
754                 t = sm->sm_pp_root;

866                 rsearch.rs_start = 0;
867                 rsearch.rs_end = MIN(max_size,
756                 ssearch.ss_start = 0;
757                 ssearch.ss_end = MIN(max_size,
868                     1ULL << (hbit + metaslab_ndf_clump_shift));
869                 rs = avl_find(t, &rsearch, &where);
870                 if (rs == NULL)
871                         rs = avl_nearest(t, where, AVL_AFTER);
872                 ASSERT(rs != NULL);
759                 ss = avl_find(t, &ssearch, &where);
760                 if (ss == NULL)
761                         ss = avl_nearest(t, where, AVL_AFTER);
762                 ASSERT(ss != NULL);
873         }

875         if ((rs->rs_end - rs->rs_start) >= size) {
876                 *cursor = rs->rs_start + size;
877                 return (rs->rs_start);
765         if (ss != NULL) {
766                 if (ss->ss_start + size <= ss->ss_end) {
767                         *cursor = ss->ss_start + size;
768                         return (ss->ss_start);
878         }
770         }
879         return (-1ULL);
880 }

882 static boolean_t
883 metaslab_ndf_fragmented(metaslab_t *msp)
775 metaslab_ndf_fragmented(space_map_t *sm)
884 {
885         return (metaslab_block_maxsize(msp) <=
886             (metaslab_min_alloc_size << metaslab_ndf_clump_shift));
777         uint64_t max_size = metaslab_pp_maxsize(sm);

779         if (max_size > (metaslab_min_alloc_size << metaslab_ndf_clump_shift))
780                 return (B_FALSE);
781         return (B_TRUE);
887 }
```



```
889 static metaslab_ops_t metaslab_ndf_ops = {

785 static space_map_ops_t metaslab_ndf_ops = {
786         metaslab_pp_load,
787         metaslab_pp_unload,
890         metaslab_ndf_alloc,
789         metaslab_pp_claim,
790         metaslab_pp_free,
791         metaslab_pp_maxsize,
891         metaslab_ndf_fragmented
892 };

894 metaslab_ops_t *zfs_metaslab_ops = &metaslab_df_ops;
795 space_map_ops_t *zfs_metaslab_ops = &metaslab_df_ops;

896 /*
897  * ==========================================================================
898  * Metaslabs
899  * ==========================================================================
900  */

902 /*
903  * Wait for any in-progress metaslab loads to complete.
904  */
905 void
906 metaslab_load_wait(metaslab_t *msp)
907 {
908         ASSERT(MUTEX_HELD(&msp->ms_lock));

910         while (msp->ms_loading) {
911                 ASSERT(!msp->ms_loaded);
912                 cv_wait(&msp->ms_load_cv, &msp->ms_lock);
913         }
914 }

916 int
917 metaslab_load(metaslab_t *msp)
918 {
919         int error = 0;

921         ASSERT(MUTEX_HELD(&msp->ms_lock));
922         ASSERT(!msp->ms_loaded);
923         ASSERT(!msp->ms_loading);

925         msp->ms_loading = B_TRUE;

927         /*
928          * If the space map has not been allocated yet, then treat
929          * all the space in the metaslab as free and add it to the
930          * ms_tree.
931          */
932         if (msp->ms_sm != NULL)
933                 error = space_map_load(msp->ms_sm, msp->ms_tree, SM_FREE);
934         else
935                 range_tree_add(msp->ms_tree, msp->ms_start, msp->ms_size);

937         msp->ms_loaded = (error == 0);
938         msp->ms_loading = B_FALSE;

940         if (msp->ms_loaded) {
941                 for (int t = 0; t < TXG_DEFER_SIZE; t++) {
942                         range_tree_walk(msp->ms_defertree[t],
943                             range_tree_remove, msp->ms_tree);
944                 }
945         }
```

```
946         cv_broadcast(&msp->ms_load_cv);
947         return (error);
948 }

950 void
951 metaslab_unload(metaslab_t *msp)
952 {
953         ASSERT(MUTEX_HELD(&msp->ms_lock));
954         range_tree_vacate(msp->ms_tree, NULL, NULL);
955         msp->ms_loaded = B_FALSE;
956         msp->ms_weight &= ~METASLAB_ACTIVE_MASK;
957 }

959 metaslab_t *
960 metaslab_init(metaslab_group_t *mg, uint64_t id, uint64_t object, uint64_t txg)
803 metaslab_init(metaslab_group_t *mg, space_map_obj_t *smo,
804         uint64_t start, uint64_t size, uint64_t txg)
961 {
962         vdev_t *vd = mg->mg_vd;
963         objset_t *mos = vd->vdev_spa->spa_meta_objset;
964         metaslab_t *msp;

966         msp = kmem_zalloc(sizeof (metaslab_t), KM_SLEEP);
967         mutex_init(&msp->ms_lock, NULL, MUTEX_DEFAULT, NULL);
968         cv_init(&msp->ms_load_cv, NULL, CV_DEFAULT, NULL);
969         msp->ms_id = id;
970         msp->ms_start = id << vd->vdev_ms_shift;
971         msp->ms_size = 1ULL << vd->vdev_ms_shift;

973         /*
974          * We only open space map objects that already exist. All others
975          * will be opened when we finally allocate an object for it.
976          */
977         if (object != 0) {
978                 VERIFY0(space_map_open(&msp->ms_sm, mos, object, msp->ms_start,
979                     msp->ms_size, vd->vdev_ashift, &msp->ms_lock));
980                 ASSERT(msp->ms_sm != NULL);
981         }
812         msp->ms_smo_syncing = *smo;

983         /*
984          * We create the main range tree here, but we don't create the
985          * alloctree and freetree until metaslab_sync_done().  This serves
815          * We create the main space map here, but we don't create the
816          * allocmaps and freemaps until metaslab_sync_done().  This serves
986          * two purposes: it allows metaslab_sync_done() to detect the
987          * addition of new space; and for debugging, it ensures that we'd
988          * data fault on any attempt to use this metaslab before it's ready.
989          */
990         msp->ms_tree = range_tree_create(&metaslab_rt_ops, msp, &msp->ms_lock);
821         msp->ms_map = kmem_zalloc(sizeof (space_map_t), KM_SLEEP);
822         space_map_create(msp->ms_map, start, size,
823             vd->vdev_ashift, &msp->ms_lock);

991         metaslab_group_add(mg, msp);

993         msp->ms_ops = mg->mg_class->mc_ops;
827         if (metaslab_debug && smo->smo_object != 0) {
828                 mutex_enter(&msp->ms_lock);
829                 VERIFY(space_map_load(msp->ms_map, mg->mg_class->mc_ops,
830                     SM_FREE, smo, spa_meta_objset(vd->vdev_spa)) == 0);
831                 mutex_exit(&msp->ms_lock);
832         }

995         /*
996          * If we're opening an existing pool (txg == 0) or creating
```



```
997          * a new one (txg == TXG_INITIAL), all space is available now.
998          * If we're adding space to an existing pool, the new space
999          * does not become available until after this txg has synced.
1000         */
1001        if (txg <= TXG_INITIAL)
1002                metaslab_sync_done(msp, 0);

1004        /*
1005         * If metaslab_debug_load is set and we're initializing a metaslab
1006         * that has an allocated space_map object then load the its space
1007         * map so that can verify frees.
1008         */
1009        if (metaslab_debug_load && msp->ms_sm != NULL) {
1010                mutex_enter(&msp->ms_lock);
1011                VERIFY0(metaslab_load(msp));
1012                mutex_exit(&msp->ms_lock);
1013        }

1015        if (txg != 0) {
1016                vdev_dirty(vd, 0, NULL, txg);
1017                vdev_dirty(vd, VDD_METASLAB, msp, txg);
1018        }

1020        return (msp);
1021 }

1023 void
1024 metaslab_fini(metaslab_t *msp)
1025 {
1026        metaslab_group_t *mg = msp->ms_group;

 856        vdev_space_update(mg->mg_vd,
 857            -msp->ms_smo.smo_alloc, 0, -msp->ms_map->sm_size);

1028        metaslab_group_remove(mg, msp);

1030        mutex_enter(&msp->ms_lock);

1032        VERIFY(msp->ms_group == NULL);
1033        vdev_space_update(mg->mg_vd, -space_map_allocated(msp->ms_sm),
1034            0, -msp->ms_size);
1035        space_map_close(msp->ms_sm);
 863        space_map_unload(msp->ms_map);
 864        space_map_destroy(msp->ms_map);
 865        kmem_free(msp->ms_map, sizeof (*msp->ms_map));

1037        metaslab_unload(msp);
1038        range_tree_destroy(msp->ms_tree);

1040        for (int t = 0; t < TXG_SIZE; t++) {
1041                range_tree_destroy(msp->ms_alloctree[t]);
1042                range_tree_destroy(msp->ms_freetree[t]);
 868                space_map_destroy(msp->ms_allocmap[t]);
 869                space_map_destroy(msp->ms_freemap[t]);
 870                kmem_free(msp->ms_allocmap[t], sizeof (*msp->ms_allocmap[t]));
 871                kmem_free(msp->ms_freemap[t], sizeof (*msp->ms_freemap[t]));
1043        }

1045        for (int t = 0; t < TXG_DEFER_SIZE; t++) {
1046                range_tree_destroy(msp->ms_defertree[t]);
 875                space_map_destroy(msp->ms_defermap[t]);
 876                kmem_free(msp->ms_defermap[t], sizeof (*msp->ms_defermap[t]));
1047        }

1049        ASSERT0(msp->ms_deferspace);
```

```
1051         mutex_exit(&msp->ms_lock);
1052         cv_destroy(&msp->ms_load_cv);
1053         mutex_destroy(&msp->ms_lock);

1055         kmem_free(msp, sizeof (metaslab_t));
1056 }

1058 /*
1059  * Apply a weighting factor based on the histogram information for this
1060  * metaslab. The current weighting factor is somewhat arbitrary and requires
1061  * additional investigation. The implementation provides a measure of
1062  * "weighted" free space and gives a higher weighting for larger contiguous
1063  * regions. The weighting factor is determined by counting the number of
1064  * sm_shift sectors that exist in each region represented by the histogram.
1065  * That value is then multiplied by the power of 2 exponent and the sm_shift
1066  * value.
1067  *
1068  * For example, assume the 2^21 histogram bucket has 4 2MB regions and the
1069  * metaslab has an sm_shift value of 9 (512B):
1070  *
1071  * 1) calculate the number of sm_shift sectors in the region:
1072  *      2^21 / 2^9 = 2^12 = 4096 * 4 (number of regions) = 16384
1073  * 2) multiply by the power of 2 exponent and the sm_shift value:
1074  *      16384 * 21 * 9 = 3096576
1075  * This value will be added to the weighting of the metaslab.
1076  */
1077 static uint64_t
1078 metaslab_weight_factor(metaslab_t *msp)
1079 {
1080         uint64_t factor = 0;
1081         uint64_t sectors;
1082         int i;
 887 #define METASLAB_WEIGHT_PRIMARY         (1ULL << 63)
 888 #define METASLAB_WEIGHT_SECONDARY       (1ULL << 62)
 889 #define METASLAB_ACTIVE_MASK            \
 890         (METASLAB_WEIGHT_PRIMARY | METASLAB_WEIGHT_SECONDARY)

1084         /*
1085          * A null space map means that the entire metaslab is free,
1086          * calculate a weight factor that spans the entire size of the
1087          * metaslab.
1088          */
1089         if (msp->ms_sm == NULL) {
1090                 vdev_t *vd = msp->ms_group->mg_vd;

1092                 i = highbit(msp->ms_size) - 1;
1093                 sectors = msp->ms_size >> vd->vdev_ashift;
1094                 return (sectors * i * vd->vdev_ashift);
1095         }

1097         if (msp->ms_sm->sm_dbuf->db_size != sizeof (space_map_phys_t))
1098                 return (0);

1100         for (i = 0; i < SPACE_MAP_HISTOGRAM_SIZE(msp->ms_sm); i++) {
1101                 if (msp->ms_sm->sm_phys->smp_histogram[i] == 0)
1102                         continue;

1104                 /*
1105                  * Determine the number of sm_shift sectors in the region
1106                  * indicated by the histogram. For example, given an
1107                  * sm_shift value of 9 (512 bytes) and i = 4 then we know
1108                  * that we're looking at an 8K region in the histogram
1109                  * (i.e. 9 + 4 = 13, 2^13 = 8192). To figure out the
1110                  * number of sm_shift sectors (512 bytes in this example),
1111                  * we would take 8192 / 512 = 16. Since the histogram
1112                  * is offset by sm_shift we can simply use the value of
```



```
1113                  * of i to calculate this (i.e. 2^i = 16 where i = 4).
1114                  */
1115                 sectors = msp->ms_sm->sm_phys->smp_histogram[i] << i;
1116                 factor += (i + msp->ms_sm->sm_shift) * sectors;
1117         }
1118         return (factor * msp->ms_sm->sm_shift);
1119 }

1121 static uint64_t
1122 metaslab_weight(metaslab_t *msp)
1123 {
1124         metaslab_group_t *mg = msp->ms_group;
 896         space_map_t *sm = msp->ms_map;
 897         space_map_obj_t *smo = &msp->ms_smo;
1125         vdev_t *vd = mg->mg_vd;
1126         uint64_t weight, space;

1128         ASSERT(MUTEX_HELD(&msp->ms_lock));

1130         /*
1131          * This vdev is in the process of being removed so there is nothing
1132          * for us to do here.
1133          */
1134         if (vd->vdev_removing) {
1135                 ASSERT0(space_map_allocated(msp->ms_sm));
 908                 ASSERT0(smo->smo_alloc);
1136                 ASSERT0(vd->vdev_ms_shift);
1137                 return (0);
1138         }

1140         /*
1141          * The baseline weight is the metaslab's free space.
1142          */
1143         space = msp->ms_size - space_map_allocated(msp->ms_sm);
 916         space = sm->sm_size - smo->smo_alloc;
1144         weight = space;

1146         /*
1147          * Modern disks have uniform bit density and constant angular velocity.
1148          * Therefore, the outer recording zones are faster (higher bandwidth)
1149          * than the inner zones by the ratio of outer to inner track diameter,
1150          * which is typically around 2:1.  We account for this by assigning
1151          * higher weight to lower metaslabs (multiplier ranging from 2x to 1x).
1152          * In effect, this means that we'll select the metaslab with the most
1153          * free bandwidth rather than simply the one with the most free space.
1154          */
1155         weight = 2 * weight - (msp->ms_id * weight) / vd->vdev_ms_count;
 928         weight = 2 * weight -
 929             ((sm->sm_start >> vd->vdev_ms_shift) * weight) / vd->vdev_ms_count;
1156         ASSERT(weight >= space && weight <= 2 * space);

1158         msp->ms_factor = metaslab_weight_factor(msp);
1159         if (metaslab_weight_factor_enable)
1160                 weight += msp->ms_factor;
 932         /*
 933          * For locality, assign higher weight to metaslabs which have
 934          * a lower offset than what we've already activated.
 935          */
 936         if (sm->sm_start <= mg->mg_bonus_area)
 937                 weight *= (metaslab_smo_bonus_pct / 100);
 938         ASSERT(weight >= space &&
 939             weight <= 2 * (metaslab_smo_bonus_pct / 100) * space);

1162         if (msp->ms_loaded && !msp->ms_ops->msop_fragmented(msp)) {
 941         if (sm->sm_loaded && !sm->sm_ops->smop_fragmented(sm)) {
1163                 /*
```

```
1164                  * If this metaslab is one we're actively using, adjust its
1165                  * weight to make it preferable to any inactive metaslab so
1166                  * we'll polish it off.
1167                  */
1168                 weight |= (msp->ms_weight & METASLAB_ACTIVE_MASK);
1169         }

1171         return (weight);
1172 }

 952 static void
 953 metaslab_prefetch(metaslab_group_t *mg)
 954 {
 955         spa_t *spa = mg->mg_vd->vdev_spa;
 956         metaslab_t *msp;
 957         avl_tree_t *t = &mg->mg_metaslab_tree;
 958         int m;

 960         mutex_enter(&mg->mg_lock);

 962         /*
 963          * Prefetch the next potential metaslabs
 964          */
 965         for (msp = avl_first(t), m = 0; msp; msp = AVL_NEXT(t, msp), m++) {
 966                 space_map_t *sm = msp->ms_map;
 967                 space_map_obj_t *smo = &msp->ms_smo;

 969                 /* If we have reached our prefetch limit then we're done */
 970                 if (m >= metaslab_prefetch_limit)
 971                         break;

 973                 if (!sm->sm_loaded && smo->smo_object != 0) {
 974                         mutex_exit(&mg->mg_lock);
 975                         dmu_prefetch(spa_meta_objset(spa), smo->smo_object,
 976                             0ULL, smo->smo_objsize);
 977                         mutex_enter(&mg->mg_lock);
 978                 }
 979         }
 980         mutex_exit(&mg->mg_lock);
 981 }

1174 static int
1175 metaslab_activate(metaslab_t *msp, uint64_t activation_weight)
1176 {
 986         metaslab_group_t *mg = msp->ms_group;
 987         space_map_t *sm = msp->ms_map;
 988         space_map_ops_t *sm_ops = msp->ms_group->mg_class->mc_ops;

1177         ASSERT(MUTEX_HELD(&msp->ms_lock));

1179         if ((msp->ms_weight & METASLAB_ACTIVE_MASK) == 0) {
1180                 metaslab_load_wait(msp);
1181                 if (!msp->ms_loaded) {
1182                         int error = metaslab_load(msp);
 993                 space_map_load_wait(sm);
 994                 if (!sm->sm_loaded) {
 995                         space_map_obj_t *smo = &msp->ms_smo;

 997                         int error = space_map_load(sm, sm_ops, SM_FREE, smo,
 998                             spa_meta_objset(msp->ms_group->mg_vd->vdev_spa));
1183                         if (error) {
1184                                 metaslab_group_sort(msp->ms_group, msp, 0);
1185                                 return (error);
1186                         }
1003                         for (int t = 0; t < TXG_DEFER_SIZE; t++)
1004                                 space_map_walk(msp->ms_defermap[t],
```



```
1005                                     space_map_claim, sm);

1187                 }

1009                 /*
1010                  * Track the bonus area as we activate new metaslabs.
1011                  */
1012                 if (sm->sm_start > mg->mg_bonus_area) {
1013                         mutex_enter(&mg->mg_lock);
1014                         mg->mg_bonus_area = sm->sm_start;
1015                         mutex_exit(&mg->mg_lock);
1016                 }

1189                 metaslab_group_sort(msp->ms_group, msp,
1190                     msp->ms_weight | activation_weight);
1191         }
1192         ASSERT(msp->ms_loaded);
1021         ASSERT(sm->sm_loaded);
1193         ASSERT(msp->ms_weight & METASLAB_ACTIVE_MASK);

1195         return (0);
1196 }

1198 static void
1199 metaslab_passivate(metaslab_t *msp, uint64_t size)
1200 {
1201         /*
1202          * If size < SPA_MINBLOCKSIZE, then we will not allocate from
1203          * this metaslab again.  In that case, it had better be empty,
1204          * or we would be leaving space on the table.
1205          */
1206         ASSERT(size >= SPA_MINBLOCKSIZE || range_tree_space(msp->ms_tree) == 0);
1035         ASSERT(size >= SPA_MINBLOCKSIZE || msp->ms_map->sm_space == 0);
1207         metaslab_group_sort(msp->ms_group, msp, MIN(msp->ms_weight, size));
1208         ASSERT((msp->ms_weight & METASLAB_ACTIVE_MASK) == 0);
1209 }

1211 static void
1212 metaslab_preload(void *arg)
1213 {
1214         metaslab_t *msp = arg;
1215         spa_t *spa = msp->ms_group->mg_vd->vdev_spa;

1217         mutex_enter(&msp->ms_lock);
1218         metaslab_load_wait(msp);
1219         if (!msp->ms_loaded)
1220                 (void) metaslab_load(msp);

1222         /*
1223          * Set the ms_access_txg value so that we don't unload it right away.
1224          */
1225         msp->ms_access_txg = spa_syncing_txg(spa) + metaslab_unload_delay + 1;
1226         mutex_exit(&msp->ms_lock);
1227 }

1229 static void
1230 metaslab_group_preload(metaslab_group_t *mg)
1231 {
1232         spa_t *spa = mg->mg_vd->vdev_spa;
1233         metaslab_t *msp;
1234         avl_tree_t *t = &mg->mg_metaslab_tree;
1235         int m = 0;

1237         if (spa_shutting_down(spa) || !metaslab_preload_enabled) {
1238                 taskq_wait(mg->mg_taskq);
1239                 return;
```

```
1240         }
1241         mutex_enter(&mg->mg_lock);

1243         /*
1244          * Prefetch the next potential metaslabs
1245          */
1246         for (msp = avl_first(t); msp != NULL; msp = AVL_NEXT(t, msp)) {

1248                 /* If we have reached our preload limit then we're done */
1249                 if (++m > metaslab_preload_limit)
1250                         break;

1252                 VERIFY(taskq_dispatch(mg->mg_taskq, metaslab_preload,
1253                     msp, TQ_SLEEP) != NULL);
1254         }
1255         mutex_exit(&mg->mg_lock);
1256 }

1258 /*
1259  * Determine if the space map's on-disk footprint is past our tolerance
1260  * for inefficiency. We would like to use the following criteria to make
1261  * our decision:
1041  * Determine if the in-core space map representation can be condensed on-disk.
1042  * We would like to use the following criteria to make our decision:
1262  *
1263  * 1. The size of the space map object should not dramatically increase as a
1264  * result of writing out the free space range tree.
1045  * result of writing out our in-core free map.
1265  *
1266  * 2. The minimal on-disk space map representation is zfs_condense_pct/100
1267  * times the size than the free space range tree representation
1268  * (i.e. zfs_condense_pct = 110 and in-core = 1MB, minimal = 1.1.MB).
1048  * times the size than the in-core representation (i.e. zfs_condense_pct = 110
1049  * and in-core = 1MB, minimal = 1.1.MB).
1269  *
1270  * Checking the first condition is tricky since we don't want to walk
1271  * the entire AVL tree calculating the estimated on-disk size. Instead we
1272  * use the size-ordered range tree in the metaslab and calculate the
1273  * size required to write out the largest segment in our free tree. If the
1053  * use the size-ordered AVL tree in the space map and calculate the
1054  * size required for the largest segment in our in-core free map. If the
1274  * size required to represent that segment on disk is larger than the space
1275  * map object then we avoid condensing this map.
1276  *
1277  * To determine the second criterion we use a best-case estimate and assume
1278  * each segment can be represented on-disk as a single 64-bit entry. We refer
1279  * to this best-case estimate as the space map's minimal form.
1280  */
1281 static boolean_t
1282 metaslab_should_condense(metaslab_t *msp)
1283 {
1284         space_map_t *sm = msp->ms_sm;
1285         range_seg_t *rs;
1065         space_map_t *sm = msp->ms_map;
1066         space_map_obj_t *smo = &msp->ms_smo_syncing;
1067         space_seg_t *ss;
1286         uint64_t size, entries, segsz;

1288         ASSERT(MUTEX_HELD(&msp->ms_lock));
1289         ASSERT(msp->ms_loaded);
1071         ASSERT(sm->sm_loaded);

1291         /*
1292          * Use the ms_size_tree range tree, which is ordered by size, to
1293          * obtain the largest segment in the free tree. If the tree is empty
1294          * then we should condense the map.
```



```
1074          * Use the sm_pp_root AVL tree, which is ordered by size, to obtain
1075          * the largest segment in the in-core free map. If the tree is
1076          * empty then we should condense the map.
1295          */
1296         rs = avl_last(&msp->ms_size_tree);
1297         if (rs == NULL)
1078         ss = avl_last(sm->sm_pp_root);
1079         if (ss == NULL)
1298                 return (B_TRUE);

1300         /*
1301          * Calculate the number of 64-bit entries this segment would
1302          * require when written to disk. If this single segment would be
1303          * larger on-disk than the entire current on-disk structure, then
1304          * clearly condensing will increase the on-disk structure size.
1305          */
1306         size = (rs->rs_end - rs->rs_start) >> sm->sm_shift;
1088         size = (ss->ss_end - ss->ss_start) >> sm->sm_shift;
1307         entries = size / (MIN(size, SM_RUN_MAX));
1308         segsz = entries * sizeof (uint64_t);

1310         return (segsz <= space_map_length(msp->ms_sm) &&
1311             space_map_length(msp->ms_sm) >= (zfs_condense_pct *
1312             sizeof (uint64_t) * avl_numnodes(&msp->ms_tree->rt_root)) / 100);
1092         return (segsz <= smo->smo_objsize &&
1093             smo->smo_objsize >= (zfs_condense_pct *
1094             sizeof (uint64_t) * avl_numnodes(&sm->sm_root)) / 100);
1313 }

1315 /*
1316  * Condense the on-disk space map representation to its minimized form.
1317  * The minimized form consists of a small number of allocations followed by
1318  * the entries of the free range tree.
1100  * the in-core free map.
1319  */
1320 static void
1321 metaslab_condense(metaslab_t *msp, uint64_t txg, dmu_tx_t *tx)
1322 {
1323         spa_t *spa = msp->ms_group->mg_vd->vdev_spa;
1324         range_tree_t *freetree = msp->ms_freetree[txg & TXG_MASK];
1325         range_tree_t *condense_tree;
1326         space_map_t *sm = msp->ms_sm;
1106         space_map_t *freemap = msp->ms_freemap[txg & TXG_MASK];
1107         space_map_t condense_map;
1108         space_map_t *sm = msp->ms_map;
1109         objset_t *mos = spa_meta_objset(spa);
1110         space_map_obj_t *smo = &msp->ms_smo_syncing;

1328         ASSERT(MUTEX_HELD(&msp->ms_lock));
1329         ASSERT3U(spa_sync_pass(spa), ==, 1);
1330         ASSERT(msp->ms_loaded);
1114         ASSERT(sm->sm_loaded);

1332         spa_dbgmsg(spa, "condensing: txg %llu, msp[%llu] %p, "
1333             "smp size %llu, segments %lu", txg, msp->ms_id, msp,
1334             space_map_length(msp->ms_sm), avl_numnodes(&msp->ms_tree->rt_root));
1117             "smo size %llu, segments %lu", txg,
1118             (msp->ms_map->sm_start / msp->ms_map->sm_size), msp,
1119             smo->smo_objsize, avl_numnodes(&sm->sm_root));

1336         /*
1337          * Create an range tree that is 100% allocated. We remove segments
1122          * Create an map that is a 100% allocated map. We remove segments
1338          * that have been freed in this txg, any deferred frees that exist,
1339          * and any allocation in the future. Removing segments should be
1340          * a relatively inexpensive operation since we expect these trees to
```

```
1341          * have a small number of nodes.
1125          * a relatively inexpensive operation since we expect these maps to
1126          * a small number of nodes.
1342          */
1343         condense_tree = range_tree_create(NULL, NULL, &msp->ms_lock);
1344         range_tree_add(condense_tree, msp->ms_start, msp->ms_size);
1128         space_map_create(&condense_map, sm->sm_start, sm->sm_size,
1129             sm->sm_shift, sm->sm_lock);
1130         space_map_add(&condense_map, condense_map.sm_start,
1131             condense_map.sm_size);

1346         /*
1347          * Remove what's been freed in this txg from the condense_tree.
1134          * Remove what's been freed in this txg from the condense_map.
1348          * Since we're in sync_pass 1, we know that all the frees from
1349          * this txg are in the freetree.
1136          * this txg are in the freemap.
1350          */
1351         range_tree_walk(freetree, range_tree_remove, condense_tree);
1138         space_map_walk(freemap, space_map_remove, &condense_map);

1353         for (int t = 0; t < TXG_DEFER_SIZE; t++) {
1354                 range_tree_walk(msp->ms_defertree[t],
1355                     range_tree_remove, condense_tree);
1356         }
1140         for (int t = 0; t < TXG_DEFER_SIZE; t++)
1141                 space_map_walk(msp->ms_defermap[t],
1142                     space_map_remove, &condense_map);

1358         for (int t = 1; t < TXG_CONCURRENT_STATES; t++) {
1359                 range_tree_walk(msp->ms_alloctree[(txg + t) & TXG_MASK],
1360                     range_tree_remove, condense_tree);
1361         }
1144         for (int t = 1; t < TXG_CONCURRENT_STATES; t++)
1145                 space_map_walk(msp->ms_allocmap[(txg + t) & TXG_MASK],
1146                     space_map_remove, &condense_map);

1363         /*
1364          * We're about to drop the metaslab's lock thus allowing
1365          * other consumers to change it's content. Set the
1366          * metaslab's ms_condensing flag to ensure that
1151          * space_map's sm_condensing flag to ensure that
1367          * allocations on this metaslab do not occur while we're
1368          * in the middle of committing it to disk. This is only critical
1369          * for the ms_tree as all other range trees use per txg
1154          * for the ms_map as all other space_maps use per txg
1370          * views of their content.
1371          */
1372         msp->ms_condensing = B_TRUE;
1157         sm->sm_condensing = B_TRUE;

1374         mutex_exit(&msp->ms_lock);
1375         space_map_truncate(sm, tx);
1160         space_map_truncate(smo, mos, tx);
1376         mutex_enter(&msp->ms_lock);

1378         /*
1379          * While we would ideally like to create a space_map representation
1380          * that consists only of allocation records, doing so can be
1381          * prohibitively expensive because the in-core free tree can be
1166          * prohibitively expensive because the in-core free map can be
1382          * large, and therefore computationally expensive to subtract
1383          * from the condense_tree. Instead we sync out two trees, a cheap
1384          * allocation only tree followed by the in-core free tree. While not
1168          * from the condense_map. Instead we sync out two maps, a cheap
1169          * allocation only map followed by the in-core free map. While not
```



```
1385          * optimal, this is typically close to optimal, and much cheaper to
1386          * compute.
1387          */
1388         space_map_write(sm, condense_tree, SM_ALLOC, tx);
1389         range_tree_vacate(condense_tree, NULL, NULL);
1390         range_tree_destroy(condense_tree);
1173         space_map_sync(&condense_map, SM_ALLOC, smo, mos, tx);
1174         space_map_vacate(&condense_map, NULL, NULL);
1175         space_map_destroy(&condense_map);

1392         space_map_write(sm, msp->ms_tree, SM_FREE, tx);
1393         msp->ms_condensing = B_FALSE;
1177         space_map_sync(sm, SM_FREE, smo, mos, tx);
1178         sm->sm_condensing = B_FALSE;

1180         spa_dbgmsg(spa, "condensed: txg %llu, msp[%llu] %p, "
1181             "smo size %llu", txg,
1182             (msp->ms_map->sm_start / msp->ms_map->sm_size), msp,
1183             smo->smo_objsize);
1394 }

1396 /*
1397  * Write a metaslab to disk in the context of the specified transaction group.
1398  */
1399 void
1400 metaslab_sync(metaslab_t *msp, uint64_t txg)
1401 {
1402         metaslab_group_t *mg = msp->ms_group;
1403         vdev_t *vd = mg->mg_vd;
1192         vdev_t *vd = msp->ms_group->mg_vd;
1404         spa_t *spa = vd->vdev_spa;
1405         objset_t *mos = spa_meta_objset(spa);
1406         range_tree_t *alloctree = msp->ms_alloctree[txg & TXG_MASK];
1407         range_tree_t **freetree = &msp->ms_freetree[txg & TXG_MASK];
1408         range_tree_t **freed_tree =
1409             &msp->ms_freetree[TXG_CLEAN(txg) & TXG_MASK];
1195         space_map_t *allocmap = msp->ms_allocmap[txg & TXG_MASK];
1196         space_map_t **freemap = &msp->ms_freemap[txg & TXG_MASK];
1197         space_map_t **freed_map = &msp->ms_freemap[TXG_CLEAN(txg) & TXG_MASK];
1198         space_map_t *sm = msp->ms_map;
1199         space_map_obj_t *smo = &msp->ms_smo_syncing;
1200         dmu_buf_t *db;
1410         dmu_tx_t *tx;
1411         uint64_t object = space_map_object(msp->ms_sm);

1413         ASSERT(!vd->vdev_ishole);

1415         /*
1416          * This metaslab has just been added so there's no work to do now.
1417          */
1418         if (*freetree == NULL) {
1419                 ASSERT3P(alloctree, ==, NULL);
1208         if (*freemap == NULL) {
1209                 ASSERT3P(allocmap, ==, NULL);
1420                 return;
1421         }

1423         ASSERT3P(alloctree, !=, NULL);
1424         ASSERT3P(*freetree, !=, NULL);
1425         ASSERT3P(*freed_tree, !=, NULL);
1213         ASSERT3P(allocmap, !=, NULL);
1214         ASSERT3P(*freemap, !=, NULL);
1215         ASSERT3P(*freed_map, !=, NULL);

1427         if (range_tree_space(alloctree) == 0 &&
1428             range_tree_space(*freetree) == 0)
```

```
1217         if (allocmap->sm_space == 0 && (*freemap)->sm_space == 0)
1429                 return;

1431         /*
1432          * The only state that can actually be changing concurrently with
1433          * metaslab_sync() is the metaslab's ms_tree.  No other thread can
1434          * be modifying this txg's alloctree, freetree, freed_tree, or
1435          * space_map_phys_t. Therefore, we only hold ms_lock to satify
1436          * space_map ASSERTs. We drop it whenever we call into the DMU,
1437          * because the DMU can call down to us (e.g. via zio_free()) at
1438          * any time.
1222          * metaslab_sync() is the metaslab's ms_map.  No other thread can
1223          * be modifying this txg's allocmap, freemap, freed_map, or smo.
1224          * Therefore, we only hold ms_lock to satify space_map ASSERTs.
1225          * We drop it whenever we call into the DMU, because the DMU
1226          * can call down to us (e.g. via zio_free()) at any time.
1439          */

1441         tx = dmu_tx_create_assigned(spa_get_dsl(spa), txg);

1443         if (msp->ms_sm == NULL) {
1444                 uint64_t new_object;

1446                 new_object = space_map_alloc(mos, tx);
1447                 VERIFY3U(new_object, !=, 0);

1449                 VERIFY0(space_map_open(&msp->ms_sm, mos, new_object,
1450                     msp->ms_start, msp->ms_size, vd->vdev_ashift,
1451                     &msp->ms_lock));
1452                 ASSERT(msp->ms_sm != NULL);
1231         if (smo->smo_object == 0) {
1232                 ASSERT(smo->smo_objsize == 0);
1233                 ASSERT(smo->smo_alloc == 0);
1234                 smo->smo_object = dmu_object_alloc(mos,
1235                     DMU_OT_SPACE_MAP, 1 << SPACE_MAP_BLOCKSHIFT,
1236                     DMU_OT_SPACE_MAP_HEADER, sizeof (*smo), tx);
1237                 ASSERT(smo->smo_object != 0);
1238                 dmu_write(mos, vd->vdev_ms_array, sizeof (uint64_t) *
1239                     (sm->sm_start >> vd->vdev_ms_shift),
1240                     sizeof (uint64_t), &smo->smo_object, tx);
1453         }

1455         mutex_enter(&msp->ms_lock);

1457         if (msp->ms_loaded && spa_sync_pass(spa) == 1 &&
1245         if (sm->sm_loaded && spa_sync_pass(spa) == 1 &&
1458             metaslab_should_condense(msp)) {
1459                 metaslab_condense(msp, txg, tx);
1460         } else {
1461                 space_map_write(msp->ms_sm, alloctree, SM_ALLOC, tx);
1462                 space_map_write(msp->ms_sm, *freetree, SM_FREE, tx);
1249                 space_map_sync(allocmap, SM_ALLOC, smo, mos, tx);
1250                 space_map_sync(*freemap, SM_FREE, smo, mos, tx);
1463         }

1465         range_tree_vacate(alloctree, NULL, NULL);
1253         space_map_vacate(allocmap, NULL, NULL);

1467         if (msp->ms_loaded) {
1468                 /*
1469                  * When the space map is loaded, we have an accruate
1470                  * histogram in the range tree. This gives us an opportunity
1471                  * to bring the space map's histogram up-to-date so we clear
1472                  * it first before updating it.
1473                  */
1474                 space_map_histogram_clear(msp->ms_sm);
```



```
1475                 space_map_histogram_add(msp->ms_sm, msp->ms_tree, tx);
1476         } else {
1477                 /*
1478                  * Since the space map is not loaded we simply update the
1479                  * exisiting histogram with what was freed in this txg. This
1480                  * means that the on-disk histogram may not have an accurate
1481                  * view of the free space but it's close enough to allow
1482                  * us to make allocation decisions.
1483                  */
1484                 space_map_histogram_add(msp->ms_sm, *freetree, tx);
1485         }

1487         /*
1488          * For sync pass 1, we avoid traversing this txg's free range tree
1489          * and instead will just swap the pointers for freetree and
1490          * freed_tree. We can safely do this since the freed_tree is
1256          * For sync pass 1, we avoid walking the entire space map and
1257          * instead will just swap the pointers for freemap and
1258          * freed_map. We can safely do this since the freed_map is
1491          * guaranteed to be empty on the initial pass.
1492          */
1493         if (spa_sync_pass(spa) == 1) {
1494                 range_tree_swap(freetree, freed_tree);
1262                 ASSERT0((*freed_map)->sm_space);
1263                 ASSERT0(avl_numnodes(&(*freed_map)->sm_root));
1264                 space_map_swap(freemap, freed_map);
1495         } else {
1496                 range_tree_vacate(*freetree, range_tree_add, *freed_tree);
1266                 space_map_vacate(*freemap, space_map_add, *freed_map);
1497         }

1499         ASSERT0(range_tree_space(msp->ms_alloctree[txg & TXG_MASK]));
1500         ASSERT0(range_tree_space(msp->ms_freetree[txg & TXG_MASK]));
1269         ASSERT0(msp->ms_allocmap[txg & TXG_MASK]->sm_space);
1270         ASSERT0(msp->ms_freemap[txg & TXG_MASK]->sm_space);

1502         mutex_exit(&msp->ms_lock);

1504         if (object != space_map_object(msp->ms_sm)) {
1505                 object = space_map_object(msp->ms_sm);
1506                 dmu_write(mos, vd->vdev_ms_array, sizeof (uint64_t) *
1507                     msp->ms_id, sizeof (uint64_t), &object, tx);
1508         }
1274         VERIFY0(dmu_bonus_hold(mos, smo->smo_object, FTAG, &db));
1275         dmu_buf_will_dirty(db, tx);
1276         ASSERT3U(db->db_size, >=, sizeof (*smo));
1277         bcopy(smo, db->db_data, sizeof (*smo));
1278         dmu_buf_rele(db, FTAG);

1509         dmu_tx_commit(tx);
1510 }

1512 /*
1513  * Called after a transaction group has completely synced to mark
1514  * all of the metaslab's free space as usable.
1515  */
1516 void
1517 metaslab_sync_done(metaslab_t *msp, uint64_t txg)
1518 {
1290         space_map_obj_t *smo = &msp->ms_smo;
1291         space_map_obj_t *smosync = &msp->ms_smo_syncing;
1292         space_map_t *sm = msp->ms_map;
1293         space_map_t **freed_map = &msp->ms_freemap[TXG_CLEAN(txg) & TXG_MASK];
1294         space_map_t **defer_map = &msp->ms_defermap[txg % TXG_DEFER_SIZE];
1519         metaslab_group_t *mg = msp->ms_group;
1520         vdev_t *vd = mg->mg_vd;
```

```
1521         range_tree_t **freed_tree;
1522         range_tree_t **defer_tree;
1523         int64_t alloc_delta, defer_delta;

1525         ASSERT(!vd->vdev_ishole);

1527         mutex_enter(&msp->ms_lock);

1529         /*
1530          * If this metaslab is just becoming available, initialize its
1531          * alloctrees, freetrees, and defertree and add its capacity to
1532          * the vdev.
1305          * allocmaps, freemaps, and defermap and add its capacity to the vdev.
1533          */
1534         if (msp->ms_freetree[TXG_CLEAN(txg) & TXG_MASK] == NULL) {
1307         if (*freed_map == NULL) {
1308                 ASSERT(*defer_map == NULL);
1535                 for (int t = 0; t < TXG_SIZE; t++) {
1536                         ASSERT(msp->ms_alloctree[t] == NULL);
1537                         ASSERT(msp->ms_freetree[t] == NULL);

1539                         msp->ms_alloctree[t] = range_tree_create(NULL, msp,
1540                             &msp->ms_lock);
1541                         msp->ms_freetree[t] = range_tree_create(NULL, msp,
1542                             &msp->ms_lock);
1310                         msp->ms_allocmap[t] = kmem_zalloc(sizeof (space_map_t),
1311                             KM_SLEEP);
1312                         space_map_create(msp->ms_allocmap[t], sm->sm_start,
1313                             sm->sm_size, sm->sm_shift, sm->sm_lock);
1314                         msp->ms_freemap[t] = kmem_zalloc(sizeof (space_map_t),
1315                             KM_SLEEP);
1316                         space_map_create(msp->ms_freemap[t], sm->sm_start,
1317                             sm->sm_size, sm->sm_shift, sm->sm_lock);
1543                 }

1545                 for (int t = 0; t < TXG_DEFER_SIZE; t++) {
1546                         ASSERT(msp->ms_defertree[t] == NULL);

1548                         msp->ms_defertree[t] = range_tree_create(NULL, msp,
1549                             &msp->ms_lock);
1321                         msp->ms_defermap[t] = kmem_zalloc(sizeof (space_map_t),
1322                             KM_SLEEP);
1323                         space_map_create(msp->ms_defermap[t], sm->sm_start,
1324                             sm->sm_size, sm->sm_shift, sm->sm_lock);
1550                 }

1552                 vdev_space_update(vd, 0, 0, msp->ms_size);
1327                 freed_map = &msp->ms_freemap[TXG_CLEAN(txg) & TXG_MASK];
1328                 defer_map = &msp->ms_defermap[txg % TXG_DEFER_SIZE];

1330                 vdev_space_update(vd, 0, 0, sm->sm_size);
1553         }

1555         freed_tree = &msp->ms_freetree[TXG_CLEAN(txg) & TXG_MASK];
1556         defer_tree = &msp->ms_defertree[txg % TXG_DEFER_SIZE];
1333         alloc_delta = smosync->smo_alloc - smo->smo_alloc;
1334         defer_delta = (*freed_map)->sm_space - (*defer_map)->sm_space;

1558         alloc_delta = space_map_alloc_delta(msp->ms_sm);
1559         defer_delta = range_tree_space(*freed_tree) -
1560             range_tree_space(*defer_tree);

1562         vdev_space_update(vd, alloc_delta + defer_delta, defer_delta, 0);

1564         ASSERT0(range_tree_space(msp->ms_alloctree[txg & TXG_MASK]));
1565         ASSERT0(range_tree_space(msp->ms_freetree[txg & TXG_MASK]));
```



```
1338         ASSERT(msp->ms_allocmap[txg & TXG_MASK]->sm_space == 0);
1339         ASSERT(msp->ms_freemap[txg & TXG_MASK]->sm_space == 0);

1567         /*
1568          * If there's a metaslab_load() in progress, wait for it to complete
1342          * If there's a space_map_load() in progress, wait for it to complete
1569          * so that we have a consistent view of the in-core space map.
1570          */
1571         metaslab_load_wait(msp);
1345         space_map_load_wait(sm);

1573         /*
1574          * Move the frees from the defer_tree back to the free
1575          * range tree (if it's loaded). Swap the freed_tree and the
1576          * defer_tree -- this is safe to do because we've just emptied out
1577          * the defer_tree.
1348          * Move the frees from the defer_map to this map (if it's loaded).
1349          * Swap the freed_map and the defer_map -- this is safe to do
1350          * because we've just emptied out the defer_map.
1578          */
1579         range_tree_vacate(*defer_tree,
1580             msp->ms_loaded ? range_tree_add : NULL, msp->ms_tree);
1581         range_tree_swap(freed_tree, defer_tree);
1352         space_map_vacate(*defer_map, sm->sm_loaded ? space_map_free : NULL, sm);
1353         ASSERT0((*defer_map)->sm_space);
1354         ASSERT0(avl_numnodes(&(*defer_map)->sm_root));
1355         space_map_swap(freed_map, defer_map);

1583         space_map_update(msp->ms_sm);
1357         *smo = *smosync;

1585         msp->ms_deferspace += defer_delta;
1586         ASSERT3S(msp->ms_deferspace, >=, 0);
1587         ASSERT3S(msp->ms_deferspace, <=, msp->ms_size);
1361         ASSERT3S(msp->ms_deferspace, <=, sm->sm_size);
1588         if (msp->ms_deferspace != 0) {
1589                 /*
1590                  * Keep syncing this metaslab until all deferred frees
1591                  * are back in circulation.
1592                  */
1593                 vdev_dirty(vd, VDD_METASLAB, msp, txg + 1);
1594         }

1596         if (msp->ms_loaded && msp->ms_access_txg < txg) {
1597                 for (int t = 1; t < TXG_CONCURRENT_STATES; t++) {
1598                         VERIFY0(range_tree_space(
1599                             msp->ms_alloctree[(txg + t) & TXG_MASK]));
1600                 }
1370         /*
1371          * If the map is loaded but no longer active, evict it as soon as all
1372          * future allocations have synced.  (If we unloaded it now and then
1373          * loaded a moment later, the map wouldn't reflect those allocations.)
1374          */
1375         if (sm->sm_loaded && (msp->ms_weight & METASLAB_ACTIVE_MASK) == 0) {
1376                 int evictable = 1;

1602                 if (!metaslab_debug_unload)
1603                         metaslab_unload(msp);
1378                 for (int t = 1; t < TXG_CONCURRENT_STATES; t++)
1379                         if (msp->ms_allocmap[(txg + t) & TXG_MASK]->sm_space)
1380                                 evictable = 0;

1382                 if (evictable && !metaslab_debug)
1383                         space_map_unload(sm);
1604         }
```

```
1606         metaslab_group_sort(mg, msp, metaslab_weight(msp));
1607         mutex_exit(&msp->ms_lock);

1388         mutex_exit(&msp->ms_lock);
1609 }

1611 void
1612 metaslab_sync_reassess(metaslab_group_t *mg)
1613 {
1394         vdev_t *vd = mg->mg_vd;
1614         int64_t failures = mg->mg_alloc_failures;

1616         metaslab_group_alloc_update(mg);

1399         /*
1400          * Re-evaluate all metaslabs which have lower offsets than the
1401          * bonus area.
1402          */
1403         for (int m = 0; m < vd->vdev_ms_count; m++) {
1404                 metaslab_t *msp = vd->vdev_ms[m];

1406                 if (msp->ms_map->sm_start > mg->mg_bonus_area)
1407                         break;

1409                 mutex_enter(&msp->ms_lock);
1410                 metaslab_group_sort(mg, msp, metaslab_weight(msp));
1411                 mutex_exit(&msp->ms_lock);
1412         }

1617         atomic_add_64(&mg->mg_alloc_failures, -failures);

1619         /*
1620          * Preload the next potential metaslabs
1417          * Prefetch the next potential metaslabs
1621          */
1622         metaslab_group_preload(mg);
1419         metaslab_prefetch(mg);
1623 }

1625 static uint64_t
1626 metaslab_distance(metaslab_t *msp, dva_t *dva)
1627 {
1628         uint64_t ms_shift = msp->ms_group->mg_vd->vdev_ms_shift;
1629         uint64_t offset = DVA_GET_OFFSET(dva) >> ms_shift;
1630         uint64_t start = msp->ms_id;
1427         uint64_t start = msp->ms_map->sm_start >> ms_shift;

1632         if (msp->ms_group->mg_vd->vdev_id != DVA_GET_VDEV(dva))
1633                 return (1ULL << 63);

1635         if (offset < start)
1636                 return ((start - offset) << ms_shift);
1637         if (offset > start)
1638                 return ((offset - start) << ms_shift);
1639         return (0);
1640 }

1642 static uint64_t
1643 metaslab_group_alloc(metaslab_group_t *mg, uint64_t psize, uint64_t asize,
1644     uint64_t txg, uint64_t min_distance, dva_t *dva, int d, int flags)
1645 {
1646         spa_t *spa = mg->mg_vd->vdev_spa;
1647         metaslab_t *msp = NULL;
1648         uint64_t offset = -1ULL;
1649         avl_tree_t *t = &mg->mg_metaslab_tree;
1650         uint64_t activation_weight;
```



```
1651         uint64_t target_distance;
1652         int i;

1654         activation_weight = METASLAB_WEIGHT_PRIMARY;
1655         for (i = 0; i < d; i++) {
1656                 if (DVA_GET_VDEV(&dva[i]) == mg->mg_vd->vdev_id) {
1657                         activation_weight = METASLAB_WEIGHT_SECONDARY;
1658                         break;
1659                 }
1660         }

1662         for (;;) {
1663                 boolean_t was_active;

1665                 mutex_enter(&mg->mg_lock);
1666                 for (msp = avl_first(t); msp; msp = AVL_NEXT(t, msp)) {
1667                         if (msp->ms_weight < asize) {
1668                                 spa_dbgmsg(spa, "%s: failed to meet weight "
1669                                     "requirement: vdev %llu, txg %llu, mg %p, "
1670                                     "msp %p, psize %llu, asize %llu, "
1671                                     "failures %llu, weight %llu",
1672                                     spa_name(spa), mg->mg_vd->vdev_id, txg,
1673                                     mg, msp, psize, asize,
1674                                     mg->mg_alloc_failures, msp->ms_weight);
1675                                 mutex_exit(&mg->mg_lock);
1676                                 return (-1ULL);
1677                         }

1679                         /*
1680                          * If the selected metaslab is condensing, skip it.
1681                          */
1682                         if (msp->ms_condensing)
1479                         if (msp->ms_map->sm_condensing)
1683                                 continue;

1685                         was_active = msp->ms_weight & METASLAB_ACTIVE_MASK;
1686                         if (activation_weight == METASLAB_WEIGHT_PRIMARY)
1687                                 break;

1689                         target_distance = min_distance +
1690                             (space_map_allocated(msp->ms_sm) != 0 ? 0 :
1691                             min_distance >> 1);
1487                             (msp->ms_smo.smo_alloc ? 0 : min_distance >> 1);

1693                         for (i = 0; i < d; i++)
1694                                 if (metaslab_distance(msp, &dva[i]) <
1695                                     target_distance)
1696                                         break;
1697                         if (i == d)
1698                                 break;
1699                 }
1700                 mutex_exit(&mg->mg_lock);
1701                 if (msp == NULL)
1702                         return (-1ULL);

1704                 mutex_enter(&msp->ms_lock);

1706                 /*
1707                  * If we've already reached the allowable number of failed
1708                  * allocation attempts on this metaslab group then we
1709                  * consider skipping it. We skip it only if we're allowed
1710                  * to "fast" gang, the physical size is larger than
1711                  * a gang block, and we're attempting to allocate from
1712                  * the primary metaslab.
1713                  */
1714                 if (mg->mg_alloc_failures > zfs_mg_alloc_failures &&
```

```
1715                    CAN_FASTGANG(flags) && psize > SPA_GANGBLOCKSIZE &&
1716                    activation_weight == METASLAB_WEIGHT_PRIMARY) {
1717                        spa_dbgmsg(spa, "%s: skipping metaslab group: "
1718                            "vdev %llu, txg %llu, mg %p, msp[%llu] %p, "
1719                            "psize %llu, asize %llu, failures %llu",
1720                            spa_name(spa), mg->mg_vd->vdev_id, txg, mg,
1721                            msp->ms_id, msp, psize, asize,
1514                            "vdev %llu, txg %llu, mg %p, psize %llu, "
1515                            "asize %llu, failures %llu", spa_name(spa),
1516                            mg->mg_vd->vdev_id, txg, mg, psize, asize,
1722                            mg->mg_alloc_failures);
1723                        mutex_exit(&msp->ms_lock);
1724                        return (-1ULL);
1725                }

1727                /*
1728                 * Ensure that the metaslab we have selected is still
1729                 * capable of handling our request. It's possible that
1730                 * another thread may have changed the weight while we
1731                 * were blocked on the metaslab lock.
1732                 */
1733                if (msp->ms_weight < asize || (was_active &&
1734                    !(msp->ms_weight & METASLAB_ACTIVE_MASK) &&
1735                    activation_weight == METASLAB_WEIGHT_PRIMARY)) {
1736                        mutex_exit(&msp->ms_lock);
1737                        continue;
1738                }

1740                if ((msp->ms_weight & METASLAB_WEIGHT_SECONDARY) &&
1741                    activation_weight == METASLAB_WEIGHT_PRIMARY) {
1742                        metaslab_passivate(msp,
1743                            msp->ms_weight & ~METASLAB_ACTIVE_MASK);
1744                        mutex_exit(&msp->ms_lock);
1745                        continue;
1746                }

1748                if (metaslab_activate(msp, activation_weight) != 0) {
1749                        mutex_exit(&msp->ms_lock);
1750                        continue;
1751                }

1753                /*
1754                 * If this metaslab is currently condensing then pick again as
1755                 * we can't manipulate this metaslab until it's committed
1756                 * to disk.
1757                 */
1758                if (msp->ms_condensing) {
1553                if (msp->ms_map->sm_condensing) {
1759                        mutex_exit(&msp->ms_lock);
1760                        continue;
1761                }

1763                if ((offset = metaslab_block_alloc(msp, asize)) != -1ULL)
1558                if ((offset = space_map_alloc(msp->ms_map, asize)) != -1ULL)
1764                        break;

1766                atomic_inc_64(&mg->mg_alloc_failures);

1768                metaslab_passivate(msp, metaslab_block_maxsize(msp));
1563                metaslab_passivate(msp, space_map_maxsize(msp->ms_map));

1769                mutex_exit(&msp->ms_lock);
1770        }

1772        if (range_tree_space(msp->ms_alloctree[txg & TXG_MASK]) == 0)
1568        if (msp->ms_allocmap[txg & TXG_MASK]->sm_space == 0)
```



```
1773                vdev_dirty(mg->mg_vd, VDD_METASLAB, msp, txg);

1775        range_tree_add(msp->ms_alloctree[txg & TXG_MASK], offset, asize);
1776        msp->ms_access_txg = txg + metaslab_unload_delay;
1571        space_map_add(msp->ms_allocmap[txg & TXG_MASK], offset, asize);

1778        mutex_exit(&msp->ms_lock);

1780        return (offset);
1781 }
______unchanged_portion_omitted_

1989 /*
1990  * Free the block represented by DVA in the context of the specified
1991  * transaction group.
1992  */
1993 static void
1994 metaslab_free_dva(spa_t *spa, const dva_t *dva, uint64_t txg, boolean_t now)
1995 {
1996        uint64_t vdev = DVA_GET_VDEV(dva);
1997        uint64_t offset = DVA_GET_OFFSET(dva);
1998        uint64_t size = DVA_GET_ASIZE(dva);
1999        vdev_t *vd;
2000        metaslab_t *msp;

2002        ASSERT(DVA_IS_VALID(dva));

2004        if (txg > spa_freeze_txg(spa))
2005                return;

2007        if ((vd = vdev_lookup_top(spa, vdev)) == NULL ||
2008            (offset >> vd->vdev_ms_shift) >= vd->vdev_ms_count) {
2009                cmn_err(CE_WARN, "metaslab_free_dva(): bad DVA %llu:%llu",
2010                    (u_longlong_t)vdev, (u_longlong_t)offset);
2011                ASSERT(0);
2012                return;
2013        }

2015        msp = vd->vdev_ms[offset >> vd->vdev_ms_shift];

2017        if (DVA_GET_GANG(dva))
2018                size = vdev_psize_to_asize(vd, SPA_GANGBLOCKSIZE);

2020        mutex_enter(&msp->ms_lock);

2022        if (now) {
2023                range_tree_remove(msp->ms_alloctree[txg & TXG_MASK],
1818                space_map_remove(msp->ms_allocmap[txg & TXG_MASK],
2024                    offset, size);

2026                VERIFY(!msp->ms_condensing);
2027                VERIFY3U(offset, >=, msp->ms_start);
2028                VERIFY3U(offset + size, <=, msp->ms_start + msp->ms_size);
2029                VERIFY3U(range_tree_space(msp->ms_tree) + size, <=,
2030                    msp->ms_size);
2031                VERIFY0(P2PHASE(offset, 1ULL << vd->vdev_ashift));
2032                VERIFY0(P2PHASE(size, 1ULL << vd->vdev_ashift));
2033                range_tree_add(msp->ms_tree, offset, size);
1820                space_map_free(msp->ms_map, offset, size);
2034        } else {
2035                if (range_tree_space(msp->ms_freetree[txg & TXG_MASK]) == 0)
1822                if (msp->ms_freemap[txg & TXG_MASK]->sm_space == 0)
2036                        vdev_dirty(vd, VDD_METASLAB, msp, txg);
2037                range_tree_add(msp->ms_freetree[txg & TXG_MASK],
2038                    offset, size);
1824                space_map_add(msp->ms_freemap[txg & TXG_MASK], offset, size);
```

```
2039         }

2041         mutex_exit(&msp->ms_lock);
2042 }

2044 /*
2045  * Intent log support: upon opening the pool after a crash, notify the SPA
2046  * of blocks that the intent log has allocated for immediate write, but
2047  * which are still considered free by the SPA because the last transaction
2048  * group didn't commit yet.
2049  */
2050 static int
2051 metaslab_claim_dva(spa_t *spa, const dva_t *dva, uint64_t txg)
2052 {
2053         uint64_t vdev = DVA_GET_VDEV(dva);
2054         uint64_t offset = DVA_GET_OFFSET(dva);
2055         uint64_t size = DVA_GET_ASIZE(dva);
2056         vdev_t *vd;
2057         metaslab_t *msp;
2058         int error = 0;

2060         ASSERT(DVA_IS_VALID(dva));

2062         if ((vd = vdev_lookup_top(spa, vdev)) == NULL ||
2063             (offset >> vd->vdev_ms_shift) >= vd->vdev_ms_count)
2064                 return (SET_ERROR(ENXIO));

2066         msp = vd->vdev_ms[offset >> vd->vdev_ms_shift];

2068         if (DVA_GET_GANG(dva))
2069                 size = vdev_psize_to_asize(vd, SPA_GANGBLOCKSIZE);

2071         mutex_enter(&msp->ms_lock);

2073         if ((txg != 0 && spa_writeable(spa)) || !msp->ms_loaded)
1859         if ((txg != 0 && spa_writeable(spa)) || !msp->ms_map->sm_loaded)
2074                 error = metaslab_activate(msp, METASLAB_WEIGHT_SECONDARY);

2076         if (error == 0 && !range_tree_contains(msp->ms_tree, offset, size))
1862         if (error == 0 && !space_map_contains(msp->ms_map, offset, size))
2077                 error = SET_ERROR(ENOENT);

2079         if (error || txg == 0) {        /* txg == 0 indicates dry run */
2080                 mutex_exit(&msp->ms_lock);
2081                 return (error);
2082         }

2084         VERIFY(!msp->ms_condensing);
2085         VERIFY0(P2PHASE(offset, 1ULL << vd->vdev_ashift));
2086         VERIFY0(P2PHASE(size, 1ULL << vd->vdev_ashift));
2087         VERIFY3U(range_tree_space(msp->ms_tree) - size, <=, msp->ms_size);
2088         range_tree_remove(msp->ms_tree, offset, size);
1870         space_map_claim(msp->ms_map, offset, size);

2090         if (spa_writeable(spa)) {       /* don't dirty if we're zdb(1M) */
2091                 if (range_tree_space(msp->ms_alloctree[txg & TXG_MASK]) == 0)
1873                 if (msp->ms_allocmap[txg & TXG_MASK]->sm_space == 0)
2092                         vdev_dirty(vd, VDD_METASLAB, msp, txg);
2093                 range_tree_add(msp->ms_alloctree[txg & TXG_MASK], offset, size);
1875                 space_map_add(msp->ms_allocmap[txg & TXG_MASK], offset, size);
2094         }

2096         mutex_exit(&msp->ms_lock);

2098         return (0);
2099 }
```



```
2101 int
2102 metaslab_alloc(spa_t *spa, metaslab_class_t *mc, uint64_t psize, blkptr_t *bp,
2103     int ndvas, uint64_t txg, blkptr_t *hintbp, int flags)
2104 {
2105         dva_t *dva = bp->blk_dva;
2106         dva_t *hintdva = hintbp->blk_dva;
2107         int error = 0;

2109         ASSERT(bp->blk_birth == 0);
2110         ASSERT(BP_PHYSICAL_BIRTH(bp) == 0);

2112         spa_config_enter(spa, SCL_ALLOC, FTAG, RW_READER);

2114         if (mc->mc_rotor == NULL) {     /* no vdevs in this class */
2115                 spa_config_exit(spa, SCL_ALLOC, FTAG);
2116                 return (SET_ERROR(ENOSPC));
2117         }

2119         ASSERT(ndvas > 0 && ndvas <= spa_max_replication(spa));
2120         ASSERT(BP_GET_NDVAS(bp) == 0);
2121         ASSERT(hintbp == NULL || ndvas <= BP_GET_NDVAS(hintbp));

2123         for (int d = 0; d < ndvas; d++) {
2124                 error = metaslab_alloc_dva(spa, mc, psize, dva, d, hintdva,
2125                     txg, flags);
2126                 if (error != 0) {
1908                 if (error) {
2127                         for (d--; d >= 0; d--) {
2128                                 metaslab_free_dva(spa, &dva[d], txg, B_TRUE);
2129                                 bzero(&dva[d], sizeof (dva_t));
2130                         }
2131                         spa_config_exit(spa, SCL_ALLOC, FTAG);
2132                         return (error);
2133                 }
2134         }
2135         ASSERT(error == 0);
2136         ASSERT(BP_GET_NDVAS(bp) == ndvas);

2138         spa_config_exit(spa, SCL_ALLOC, FTAG);

2140         BP_SET_BIRTH(bp, txg, txg);

2142         return (0);
2143 }
```

_____***unchanged_portion_omitted_***

```
1975 static void
1976 checkmap(space_map_t *sm, uint64_t off, uint64_t size)
1977 {
1978         space_seg_t *ss;
1979         avl_index_t where;

1981         mutex_enter(sm->sm_lock);
1982         ss = space_map_find(sm, off, size, &where);
1983         if (ss != NULL)
1984                 panic("freeing free block; ss=%p", (void *)ss);
1985         mutex_exit(sm->sm_lock);
1986 }

2193 void
2194 metaslab_check_free(spa_t *spa, const blkptr_t *bp)
2195 {
2196         if ((zfs_flags & ZFS_DEBUG_ZIO_FREE) == 0)
2197                 return;
```

```
2199         spa_config_enter(spa, SCL_VDEV, FTAG, RW_READER);
2200         for (int i = 0; i < BP_GET_NDVAS(bp); i++) {
2201                 uint64_t vdev = DVA_GET_VDEV(&bp->blk_dva[i]);
2202                 vdev_t *vd = vdev_lookup_top(spa, vdev);
2203                 uint64_t offset = DVA_GET_OFFSET(&bp->blk_dva[i]);
1996                 uint64_t vdid = DVA_GET_VDEV(&bp->blk_dva[i]);
1997                 vdev_t *vd = vdev_lookup_top(spa, vdid);
1998                 uint64_t off = DVA_GET_OFFSET(&bp->blk_dva[i]);
2204                 uint64_t size = DVA_GET_ASIZE(&bp->blk_dva[i]);
2205                 metaslab_t *msp = vd->vdev_ms[offset >> vd->vdev_ms_shift];
2000                 metaslab_t *ms = vd->vdev_ms[off >> vd->vdev_ms_shift];

2207                 if (msp->ms_loaded)
2208                         range_tree_verify(msp->ms_tree, offset, size);
2002                 if (ms->ms_map->sm_loaded)
2003                         checkmap(ms->ms_map, off, size);

2210                 for (int j = 0; j < TXG_SIZE; j++)
2211                         range_tree_verify(msp->ms_freetree[j], offset, size);
2006                         checkmap(ms->ms_freemap[j], off, size);
2212                 for (int j = 0; j < TXG_DEFER_SIZE; j++)
2213                         range_tree_verify(msp->ms_defertree[j], offset, size);
2008                         checkmap(ms->ms_defermap[j], off, size);
2214         }
2215         spa_config_exit(spa, SCL_VDEV, FTAG);
2216 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    9202 Tue Sep  3 20:27:00 2013
new/usr/src/uts/common/fs/zfs/range_tree.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <sys/zfs_context.h>
#include <sys/spa.h>
#include <sys/dmu.h>
#include <sys/dnode.h>
#include <sys/zio.h>
#include <sys/range_tree.h>

static kmem_cache_t *range_seg_cache;

void
range_tree_init(void)
{
        ASSERT(range_seg_cache == NULL);
        range_seg_cache = kmem_cache_create("range_seg_cache",
            sizeof (range_seg_t), 0, NULL, NULL, NULL, NULL, NULL, 0);
}

void
range_tree_fini(void)
{
        kmem_cache_destroy(range_seg_cache);
        range_seg_cache = NULL;
}

void
```



```
range_tree_stat_verify(range_tree_t *rt)
{
        range_seg_t *rs;
        uint64_t hist[RANGE_TREE_HISTOGRAM_SIZE] = { 0 };
        int i;

        for (rs = avl_first(&rt->rt_root); rs != NULL;
            rs = AVL_NEXT(&rt->rt_root, rs)) {
                uint64_t size = rs->rs_end - rs->rs_start;
                int idx = highbit(size) - 1;

                hist[idx]++;
                ASSERT3U(hist[idx], !=, 0);
        }

        for (i = 0; i < RANGE_TREE_HISTOGRAM_SIZE; i++) {
                if (hist[i] != rt->rt_histogram[i]) {
                        zfs_dbgmsg("i=%d, hist=%p, hist=%llu, rt_hist=%llu",
                            i, hist, hist[i], rt->rt_histogram[i]);
                }
                VERIFY3U(hist[i], ==, rt->rt_histogram[i]);
        }
}

static void
range_tree_stat_incr(range_tree_t *rt, range_seg_t *rs)
{
        uint64_t size = rs->rs_end - rs->rs_start;
        int idx = highbit(size) - 1;

        ASSERT3U(idx, <,
            sizeof (rt->rt_histogram) / sizeof (*rt->rt_histogram));

        ASSERT(MUTEX_HELD(rt->rt_lock));
        rt->rt_histogram[idx]++;
        ASSERT3U(rt->rt_histogram[idx], !=, 0);
}

static void
range_tree_stat_decr(range_tree_t *rt, range_seg_t *rs)
{
        uint64_t size = rs->rs_end - rs->rs_start;
        int idx = highbit(size) - 1;

        ASSERT3U(idx, <,
            sizeof (rt->rt_histogram) / sizeof (*rt->rt_histogram));

        ASSERT(MUTEX_HELD(rt->rt_lock));
        ASSERT3U(rt->rt_histogram[idx], !=, 0);
        rt->rt_histogram[idx]--;
}

/*
 * NOTE: caller is responsible for all locking.
 */
static int
range_tree_seg_compare(const void *x1, const void *x2)
{
        const range_seg_t *r1 = x1;
        const range_seg_t *r2 = x2;

        if (r1->rs_start < r2->rs_start) {
                if (r1->rs_end > r2->rs_start)
                        return (0);
                return (-1);
        }
```

```
120         if (r1->rs_start > r2->rs_start) {
121                 if (r1->rs_start < r2->rs_end)
122                         return (0);
123                 return (1);
124         }
125         return (0);
126 }

128 range_tree_t *
129 range_tree_create(range_tree_ops_t *ops, void *arg, kmutex_t *lp)
130 {
131         range_tree_t *rt;

133         rt = kmem_zalloc(sizeof (range_tree_t), KM_SLEEP);

135         avl_create(&rt->rt_root, range_tree_seg_compare,
136             sizeof (range_seg_t), offsetof(range_seg_t, rs_node));

138         rt->rt_lock = lp;
139         rt->rt_ops = ops;
140         rt->rt_arg = arg;

142         if (rt->rt_ops != NULL)
143                 rt->rt_ops->rtop_create(rt, rt->rt_arg);

145         return (rt);
146 }

148 void
149 range_tree_destroy(range_tree_t *rt)
150 {
151         VERIFY0(rt->rt_space);

153         if (rt->rt_ops != NULL)
154                 rt->rt_ops->rtop_destroy(rt, rt->rt_arg);

156         avl_destroy(&rt->rt_root);
157         kmem_free(rt, sizeof (*rt));
158 }

160 void
161 range_tree_add(void *arg, uint64_t start, uint64_t size)
162 {
163         range_tree_t *rt = arg;
164         avl_index_t where;
165         range_seg_t rsearch, *rs_before, *rs_after, *rs;
166         uint64_t end = start + size;
167         boolean_t merge_before, merge_after;

169         ASSERT(MUTEX_HELD(rt->rt_lock));
170         VERIFY(size != 0);

172         rsearch.rs_start = start;
173         rsearch.rs_end = end;
174         rs = avl_find(&rt->rt_root, &rsearch, &where);

176         if (rs != NULL && rs->rs_start <= start && rs->rs_end >= end) {
177                 zfs_panic_recover("zfs: allocating allocated segment"
178                     "(offset=%llu size=%llu)\n",
179                     (longlong_t)start, (longlong_t)size);
180                 return;
181         }

183         /* Make sure we don't overlap with either of our neighbors */
184         VERIFY(rs == NULL);
```



```
186         rs_before = avl_nearest(&rt->rt_root, where, AVL_BEFORE);
187         rs_after = avl_nearest(&rt->rt_root, where, AVL_AFTER);

189         merge_before = (rs_before != NULL && rs_before->rs_end == start);
190         merge_after = (rs_after != NULL && rs_after->rs_start == end);

192         if (merge_before && merge_after) {
193                 avl_remove(&rt->rt_root, rs_before);
194                 if (rt->rt_ops != NULL) {
195                         rt->rt_ops->rtop_remove(rt, rs_before, rt->rt_arg);
196                         rt->rt_ops->rtop_remove(rt, rs_after, rt->rt_arg);
197                 }

199                 range_tree_stat_decr(rt, rs_before);
200                 range_tree_stat_decr(rt, rs_after);

202                 rs_after->rs_start = rs_before->rs_start;
203                 kmem_cache_free(range_seg_cache, rs_before);
204                 rs = rs_after;
205         } else if (merge_before) {
206                 if (rt->rt_ops != NULL)
207                         rt->rt_ops->rtop_remove(rt, rs_before, rt->rt_arg);

209                 range_tree_stat_decr(rt, rs_before);

211                 rs_before->rs_end = end;
212                 rs = rs_before;
213         } else if (merge_after) {
214                 if (rt->rt_ops != NULL)
215                         rt->rt_ops->rtop_remove(rt, rs_after, rt->rt_arg);

217                 range_tree_stat_decr(rt, rs_after);

219                 rs_after->rs_start = start;
220                 rs = rs_after;
221         } else {
222                 rs = kmem_cache_alloc(range_seg_cache, KM_SLEEP);
223                 rs->rs_start = start;
224                 rs->rs_end = end;
225                 avl_insert(&rt->rt_root, rs, where);
226         }

228         if (rt->rt_ops != NULL)
229                 rt->rt_ops->rtop_add(rt, rs, rt->rt_arg);

231         range_tree_stat_incr(rt, rs);
232         rt->rt_space += size;
233 }

235 void
236 range_tree_remove(void *arg, uint64_t start, uint64_t size)
237 {
238         range_tree_t *rt = arg;
239         avl_index_t where;
240         range_seg_t rsearch, *rs, *newseg;
241         uint64_t end = start + size;
242         boolean_t left_over, right_over;

244         ASSERT(MUTEX_HELD(rt->rt_lock));
245         VERIFY3U(size, !=, 0);
246         VERIFY3U(size, <=, rt->rt_space);

248         rsearch.rs_start = start;
249         rsearch.rs_end = end;
250         rs = avl_find(&rt->rt_root, &rsearch, &where);
```

```
252         /* Make sure we completely overlap with someone */
253         if (rs == NULL) {
254                 zfs_panic_recover("zfs: freeing free segment "
255                     "(offset=%llu size=%llu)",
256                     (longlong_t)start, (longlong_t)size);
257                 return;
258         }
259         VERIFY3U(rs->rs_start, <=, start);
260         VERIFY3U(rs->rs_end, >=, end);
261
262         left_over = (rs->rs_start != start);
263         right_over = (rs->rs_end != end);
264
265         range_tree_stat_decr(rt, rs);
266
267         if (rt->rt_ops != NULL)
268                 rt->rt_ops->rtop_remove(rt, rs, rt->rt_arg);
269
270         if (left_over && right_over) {
271                 newseg = kmem_cache_alloc(range_seg_cache, KM_SLEEP);
272                 newseg->rs_start = end;
273                 newseg->rs_end = rs->rs_end;
274                 range_tree_stat_incr(rt, newseg);
275
276                 rs->rs_end = start;
277
278                 avl_insert_here(&rt->rt_root, newseg, rs, AVL_AFTER);
279                 if (rt->rt_ops != NULL)
280                         rt->rt_ops->rtop_add(rt, newseg, rt->rt_arg);
281         } else if (left_over) {
282                 rs->rs_end = start;
283         } else if (right_over) {
284                 rs->rs_start = end;
285         } else {
286                 avl_remove(&rt->rt_root, rs);
287                 kmem_cache_free(range_seg_cache, rs);
288                 rs = NULL;
289         }
290
291         if (rs != NULL) {
292                 range_tree_stat_incr(rt, rs);
293
294                 if (rt->rt_ops != NULL)
295                         rt->rt_ops->rtop_add(rt, rs, rt->rt_arg);
296         }
297
298         rt->rt_space -= size;
299 }
300
301 static range_seg_t *
302 range_tree_find(range_tree_t *rt, uint64_t start, uint64_t size,
303     avl_index_t *wherep)
304 {
305         range_seg_t rsearch, *rs;
306         uint64_t end = start + size;
307
308         ASSERT(MUTEX_HELD(rt->rt_lock));
309         VERIFY(size != 0);
310
311         rsearch.rs_start = start;
312         rsearch.rs_end = end;
313         rs = avl_find(&rt->rt_root, &rsearch, wherep);
314
315         if (rs != NULL && rs->rs_start <= start && rs->rs_end >= end)
316                 return (rs);
317         return (NULL);
```



```
318 }
319
320 void
321 range_tree_verify(range_tree_t *rt, uint64_t off, uint64_t size)
322 {
323         range_seg_t *rs;
324         avl_index_t where;
325
326         mutex_enter(rt->rt_lock);
327         rs = range_tree_find(rt, off, size, &where);
328         if (rs != NULL)
329                 panic("freeing free block; rs=%p", (void *)rs);
330         mutex_exit(rt->rt_lock);
331 }
332
333 boolean_t
334 range_tree_contains(range_tree_t *rt, uint64_t start, uint64_t size)
335 {
336         avl_index_t where;
337
338         return (range_tree_find(rt, start, size, &where) != NULL);
339 }
340
341 void
342 range_tree_swap(range_tree_t **rtsrc, range_tree_t **rtdst)
343 {
344         range_tree_t *rt;
345
346         ASSERT(MUTEX_HELD((*rtsrc)->rt_lock));
347         ASSERT0(range_tree_space(*rtdst));
348         ASSERT0(avl_numnodes(&(*rtdst)->rt_root));
349
350         rt = *rtsrc;
351         *rtsrc = *rtdst;
352         *rtdst = rt;
353 }
354
355 void
356 range_tree_vacate(range_tree_t *rt, range_tree_func_t *func, void *arg)
357 {
358         range_seg_t *rs;
359         void *cookie = NULL;
360
361         ASSERT(MUTEX_HELD(rt->rt_lock));
362
363         if (rt->rt_ops != NULL)
364                 rt->rt_ops->rtop_vacate(rt, rt->rt_arg);
365
366         while ((rs = avl_destroy_nodes(&rt->rt_root, &cookie)) != NULL) {
367                 if (func != NULL)
368                         func(arg, rs->rs_start, rs->rs_end - rs->rs_start);
369                 kmem_cache_free(range_seg_cache, rs);
370         }
371
372         bzero(rt->rt_histogram, sizeof (rt->rt_histogram));
373         rt->rt_space = 0;
374 }
375
376 void
377 range_tree_walk(range_tree_t *rt, range_tree_func_t *func, void *arg)
378 {
379         range_seg_t *rs;
380
381         ASSERT(MUTEX_HELD(rt->rt_lock));
382
383         for (rs = avl_first(&rt->rt_root); rs; rs = AVL_NEXT(&rt->rt_root, rs))
```

```
384                 func(arg, rs->rs_start, rs->rs_end - rs->rs_start);
385 }

387 uint64_t
388 range_tree_space(range_tree_t *rt)
389 {
390         return (rt->rt_space);
391 }
```

```
**********************************************************
  176003 Tue Sep  3 20:27:01 2013
new/usr/src/uts/common/fs/zfs/spa.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
______unchanged_portion_omitted_

1197 /*
1198  * Opposite of spa_load().
1199  */
1200 static void
1201 spa_unload(spa_t *spa)
1202 {
1203         int i;

1205         ASSERT(MUTEX_HELD(&spa_namespace_lock));

1207         /*
1208          * Stop async tasks.
1209          */
1210         spa_async_suspend(spa);

1212         /*
1213          * Stop syncing.
1214          */
1215         if (spa->spa_sync_on) {
1216                 txg_sync_stop(spa->spa_dsl_pool);
1217                 spa->spa_sync_on = B_FALSE;
1218         }

1220         /*
1221          * Wait for any outstanding async I/O to complete.
1222          */
1223         if (spa->spa_async_zio_root != NULL) {
1224                 (void) zio_wait(spa->spa_async_zio_root);
1225                 spa->spa_async_zio_root = NULL;
1226         }

1228         bpobj_close(&spa->spa_deferred_bpobj);

1230         spa_config_enter(spa, SCL_ALL, FTAG, RW_WRITER);

1232         /*
1233          * Close all vdevs.
1234          */
1235         if (spa->spa_root_vdev)
1236                 vdev_free(spa->spa_root_vdev);
1237         ASSERT(spa->spa_root_vdev == NULL);

1239         /*
1240          * Close the dsl pool.
1241          */
1242         if (spa->spa_dsl_pool) {
1243                 dsl_pool_close(spa->spa_dsl_pool);
1244                 spa->spa_dsl_pool = NULL;
1245                 spa->spa_meta_objset = NULL;
1246         }
```



```
1248         ddt_unload(spa);

1241         spa_config_enter(spa, SCL_ALL, FTAG, RW_WRITER);

1251         /*
1252          * Drop and purge level 2 cache
1253          */
1254         spa_l2cache_drop(spa);

1248         /*
1249          * Close all vdevs.
1250          */
1251         if (spa->spa_root_vdev)
1252                 vdev_free(spa->spa_root_vdev);
1253         ASSERT(spa->spa_root_vdev == NULL);

1256         for (i = 0; i < spa->spa_spares.sav_count; i++)
1257                 vdev_free(spa->spa_spares.sav_vdevs[i]);
1258         if (spa->spa_spares.sav_vdevs) {
1259                 kmem_free(spa->spa_spares.sav_vdevs,
1260                     spa->spa_spares.sav_count * sizeof (void *));
1261                 spa->spa_spares.sav_vdevs = NULL;
1262         }
1263         if (spa->spa_spares.sav_config) {
1264                 nvlist_free(spa->spa_spares.sav_config);
1265                 spa->spa_spares.sav_config = NULL;
1266         }
1267         spa->spa_spares.sav_count = 0;

1269         for (i = 0; i < spa->spa_l2cache.sav_count; i++) {
1270                 vdev_clear_stats(spa->spa_l2cache.sav_vdevs[i]);
1271                 vdev_free(spa->spa_l2cache.sav_vdevs[i]);
1272         }
1273         if (spa->spa_l2cache.sav_vdevs) {
1274                 kmem_free(spa->spa_l2cache.sav_vdevs,
1275                     spa->spa_l2cache.sav_count * sizeof (void *));
1276                 spa->spa_l2cache.sav_vdevs = NULL;
1277         }
1278         if (spa->spa_l2cache.sav_config) {
1279                 nvlist_free(spa->spa_l2cache.sav_config);
1280                 spa->spa_l2cache.sav_config = NULL;
1281         }
1282         spa->spa_l2cache.sav_count = 0;

1284         spa->spa_async_suspended = 0;

1286         if (spa->spa_comment != NULL) {
1287                 spa_strfree(spa->spa_comment);
1288                 spa->spa_comment = NULL;
1289         }

1291         spa_config_exit(spa, SCL_ALL, FTAG);
1292 }
______unchanged_portion_omitted_

4322 /*
4323  * Attach a device to a mirror.  The arguments are the path to any device
4324  * in the mirror, and the nvroot for the new device.  If the path specifies
4325  * a device that is not mirrored, we automatically insert the mirror vdev.
4326  *
4327  * If 'replacing' is specified, the new device is intended to replace the
4328  * existing device; in this case the two devices are made into their own
4329  * mirror using the 'replacing' vdev, which is functionally identical to
4330  * the mirror vdev (it actually reuses all the same ops) but has a few
4331  * extra rules: you can't attach to it after it's been created, and upon
4332  * completion of resilvering, the first disk (the one being replaced)
```

```
 * is automatically detached.
 */
int
spa_vdev_attach(spa_t *spa, uint64_t guid, nvlist_t *nvroot, int replacing)
{
        uint64_t txg, dtl_max_txg;
        vdev_t *rvd = spa->spa_root_vdev;
        vdev_t *oldvd, *newvd, *newrootvd, *pvd, *tvd;
        vdev_ops_t *pvops;
        char *oldvdpath, *newvdpath;
        int newvd_isspare;
        int error;

        ASSERT(spa_writeable(spa));

        txg = spa_vdev_enter(spa);

        oldvd = spa_lookup_by_guid(spa, guid, B_FALSE);

        if (oldvd == NULL)
                return (spa_vdev_exit(spa, NULL, txg, ENODEV));

        if (!oldvd->vdev_ops->vdev_op_leaf)
                return (spa_vdev_exit(spa, NULL, txg, ENOTSUP));

        pvd = oldvd->vdev_parent;

        if ((error = spa_config_parse(spa, &newrootvd, nvroot, NULL, 0,
            VDEV_ALLOC_ATTACH)) != 0)
                return (spa_vdev_exit(spa, NULL, txg, EINVAL));

        if (newrootvd->vdev_children != 1)
                return (spa_vdev_exit(spa, newrootvd, txg, EINVAL));

        newvd = newrootvd->vdev_child[0];

        if (!newvd->vdev_ops->vdev_op_leaf)
                return (spa_vdev_exit(spa, newrootvd, txg, EINVAL));

        if ((error = vdev_create(newrootvd, txg, replacing)) != 0)
                return (spa_vdev_exit(spa, newrootvd, txg, error));

        /*
         * Spares can't replace logs
         */
        if (oldvd->vdev_top->vdev_islog && newvd->vdev_isspare)
                return (spa_vdev_exit(spa, newrootvd, txg, ENOTSUP));

        if (!replacing) {
                /*
                 * For attach, the only allowable parent is a mirror or the root
                 * vdev.
                 */
                if (pvd->vdev_ops != &vdev_mirror_ops &&
                    pvd->vdev_ops != &vdev_root_ops)
                        return (spa_vdev_exit(spa, newrootvd, txg, ENOTSUP));

                pvops = &vdev_mirror_ops;
        } else {
                /*
                 * Active hot spares can only be replaced by inactive hot
                 * spares.
                 */
                if (pvd->vdev_ops == &vdev_spare_ops &&
                    oldvd->vdev_isspare &&
                    !spa_has_spare(spa, newvd->vdev_guid))
```



```
                        return (spa_vdev_exit(spa, newrootvd, txg, ENOTSUP));

                /*
                 * If the source is a hot spare, and the parent isn't already a
                 * spare, then we want to create a new hot spare.  Otherwise, we
                 * want to create a replacing vdev.  The user is not allowed to
                 * attach to a spared vdev child unless the 'isspare' state is
                 * the same (spare replaces spare, non-spare replaces
                 * non-spare).
                 */
                if (pvd->vdev_ops == &vdev_replacing_ops &&
                    spa_version(spa) < SPA_VERSION_MULTI_REPLACE) {
                        return (spa_vdev_exit(spa, newrootvd, txg, ENOTSUP));
                } else if (pvd->vdev_ops == &vdev_spare_ops &&
                    newvd->vdev_isspare != oldvd->vdev_isspare) {
                        return (spa_vdev_exit(spa, newrootvd, txg, ENOTSUP));
                }

                if (newvd->vdev_isspare)
                        pvops = &vdev_spare_ops;
                else
                        pvops = &vdev_replacing_ops;
        }

        /*
         * Make sure the new device is big enough.
         */
        if (newvd->vdev_asize < vdev_get_min_asize(oldvd))
                return (spa_vdev_exit(spa, newrootvd, txg, EOVERFLOW));

        /*
         * The new device cannot have a higher alignment requirement
         * than the top-level vdev.
         */
        if (newvd->vdev_ashift > oldvd->vdev_top->vdev_ashift)
                return (spa_vdev_exit(spa, newrootvd, txg, EDOM));

        /*
         * If this is an in-place replacement, update oldvd's path and devid
         * to make it distinguishable from newvd, and unopenable from now on.
         */
        if (strcmp(oldvd->vdev_path, newvd->vdev_path) == 0) {
                spa_strfree(oldvd->vdev_path);
                oldvd->vdev_path = kmem_alloc(strlen(newvd->vdev_path) + 5,
                    KM_SLEEP);
                (void) sprintf(oldvd->vdev_path, "%s/%s",
                    newvd->vdev_path, "old");
                if (oldvd->vdev_devid != NULL) {
                        spa_strfree(oldvd->vdev_devid);
                        oldvd->vdev_devid = NULL;
                }
        }

        /* mark the device being resilvered */
        newvd->vdev_resilver_txg = txg;

        /*
         * If the parent is not a mirror, or if we're replacing, insert the new
         * mirror/replacing/spare vdev above oldvd.
         */
        if (pvd->vdev_ops != pvops)
                pvd = vdev_add_parent(oldvd, pvops);

        ASSERT(pvd->vdev_top->vdev_parent == rvd);
        ASSERT(pvd->vdev_ops == pvops);
        ASSERT(oldvd->vdev_parent == pvd);
```

```
4466         /*
4467          * Extract the new device from its root and add it to pvd.
4468          */
4469         vdev_remove_child(newrootvd, newvd);
4470         newvd->vdev_id = pvd->vdev_children;
4471         newvd->vdev_crtxg = oldvd->vdev_crtxg;
4472         vdev_add_child(pvd, newvd);

4474         tvd = newvd->vdev_top;
4475         ASSERT(pvd->vdev_top == tvd);
4476         ASSERT(tvd->vdev_parent == rvd);

4478         vdev_config_dirty(tvd);

4480         /*
4481          * Set newvd's DTL to [TXG_INITIAL, dtl_max_txg) so that we account
4482          * for any dmu_sync-ed blocks.  It will propagate upward when
4483          * spa_vdev_exit() calls vdev_dtl_reassess().
4484          */
4485         dtl_max_txg = txg + TXG_CONCURRENT_STATES;

4487         vdev_dtl_dirty(newvd, DTL_MISSING, TXG_INITIAL,
4488             dtl_max_txg - TXG_INITIAL);

4490         if (newvd->vdev_isspare) {
4491                 spa_spare_activate(newvd);
4492                 spa_event_notify(spa, newvd, ESC_ZFS_VDEV_SPARE);
4493         }

4495         oldvdpath = spa_strdup(oldvd->vdev_path);
4496         newvdpath = spa_strdup(newvd->vdev_path);
4497         newvd_isspare = newvd->vdev_isspare;

4499         /*
4500          * Mark newvd's DTL dirty in this txg.
4501          */
4502         vdev_dirty(tvd, VDD_DTL, newvd, txg);

4504         /*
4505          * Schedule the resilver to restart in the future. We do this to
4506          * ensure that dmu_sync-ed blocks have been stitched into the
4507          * respective datasets.
4504          * Restart the resilver
4508          */
4509         dsl_resilver_restart(spa->spa_dsl_pool, dtl_max_txg);

4511         /*
4512          * Commit the config
4513          */
4514         (void) spa_vdev_exit(spa, newrootvd, dtl_max_txg, 0);

4516         spa_history_log_internal(spa, "vdev attach", NULL,
4517             "%s vdev=%s %s vdev=%s",
4518             replacing && newvd_isspare ? "spare in" :
4519             replacing ? "replace" : "attach", newvdpath,
4520             replacing ? "for" : "to", oldvdpath);

4522         spa_strfree(oldvdpath);
4523         spa_strfree(newvdpath);

4525         if (spa->spa_bootfs)
4526                 spa_event_notify(spa, newvd, ESC_ZFS_BOOTFS_VDEV_ATTACH);

4528         return (0);
4529 }
```

*_____unchanged_portion_omitted_*


```
5093 /*
5094  * Evacuate the device.
5095  */
5096 static int
5097 spa_vdev_remove_evacuate(spa_t *spa, vdev_t *vd)
5098 {
5099         uint64_t txg;
5100         int error = 0;

5102         ASSERT(MUTEX_HELD(&spa_namespace_lock));
5103         ASSERT(spa_config_held(spa, SCL_ALL, RW_WRITER) == 0);
5104         ASSERT(vd == vd->vdev_top);

5106         /*
5107          * Evacuate the device.  We don't hold the config lock as writer
5108          * since we need to do I/O but we do keep the
5109          * spa_namespace_lock held.  Once this completes the device
5110          * should no longer have any blocks allocated on it.
5111          */
5112         if (vd->vdev_islog) {
5113                 if (vd->vdev_stat.vs_alloc != 0)
5114                         error = spa_offline_log(spa);
5115         } else {
5116                 error = SET_ERROR(ENOTSUP);
5117         }

5119         if (error)
5120                 return (error);

5122         /*
5123          * The evacuation succeeded.  Remove any remaining MOS metadata
5124          * associated with this vdev, and wait for these changes to sync.
5125          */
5126         ASSERT0(vd->vdev_stat.vs_alloc);
5127         txg = spa_vdev_config_enter(spa);
5128         vd->vdev_removing = B_TRUE;
5129         vdev_dirty_leaves(vd, VDD_DTL, txg);
5126         vdev_dirty(vd, 0, NULL, txg);
5130         vdev_config_dirty(vd);
5131         spa_vdev_config_exit(spa, NULL, txg, 0, FTAG);

5133         return (0);
5134 }
```

*_____unchanged_portion_omitted_*

```
5900 /*
5901  * Set zpool properties.
5902  */
5903 static void
5904 spa_sync_props(void *arg, dmu_tx_t *tx)
5905 {
5906         nvlist_t *nvp = arg;
5907         spa_t *spa = dmu_tx_pool(tx)->dp_spa;
5908         objset_t *mos = spa->spa_meta_objset;
5909         nvpair_t *elem = NULL;

5911         mutex_enter(&spa->spa_props_lock);

5913         while ((elem = nvlist_next_nvpair(nvp, elem))) {
5914                 uint64_t intval;
5915                 char *strval, *fname;
5916                 zpool_prop_t prop;
5917                 const char *propname;
5918                 zprop_type_t proptype;
5919                 zfeature_info_t *feature;
```

```
5921                switch (prop = zpool_name_to_prop(nvpair_name(elem))) {
5922                case ZPROP_INVAL:
5923                        /*
5924                         * We checked this earlier in spa_prop_validate().
5925                         */
5926                        ASSERT(zpool_prop_feature(nvpair_name(elem)));

5928                        fname = strchr(nvpair_name(elem), '@') + 1;
5929                        VERIFY0(zfeature_lookup_name(fname, &feature));
5926                        VERIFY3U(0, ==, zfeature_lookup_name(fname, &feature));

5931                        spa_feature_enable(spa, feature, tx);
5932                        spa_history_log_internal(spa, "set", tx,
5933                            "%s=enabled", nvpair_name(elem));
5934                        break;

5936                case ZPOOL_PROP_VERSION:
5937                        intval = fnvpair_value_uint64(elem);
5934                        VERIFY(nvpair_value_uint64(elem, &intval) == 0);
5938                        /*
5939                         * The version is synced seperatly before other
5940                         * properties and should be correct by now.
5941                         */
5942                        ASSERT3U(spa_version(spa), >=, intval);
5943                        break;

5945                case ZPOOL_PROP_ALTROOT:
5946                        /*
5947                         * 'altroot' is a non-persistent property. It should
5948                         * have been set temporarily at creation or import time.
5949                         */
5950                        ASSERT(spa->spa_root != NULL);
5951                        break;

5953                case ZPOOL_PROP_READONLY:
5954                case ZPOOL_PROP_CACHEFILE:
5955                        /*
5956                         * 'readonly' and 'cachefile' are also non-persisitent
5957                         * properties.
5958                         */
5959                        break;
5960                case ZPOOL_PROP_COMMENT:
5961                        strval = fnvpair_value_string(elem);
5958                        VERIFY(nvpair_value_string(elem, &strval) == 0);
5962                        if (spa->spa_comment != NULL)
5963                                spa_strfree(spa->spa_comment);
5964                        spa->spa_comment = spa_strdup(strval);
5965                        /*
5966                         * We need to dirty the configuration on all the vdevs
5967                         * so that their labels get updated.  It's unnecessary
5968                         * to do this for pool creation since the vdev's
5969                         * configuratoin has already been dirtied.
5970                         */
5971                        if (tx->tx_txg != TXG_INITIAL)
5972                                vdev_config_dirty(spa->spa_root_vdev);
5973                        spa_history_log_internal(spa, "set", tx,
5974                            "%s=%s", nvpair_name(elem), strval);
5975                        break;
5976                default:
5977                        /*
5978                         * Set pool property values in the poolprops mos object.
5979                         */
5980                        if (spa->spa_pool_props_object == 0) {
5981                                spa->spa_pool_props_object =
5982                                    zap_create_link(mos, DMU_OT_POOL_PROPS,
```



```
5983                                    DMU_POOL_DIRECTORY_OBJECT, DMU_POOL_PROPS,
5984                                    tx);
5985                        }

5987                        /* normalize the property name */
5988                        propname = zpool_prop_to_name(prop);
5989                        proptype = zpool_prop_get_type(prop);

5991                        if (nvpair_type(elem) == DATA_TYPE_STRING) {
5992                                ASSERT(proptype == PROP_TYPE_STRING);
5993                                strval = fnvpair_value_string(elem);
5994                                VERIFY0(zap_update(mos,
5990                                VERIFY(nvpair_value_string(elem, &strval) == 0);
5991                                VERIFY(zap_update(mos,
5995                                    spa->spa_pool_props_object, propname,
5996                                    1, strlen(strval) + 1, strval, tx));
5993                                    1, strlen(strval) + 1, strval, tx) == 0);
5997                                spa_history_log_internal(spa, "set", tx,
5998                                    "%s=%s", nvpair_name(elem), strval);
5999                        } else if (nvpair_type(elem) == DATA_TYPE_UINT64) {
6000                                intval = fnvpair_value_uint64(elem);
5997                                VERIFY(nvpair_value_uint64(elem, &intval) == 0);

6002                                if (proptype == PROP_TYPE_INDEX) {
6003                                        const char *unused;
6004                                        VERIFY0(zpool_prop_index_to_string(
6005                                            prop, intval, &unused));
6001                                        VERIFY(zpool_prop_index_to_string(
6002                                            prop, intval, &unused) == 0);
6006                                }
6007                                VERIFY0(zap_update(mos,
6004                                VERIFY(zap_update(mos,
6008                                    spa->spa_pool_props_object, propname,
6009                                    8, 1, &intval, tx));
6006                                    8, 1, &intval, tx) == 0);
6010                                spa_history_log_internal(spa, "set", tx,
6011                                    "%s=%lld", nvpair_name(elem), intval);
6012                        } else {
6013                                ASSERT(0); /* not allowed */
6014                        }

6016                        switch (prop) {
6017                        case ZPOOL_PROP_DELEGATION:
6018                                spa->spa_delegation = intval;
6019                                break;
6020                        case ZPOOL_PROP_BOOTFS:
6021                                spa->spa_bootfs = intval;
6022                                break;
6023                        case ZPOOL_PROP_FAILUREMODE:
6024                                spa->spa_failmode = intval;
6025                                break;
6026                        case ZPOOL_PROP_AUTOEXPAND:
6027                                spa->spa_autoexpand = intval;
6028                                if (tx->tx_txg != TXG_INITIAL)
6029                                        spa_async_request(spa,
6030                                            SPA_ASYNC_AUTOEXPAND);
6031                                break;
6032                        case ZPOOL_PROP_DEDUPDITTO:
6033                                spa->spa_dedup_ditto = intval;
6034                                break;
6035                        default:
6036                                break;
6037                        }
6038                }

6040        }
```

```
6042         mutex_exit(&spa->spa_props_lock);
6043 }
```

unchanged_portion_omitted_

```
**********************************************************
   46220 Tue Sep  3 20:27:03 2013
new/usr/src/uts/common/fs/zfs/spa_misc.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 450 /*
 451  * Fires when spa_sync has not completed within zfs_deadman_synctime_ms.
 452  * If the zfs_deadman_enabled flag is set then it inspects all vdev queues
 453  * looking for potentially hung I/Os.
 454  */
 455 void
 456 spa_deadman(void *arg)
 457 {
 458         spa_t *spa = arg;

 460         /*
 461          * Disable the deadman timer if the pool is suspended.
 462          */
 463         if (spa_suspended(spa)) {
 464                 VERIFY(cyclic_reprogram(spa->spa_deadman_cycid, CY_INFINITY));
 465                 return;
 466         }

 468         zfs_dbgmsg("slow spa_sync: started %llu seconds ago, calls %llu",
 469             (gethrtime() - spa->spa_sync_starttime) / NANOSEC,
 470             ++spa->spa_deadman_calls);
 471         if (zfs_deadman_enabled)
 472                 vdev_deadman(spa->spa_root_vdev);
 473 }
______unchanged_portion_omitted_

 986 /*
 987  * Used in combination with spa_vdev_config_enter() to allow the syncing
 988  * of multiple transactions without releasing the spa_namespace_lock.
 989  */
 990 void
 991 spa_vdev_config_exit(spa_t *spa, vdev_t *vd, uint64_t txg, int error, char *tag)
 992 {
 993         ASSERT(MUTEX_HELD(&spa_namespace_lock));

 995         int config_changed = B_FALSE;

 997         ASSERT(txg > spa_last_synced_txg(spa));

 999         spa->spa_pending_vdev = NULL;

1001         /*
1002          * Reassess the DTLs.
1003          */
1004         vdev_dtl_reassess(spa->spa_root_vdev, 0, 0, B_FALSE);

1006         if (error == 0 && !list_is_empty(&spa->spa_config_dirty_list)) {
1007                 config_changed = B_TRUE;
1008                 spa->spa_config_generation++;
1009         }
```



```
1011         /*
1012          * Verify the metaslab classes.
1013          */
1014         ASSERT(metaslab_class_validate(spa_normal_class(spa)) == 0);
1015         ASSERT(metaslab_class_validate(spa_log_class(spa)) == 0);

1017         spa_config_exit(spa, SCL_ALL, spa);

1019         /*
1020          * Panic the system if the specified tag requires it.  This
1021          * is useful for ensuring that configurations are updated
1022          * transactionally.
1023          */
1024         if (zio_injection_enabled)
1025                 zio_handle_panic_injection(spa, tag, 0);

1027         /*
1028          * Note: this txg_wait_synced() is important because it ensures
1029          * that there won't be more than one config change per txg.
1030          * This allows us to use the txg as the generation number.
1031          */
1032         if (error == 0)
1033                 txg_wait_synced(spa->spa_dsl_pool, txg);

1035         if (vd != NULL) {
1036                 ASSERT(!vd->vdev_detached || vd->vdev_dtl_sm == NULL);
1028                 ASSERT(!vd->vdev_detached || vd->vdev_dtl_smo.smo_object == 0);
1037                 spa_config_enter(spa, SCL_ALL, spa, RW_WRITER);
1038                 vdev_free(vd);
1039                 spa_config_exit(spa, SCL_ALL, spa);
1040         }

1042         /*
1043          * If the config changed, update the config cache.
1044          */
1045         if (config_changed)
1046                 spa_config_sync(spa, B_FALSE, B_TRUE);
1047 }
______unchanged_portion_omitted_

1678 void
1679 spa_init(int mode)
1680 {
1681         mutex_init(&spa_namespace_lock, NULL, MUTEX_DEFAULT, NULL);
1682         mutex_init(&spa_spare_lock, NULL, MUTEX_DEFAULT, NULL);
1683         mutex_init(&spa_l2cache_lock, NULL, MUTEX_DEFAULT, NULL);
1684         cv_init(&spa_namespace_cv, NULL, CV_DEFAULT, NULL);

1686         avl_create(&spa_namespace_avl, spa_name_compare, sizeof (spa_t),
1687             offsetof(spa_t, spa_avl));

1689         avl_create(&spa_spare_avl, spa_spare_compare, sizeof (spa_aux_t),
1690             offsetof(spa_aux_t, aux_avl));

1692         avl_create(&spa_l2cache_avl, spa_l2cache_compare, sizeof (spa_aux_t),
1693             offsetof(spa_aux_t, aux_avl));

1695         spa_mode_global = mode;

1697 #ifdef _KERNEL
1698         spa_arch_init();
1699 #else
1700         if (spa_mode_global != FREAD && dprintf_find_string("watch")) {
1701                 arc_procfd = open("/proc/self/ctl", O_WRONLY);
1702                 if (arc_procfd == -1) {
1703                         perror("could not enable watchpoints: "
```

```
1704                                "opening /proc/self/ctl failed: ");
1705                } else {
1706                        arc_watch = B_TRUE;
1707                }
1708        }
1709 #endif

1711        refcount_init();
1712        unique_init();
1713        range_tree_init();
1705        space_map_init();
1714        zio_init();
1715        dmu_init();
1716        zil_init();
1717        vdev_cache_stat_init();
1718        zfs_prop_init();
1719        zpool_prop_init();
1720        zpool_feature_init();
1721        spa_config_load();
1722        l2arc_start();
1723 }

1725 void
1726 spa_fini(void)
1727 {
1728        l2arc_stop();

1730        spa_evict_all();

1732        vdev_cache_stat_fini();
1733        zil_fini();
1734        dmu_fini();
1735        zio_fini();
1736        range_tree_fini();
1728        space_map_fini();
1737        unique_fini();
1738        refcount_fini();

1740        avl_destroy(&spa_namespace_avl);
1741        avl_destroy(&spa_spare_avl);
1742        avl_destroy(&spa_l2cache_avl);

1744        cv_destroy(&spa_namespace_cv);
1745        mutex_destroy(&spa_namespace_lock);
1746        mutex_destroy(&spa_spare_lock);
1747        mutex_destroy(&spa_l2cache_lock);
1748 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   15953 Tue Sep  3 20:27:04 2013
new/usr/src/uts/common/fs/zfs/space_map.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

#include <sys/zfs_context.h>
#include <sys/spa.h>
#include <sys/dmu.h>
#include <sys/dmu_tx.h>
#include <sys/dnode.h>
#include <sys/dsl_pool.h>
#include <sys/zio.h>
#include <sys/space_map.h>
#include <sys/refcount.h>
#include <sys/zfeature.h>

static kmem_cache_t *space_seg_cache;

void
space_map_init(void)
{
        ASSERT(space_seg_cache == NULL);
        space_seg_cache = kmem_cache_create("space_seg_cache",
            sizeof (space_seg_t), 0, NULL, NULL, NULL, NULL, NULL, 0);
}

void
space_map_fini(void)
{
```



```
        kmem_cache_destroy(space_seg_cache);
        space_seg_cache = NULL;
}

/*
 * This value controls how the space map's block size is allowed to grow.
 * If the value is set to the same size as SPACE_MAP_INITIAL_BLOCKSIZE then
 * the space map block size will remain fixed. Setting this value to something
 * greater than SPACE_MAP_INITIAL_BLOCKSIZE will allow the space map to
 * increase its block size as needed. To maintain backwards compatibilty the
 * space map's block size must be a power of 2 and SPACE_MAP_INITIAL_BLOCKSIZE
 * or larger.
 * Space map routines.
 * NOTE: caller is responsible for all locking.
 */
int space_map_max_blksz = (1 << 12);
static int
space_map_seg_compare(const void *x1, const void *x2)
{
        const space_seg_t *s1 = x1;
        const space_seg_t *s2 = x2;

        if (s1->ss_start < s2->ss_start) {
                if (s1->ss_end > s2->ss_start)
                        return (0);
                return (-1);
        }
        if (s1->ss_start > s2->ss_start) {
                if (s1->ss_start < s2->ss_end)
                        return (0);
                return (1);
        }
        return (0);
}

void
space_map_create(space_map_t *sm, uint64_t start, uint64_t size, uint8_t shift,
        kmutex_t *lp)
{
        bzero(sm, sizeof (*sm));

        cv_init(&sm->sm_load_cv, NULL, CV_DEFAULT, NULL);

        avl_create(&sm->sm_root, space_map_seg_compare,
            sizeof (space_seg_t), offsetof(struct space_seg, ss_node));

        sm->sm_start = start;
        sm->sm_size = size;
        sm->sm_shift = shift;
        sm->sm_lock = lp;
}

void
space_map_destroy(space_map_t *sm)
{
        ASSERT(!sm->sm_loaded && !sm->sm_loading);
        VERIFY0(sm->sm_space);
        avl_destroy(&sm->sm_root);
        cv_destroy(&sm->sm_load_cv);
}

void
space_map_add(space_map_t *sm, uint64_t start, uint64_t size)
{
        avl_index_t where;
        space_seg_t *ss_before, *ss_after, *ss;
```

```
106         uint64_t end = start + size;
107         int merge_before, merge_after;
108
109         ASSERT(MUTEX_HELD(sm->sm_lock));
110         VERIFY(!sm->sm_condensing);
111         VERIFY(size != 0);
112         VERIFY3U(start, >=, sm->sm_start);
113         VERIFY3U(end, <=, sm->sm_start + sm->sm_size);
114         VERIFY(sm->sm_space + size <= sm->sm_size);
115         VERIFY(P2PHASE(start, 1ULL << sm->sm_shift) == 0);
116         VERIFY(P2PHASE(size, 1ULL << sm->sm_shift) == 0);
117
118         ss = space_map_find(sm, start, size, &where);
119         if (ss != NULL) {
120                 zfs_panic_recover("zfs: allocating allocated segment"
121                     "(offset=%llu size=%llu)\n",
122                     (longlong_t)start, (longlong_t)size);
123                 return;
124         }
125
126         /* Make sure we don't overlap with either of our neighbors */
127         VERIFY(ss == NULL);
128
129         ss_before = avl_nearest(&sm->sm_root, where, AVL_BEFORE);
130         ss_after = avl_nearest(&sm->sm_root, where, AVL_AFTER);
131
132         merge_before = (ss_before != NULL && ss_before->ss_end == start);
133         merge_after = (ss_after != NULL && ss_after->ss_start == end);
134
135         if (merge_before && merge_after) {
136                 avl_remove(&sm->sm_root, ss_before);
137                 if (sm->sm_pp_root) {
138                         avl_remove(sm->sm_pp_root, ss_before);
139                         avl_remove(sm->sm_pp_root, ss_after);
140                 }
141                 ss_after->ss_start = ss_before->ss_start;
142                 kmem_cache_free(space_seg_cache, ss_before);
143                 ss = ss_after;
144         } else if (merge_before) {
145                 ss_before->ss_end = end;
146                 if (sm->sm_pp_root)
147                         avl_remove(sm->sm_pp_root, ss_before);
148                 ss = ss_before;
149         } else if (merge_after) {
150                 ss_after->ss_start = start;
151                 if (sm->sm_pp_root)
152                         avl_remove(sm->sm_pp_root, ss_after);
153                 ss = ss_after;
154         } else {
155                 ss = kmem_cache_alloc(space_seg_cache, KM_SLEEP);
156                 ss->ss_start = start;
157                 ss->ss_end = end;
158                 avl_insert(&sm->sm_root, ss, where);
159         }
160
161         if (sm->sm_pp_root)
162                 avl_add(sm->sm_pp_root, ss);
163
164         sm->sm_space += size;
165 }
166
167 void
168 space_map_remove(space_map_t *sm, uint64_t start, uint64_t size)
169 {
170         avl_index_t where;
171         space_seg_t *ss, *newseg;
```



```
172         uint64_t end = start + size;
173         int left_over, right_over;
174
175         VERIFY(!sm->sm_condensing);
176         ss = space_map_find(sm, start, size, &where);
177
178         /* Make sure we completely overlap with someone */
179         if (ss == NULL) {
180                 zfs_panic_recover("zfs: freeing free segment "
181                     "(offset=%llu size=%llu)",
182                     (longlong_t)start, (longlong_t)size);
183                 return;
184         }
185         VERIFY3U(ss->ss_start, <=, start);
186         VERIFY3U(ss->ss_end, >=, end);
187         VERIFY(sm->sm_space - size <= sm->sm_size);
188
189         left_over = (ss->ss_start != start);
190         right_over = (ss->ss_end != end);
191
192         if (sm->sm_pp_root)
193                 avl_remove(sm->sm_pp_root, ss);
194
195         if (left_over && right_over) {
196                 newseg = kmem_cache_alloc(space_seg_cache, KM_SLEEP);
197                 newseg->ss_start = end;
198                 newseg->ss_end = ss->ss_end;
199                 ss->ss_end = start;
200                 avl_insert_here(&sm->sm_root, newseg, ss, AVL_AFTER);
201                 if (sm->sm_pp_root)
202                         avl_add(sm->sm_pp_root, newseg);
203         } else if (left_over) {
204                 ss->ss_end = start;
205         } else if (right_over) {
206                 ss->ss_start = end;
207         } else {
208                 avl_remove(&sm->sm_root, ss);
209                 kmem_cache_free(space_seg_cache, ss);
210                 ss = NULL;
211         }
212
213         if (sm->sm_pp_root && ss != NULL)
214                 avl_add(sm->sm_pp_root, ss);
215
216         sm->sm_space -= size;
217 }
218
219 space_seg_t *
220 space_map_find(space_map_t *sm, uint64_t start, uint64_t size,
221     avl_index_t *wherep)
222 {
223         space_seg_t ssearch, *ss;
224
225         ASSERT(MUTEX_HELD(sm->sm_lock));
226         VERIFY(size != 0);
227         VERIFY(P2PHASE(start, 1ULL << sm->sm_shift) == 0);
228         VERIFY(P2PHASE(size, 1ULL << sm->sm_shift) == 0);
229
230         ssearch.ss_start = start;
231         ssearch.ss_end = start + size;
232         ss = avl_find(&sm->sm_root, &ssearch, wherep);
233
234         if (ss != NULL && ss->ss_start <= start && ss->ss_end >= start + size)
235                 return (ss);
236         return (NULL);
237 }
```

```
239 boolean_t
240 space_map_contains(space_map_t *sm, uint64_t start, uint64_t size)
241 {
242         avl_index_t where;

244         return (space_map_find(sm, start, size, &where) != 0);
245 }

247 void
248 space_map_swap(space_map_t **msrc, space_map_t **mdst)
249 {
250         space_map_t *sm;

252         ASSERT(MUTEX_HELD((*msrc)->sm_lock));
253         ASSERT0((*mdst)->sm_space);
254         ASSERT0(avl_numnodes(&(*mdst)->sm_root));

256         sm = *msrc;
257         *msrc = *mdst;
258         *mdst = sm;
259 }

261 void
262 space_map_vacate(space_map_t *sm, space_map_func_t *func, space_map_t *mdest)
263 {
264         space_seg_t *ss;
265         void *cookie = NULL;

267         ASSERT(MUTEX_HELD(sm->sm_lock));

269         while ((ss = avl_destroy_nodes(&sm->sm_root, &cookie)) != NULL) {
270                 if (func != NULL)
271                         func(mdest, ss->ss_start, ss->ss_end - ss->ss_start);
272                 kmem_cache_free(space_seg_cache, ss);
273         }
274         sm->sm_space = 0;
275 }

277 void
278 space_map_walk(space_map_t *sm, space_map_func_t *func, space_map_t *mdest)
279 {
280         space_seg_t *ss;

282         ASSERT(MUTEX_HELD(sm->sm_lock));

284         for (ss = avl_first(&sm->sm_root); ss; ss = AVL_NEXT(&sm->sm_root, ss))
285                 func(mdest, ss->ss_start, ss->ss_end - ss->ss_start);
286 }

 51 /*
 52  * Load the space map disk into the specified range tree. Segments of maptype
 53  * are added to the range tree, other segment types are removed.
 54  *
289  * Wait for any in-progress space_map_load() to complete.
290  */
291 void
292 space_map_load_wait(space_map_t *sm)
293 {
294         ASSERT(MUTEX_HELD(sm->sm_lock));

296         while (sm->sm_loading) {
297                 ASSERT(!sm->sm_loaded);
298                 cv_wait(&sm->sm_load_cv, sm->sm_lock);
299         }
300 }
```



```
302 /*
 55  * Note: space_map_load() will drop sm_lock across dmu_read() calls.
 56  * The caller must be OK with this.
 57  */
 58 int
 59 space_map_load(space_map_t *sm, range_tree_t *rt, maptype_t maptype)
307 space_map_load(space_map_t *sm, space_map_ops_t *ops, uint8_t maptype,
308         space_map_obj_t *smo, objset_t *os)
 60 {
 61         uint64_t *entry, *entry_map, *entry_map_end;
 62         uint64_t bufsize, size, offset, end, space;
312         uint64_t mapstart = sm->sm_start;
 63         int error = 0;

 65         ASSERT(MUTEX_HELD(sm->sm_lock));
316         ASSERT(!sm->sm_loaded);
317         ASSERT(!sm->sm_loading);

 67         end = space_map_length(sm);
 68         space = space_map_allocated(sm);
319         sm->sm_loading = B_TRUE;
320         end = smo->smo_objsize;
321         space = smo->smo_alloc;

 70         VERIFY0(range_tree_space(rt));
323         ASSERT(sm->sm_ops == NULL);
324         VERIFY0(sm->sm_space);

 72         if (maptype == SM_FREE) {
 73                 range_tree_add(rt, sm->sm_start, sm->sm_size);
327                 space_map_add(sm, sm->sm_start, sm->sm_size);
 74                 space = sm->sm_size - space;
 75         }

 77         bufsize = MAX(sm->sm_blksz, SPA_MINBLOCKSIZE);
331         bufsize = 1ULL << SPACE_MAP_BLOCKSHIFT;
 78         entry_map = zio_buf_alloc(bufsize);

 80         mutex_exit(sm->sm_lock);
 81         if (end > bufsize) {
 82                 dmu_prefetch(sm->sm_os, space_map_object(sm), bufsize,
 83                     end - bufsize);
 84         }
335         if (end > bufsize)
336                 dmu_prefetch(os, smo->smo_object, bufsize, end - bufsize);
 85         mutex_enter(sm->sm_lock);

 87         for (offset = 0; offset < end; offset += bufsize) {
 88                 size = MIN(end - offset, bufsize);
 89                 VERIFY(P2PHASE(size, sizeof (uint64_t)) == 0);
 90                 VERIFY(size != 0);
 91                 ASSERT3U(sm->sm_blksz, !=, 0);

 93                 dprintf("object=%llu  offset=%llx  size=%llx\n",
 94                     space_map_object(sm), offset, size);
345                     smo->smo_object, offset, size);

 96                 mutex_exit(sm->sm_lock);
 97                 error = dmu_read(sm->sm_os, space_map_object(sm), offset, size,
 98                     entry_map, DMU_READ_PREFETCH);
348                 error = dmu_read(os, smo->smo_object, offset, size, entry_map,
349                     DMU_READ_PREFETCH);
 99                 mutex_enter(sm->sm_lock);
100                 if (error != 0)
101                         break;
```

```
103                 entry_map_end = entry_map + (size / sizeof (uint64_t));
104                 for (entry = entry_map; entry < entry_map_end; entry++) {
105                         uint64_t e = *entry;
106                         uint64_t offset, size;

108                         if (SM_DEBUG_DECODE(e))         /* Skip debug entries */
109                                 continue;

111                         offset = (SM_OFFSET_DECODE(e) << sm->sm_shift) +
112                             sm->sm_start;
113                         size = SM_RUN_DECODE(e) << sm->sm_shift;

115                         VERIFY0(P2PHASE(offset, 1ULL << sm->sm_shift));
116                         VERIFY0(P2PHASE(size, 1ULL << sm->sm_shift));
117                         VERIFY3U(offset, >=, sm->sm_start);
118                         VERIFY3U(offset + size, <=, sm->sm_start + sm->sm_size);
119                         if (SM_TYPE_DECODE(e) == maptype) {
120                                 VERIFY3U(range_tree_space(rt) + size, <=,
121                                     sm->sm_size);
122                                 range_tree_add(rt, offset, size);
123                         } else {
124                                 range_tree_remove(rt, offset, size);
361                         (SM_TYPE_DECODE(e) == maptype ?
362                             space_map_add : space_map_remove)(sm,
363                             (SM_OFFSET_DECODE(e) << sm->sm_shift) + mapstart,
364                             SM_RUN_DECODE(e) << sm->sm_shift);
125                         }
126                 }
127         }

129         if (error == 0)
130                 VERIFY3U(range_tree_space(rt), ==, space);
131         else
132                 range_tree_vacate(rt, NULL, NULL);
368         if (error == 0) {
369                 VERIFY3U(sm->sm_space, ==, space);

371                 sm->sm_loaded = B_TRUE;
372                 sm->sm_ops = ops;
373                 if (ops != NULL)
374                         ops->smop_load(sm);
375         } else {
376                 space_map_vacate(sm, NULL, NULL);
377         }

134         zio_buf_free(entry_map, bufsize);
135         return (error);
136 }

138 void
139 space_map_histogram_clear(space_map_t *sm)
140 {
141         if (sm->sm_dbuf->db_size != sizeof (space_map_phys_t))
142                 return;
381         sm->sm_loading = B_FALSE;

144         bzero(sm->sm_phys->smp_histogram, sizeof (sm->sm_phys->smp_histogram));
145 }
383         cv_broadcast(&sm->sm_load_cv);

147 boolean_t
148 space_map_histogram_verify(space_map_t *sm, range_tree_t *rt)
149 {
150         /*
151          * Verify that the in-core range tree does not have any
```



```
152          * ranges smaller than our sm_shift size.
153          */
154         for (int i = 0; i < sm->sm_shift; i++) {
155                 if (rt->rt_histogram[i] != 0)
156                         return (B_FALSE);
157         }
158         return (B_TRUE);
385         return (error);
159 }

161 void
162 space_map_histogram_add(space_map_t *sm, range_tree_t *rt, dmu_tx_t *tx)
389 space_map_unload(space_map_t *sm)
163 {
164         int idx = 0;
391         ASSERT(MUTEX_HELD(sm->sm_lock));

166         ASSERT(MUTEX_HELD(rt->rt_lock));
167         ASSERT(dmu_tx_is_syncing(tx));
168         VERIFY3U(space_map_object(sm), !=, 0);
393         if (sm->sm_loaded && sm->sm_ops != NULL)
394                 sm->sm_ops->smop_unload(sm);

170         if (sm->sm_dbuf->db_size != sizeof (space_map_phys_t))
171                 return;
396         sm->sm_loaded = B_FALSE;
397         sm->sm_ops = NULL;

173         dmu_buf_will_dirty(sm->sm_dbuf, tx);
399         space_map_vacate(sm, NULL, NULL);
400 }

175         ASSERT(space_map_histogram_verify(sm, rt));

177         /*
178          * Transfer the content of the range tree histogram to the space
179          * map histogram. The space map histogram contains 32 buckets ranging
180          * between 2^sm_shift to 2^(32+sm_shift-1). The range tree,
181          * however, can represent ranges from 2^0 to 2^63. Since the space
182          * map only cares about allocatable blocks (minimum of sm_shift) we
183          * can safely ignore all ranges in the range tree smaller than sm_shift.
184          */
185         for (int i = sm->sm_shift; i < RANGE_TREE_HISTOGRAM_SIZE; i++) {

187                 /*
188                  * Since the largest histogram bucket in the space map is
189                  * 2^(32+sm_shift-1), we need to normalize the values in
190                  * the range tree for any bucket larger than that size. For
191                  * example given an sm_shift of 9, ranges larger than 2^40
192                  * would get normalized as if they were 1TB ranges. Assume
193                  * the range tree had a count of 5 in the 2^44 (16TB) bucket,
194                  * the calculation below would normalize this to 5 * 2^4 (16).
195                  */
196                 ASSERT3U(i, >=, idx + sm->sm_shift);
197                 sm->sm_phys->smp_histogram[idx] +=
198                     rt->rt_histogram[i] << (i - idx - sm->sm_shift);

200                 /*
201                  * Increment the space map's index as long as we haven't
202                  * reached the maximum bucket size. Accumulate all ranges
203                  * larger than the max bucket size into the last bucket.
204                  */
205                 if (idx < SPACE_MAP_HISTOGRAM_SIZE(sm) - 1) {
206                         ASSERT3U(idx + sm->sm_shift, ==, i);
207                         idx++;
208                         ASSERT3U(idx, <, SPACE_MAP_HISTOGRAM_SIZE(sm));
```

```
209                 }
210         }
402 uint64_t
403 space_map_maxsize(space_map_t *sm)
404 {
405         ASSERT(sm->sm_ops != NULL);
406         return (sm->sm_ops->smop_max(sm));
211 }

213 uint64_t
214 space_map_entries(space_map_t *sm, range_tree_t *rt)
410 space_map_alloc(space_map_t *sm, uint64_t size)
215 {
216         avl_tree_t *t = &rt->rt_root;
217         range_seg_t *rs;
218         uint64_t size, entries;
412         uint64_t start;

220         /*
221          * All space_maps always have a debug entry so account for it here.
222          */
223         entries = 1;

225         /*
226          * Traverse the range tree and calculate the number of space map
227          * entries that would be required to write out the range tree.
228          */
229         for (rs = avl_first(t); rs != NULL; rs = AVL_NEXT(t, rs)) {
230                 size = (rs->rs_end - rs->rs_start) >> sm->sm_shift;
231                 entries += howmany(size, SM_RUN_MAX);
232         }
233         return (entries);
414         start = sm->sm_ops->smop_alloc(sm, size);
415         if (start != -1ULL)
416                 space_map_remove(sm, start, size);
417         return (start);
234 }

236 void
237 space_map_set_blocksize(space_map_t *sm, uint64_t size, dmu_tx_t *tx)
421 space_map_claim(space_map_t *sm, uint64_t start, uint64_t size)
238 {
239         uint32_t blksz;
240         u_longlong_t blocks;
423         sm->sm_ops->smop_claim(sm, start, size);
424         space_map_remove(sm, start, size);
425 }

242         ASSERT3U(sm->sm_blksz, !=, 0);
243         ASSERT3U(space_map_object(sm), !=, 0);
244         ASSERT(sm->sm_dbuf != NULL);
245         VERIFY(ISP2(space_map_max_blksz));

247         if (sm->sm_blksz >= space_map_max_blksz)
248                 return;

250         /*
251          * The object contains more than one block so we can't adjust
252          * its size.
253          */
254         if (sm->sm_phys->smp_objsize > sm->sm_blksz)
255                 return;

257         if (size > sm->sm_blksz) {
258                 uint64_t newsz;
```



```
260                 /*
261                  * Older software versions treat space map blocks as fixed
262                  * entities. The DMU is capable of handling different block
263                  * sizes making it possible for us to increase the
264                  * block size and maintain backwards compatibility. The
265                  * caveat is that the new block sizes must be a
266                  * power of 2 so that old software can append to the file,
267                  * adding more blocks. The block size can grow until it
268                  * reaches space_map_max_blksz.
269                  */
270                 newsz = ISP2(size) ? size : 1ULL << highbit(size);
271                 if (newsz > space_map_max_blksz)
272                         newsz = space_map_max_blksz;

274                 VERIFY0(dmu_object_set_blocksize(sm->sm_os,
275                     space_map_object(sm), newsz, 0, tx));
276                 dmu_object_size_from_db(sm->sm_dbuf, &blksz, &blocks);

278                 zfs_dbgmsg("txg %llu, spa %s, increasing blksz from %d to %d",
279                     dmu_tx_get_txg(tx), spa_name(dmu_objset_spa(sm->sm_os)),
280                     sm->sm_blksz, blksz);

282                 VERIFY3U(newsz, ==, blksz);
283                 VERIFY3U(sm->sm_blksz, <, blksz);
284                 sm->sm_blksz = blksz;
285         }
427 void
428 space_map_free(space_map_t *sm, uint64_t start, uint64_t size)
429 {
430         space_map_add(sm, start, size);
431         sm->sm_ops->smop_free(sm, start, size);
286 }

288 /*
289  * Note: space_map_write() will drop sm_lock across dmu_write() calls.
435  * Note: space_map_sync() will drop sm_lock across dmu_write() calls.
290  */
291 void
292 space_map_write(space_map_t *sm, range_tree_t *rt, maptype_t maptype,
293     dmu_tx_t *tx)
438 space_map_sync(space_map_t *sm, uint8_t maptype,
439         space_map_obj_t *smo, objset_t *os, dmu_tx_t *tx)
294 {
295         objset_t *os = sm->sm_os;
296         spa_t *spa = dmu_objset_spa(os);
297         avl_tree_t *t = &rt->rt_root;
298         range_seg_t *rs;
299         uint64_t size, total, rt_space, nodes;
442         avl_tree_t *t = &sm->sm_root;
443         space_seg_t *ss;
444         uint64_t bufsize, start, size, run_len, total, sm_space, nodes;
300         uint64_t *entry, *entry_map, *entry_map_end;
301         uint64_t newsz, expected_entries, actual_entries = 1;

303         ASSERT(MUTEX_HELD(rt->rt_lock));
304         ASSERT(dsl_pool_sync_context(dmu_objset_pool(os)));
305         VERIFY3U(space_map_object(sm), !=, 0);
306         dmu_buf_will_dirty(sm->sm_dbuf, tx);
447         ASSERT(MUTEX_HELD(sm->sm_lock));

308         /*
309          * This field is no longer necessary since the in-core space map
310          * now contains the object number but is maintained for backwards
311          * compatibility.
312          */
313         sm->sm_phys->smp_object = sm->sm_object;
```

```
315         if (range_tree_space(rt) == 0) {
316                 VERIFY3U(sm->sm_object, ==, sm->sm_phys->smp_object);
449         if (sm->sm_space == 0)
317                 return;
318         }

452         dprintf("object %4llu, txg %llu, pass %d, %c, count %lu, space %llx\n",
453             smo->smo_object, dmu_tx_get_txg(tx), spa_sync_pass(spa),
454             maptype == SM_ALLOC ? 'A' : 'F', avl_numnodes(&sm->sm_root),
455             sm->sm_space);

320         if (maptype == SM_ALLOC)
321                 sm->sm_phys->smp_alloc += range_tree_space(rt);
458                 smo->smo_alloc += sm->sm_space;
322         else
323                 sm->sm_phys->smp_alloc -= range_tree_space(rt);
460                 smo->smo_alloc -= sm->sm_space;

325         expected_entries = space_map_entries(sm, rt);

327         /*
328          * Calculate the new size for the space map on-disk and see if
329          * we can grow the block size to accommodate the new size.
330          */
331         newsz = sm->sm_phys->smp_objsize + expected_entries * sizeof (uint64_t);
332         space_map_set_blocksize(sm, newsz, tx);

334         entry_map = zio_buf_alloc(sm->sm_blksz);
335         entry_map_end = entry_map + (sm->sm_blksz / sizeof (uint64_t));
462         bufsize = (8 + avl_numnodes(&sm->sm_root)) * sizeof (uint64_t);
463         bufsize = MIN(bufsize, 1ULL << SPACE_MAP_BLOCKSHIFT);
464         entry_map = zio_buf_alloc(bufsize);
465         entry_map_end = entry_map + (bufsize / sizeof (uint64_t));
336         entry = entry_map;

338         *entry++ = SM_DEBUG_ENCODE(1) |
339             SM_DEBUG_ACTION_ENCODE(maptype) |
340             SM_DEBUG_SYNCPASS_ENCODE(spa_sync_pass(spa)) |
341             SM_DEBUG_TXG_ENCODE(dmu_tx_get_txg(tx));

343         total = 0;
344         nodes = avl_numnodes(&rt->rt_root);
345         rt_space = range_tree_space(rt);
346         for (rs = avl_first(t); rs != NULL; rs = AVL_NEXT(t, rs)) {
347                 uint64_t start;
474         nodes = avl_numnodes(&sm->sm_root);
475         sm_space = sm->sm_space;
476         for (ss = avl_first(t); ss != NULL; ss = AVL_NEXT(t, ss)) {
477                 size = ss->ss_end - ss->ss_start;
478                 start = (ss->ss_start - sm->sm_start) >> sm->sm_shift;

349                 size = (rs->rs_end - rs->rs_start) >> sm->sm_shift;
350                 start = (rs->rs_start - sm->sm_start) >> sm->sm_shift;
480                 total += size;
481                 size >>= sm->sm_shift;

352                 total += size << sm->sm_shift;

354                 while (size != 0) {
355                         uint64_t run_len;

483                 while (size) {
357                         run_len = MIN(size, SM_RUN_MAX);

359                         if (entry == entry_map_end) {
```



```
360                             mutex_exit(rt->rt_lock);
361                             dmu_write(os, space_map_object(sm),
362                                 sm->sm_phys->smp_objsize, sm->sm_blksz,
363                                 entry_map, tx);
364                             mutex_enter(rt->rt_lock);
365                             sm->sm_phys->smp_objsize += sm->sm_blksz;
487                             mutex_exit(sm->sm_lock);
488                             dmu_write(os, smo->smo_object, smo->smo_objsize,
489                                 bufsize, entry_map, tx);
490                             mutex_enter(sm->sm_lock);
491                             smo->smo_objsize += bufsize;
366                             entry = entry_map;
367                     }

369                     *entry++ = SM_OFFSET_ENCODE(start) |
370                         SM_TYPE_ENCODE(maptype) |
371                         SM_RUN_ENCODE(run_len);

373                     start += run_len;
374                     size -= run_len;
375                     actual_entries++;
376             }
377         }

379         if (entry != entry_map) {
380                 size = (entry - entry_map) * sizeof (uint64_t);
381                 mutex_exit(rt->rt_lock);
382                 dmu_write(os, space_map_object(sm), sm->sm_phys->smp_objsize,
506                 mutex_exit(sm->sm_lock);
507                 dmu_write(os, smo->smo_object, smo->smo_objsize,
383                     size, entry_map, tx);
384                 mutex_enter(rt->rt_lock);
385                 sm->sm_phys->smp_objsize += size;
509                 mutex_enter(sm->sm_lock);
510                 smo->smo_objsize += size;
386         }
387         ASSERT3U(expected_entries, ==, actual_entries);

389         /*
390          * Ensure that the space_map's accounting wasn't changed
391          * while we were in the middle of writing it out.
392          */
393         VERIFY3U(nodes, ==, avl_numnodes(&rt->rt_root));
394         VERIFY3U(range_tree_space(rt), ==, rt_space);
395         VERIFY3U(range_tree_space(rt), ==, total);
517         VERIFY3U(nodes, ==, avl_numnodes(&sm->sm_root));
518         VERIFY3U(sm->sm_space, ==, sm_space);
519         VERIFY3U(sm->sm_space, ==, total);

397         zio_buf_free(entry_map, sm->sm_blksz);
521         zio_buf_free(entry_map, bufsize);
398 }

400 static int
401 space_map_open_impl(space_map_t *sm)
524 void
525 space_map_truncate(space_map_obj_t *smo, objset_t *os, dmu_tx_t *tx)
402 {
403         int error;
404         u_longlong_t blocks;
527         VERIFY(dmu_free_range(os, smo->smo_object, 0, -1ULL, tx) == 0);

406         error = dmu_bonus_hold(sm->sm_os, sm->sm_object, sm, &sm->sm_dbuf);
407         if (error)
408                 return (error);
```

```
410         dmu_object_size_from_db(sm->sm_dbuf, &sm->sm_blksz, &blocks);
411         sm->sm_phys = sm->sm_dbuf->db_data;
412         return (0);
529         smo->smo_objsize = 0;
530         smo->smo_alloc = 0;
413 }

415 int
416 space_map_open(space_map_t **smp, objset_t *os, uint64_t object,
417     uint64_t start, uint64_t size, uint8_t shift, kmutex_t *lp)
533 /*
534  * Space map reference trees.
535  *
536  * A space map is a collection of integers.  Every integer is either
537  * in the map, or it's not.  A space map reference tree generalizes
538  * the idea: it allows its members to have arbitrary reference counts,
539  * as opposed to the implicit reference count of 0 or 1 in a space map.
540  * This representation comes in handy when computing the union or
541  * intersection of multiple space maps.  For example, the union of
542  * N space maps is the subset of the reference tree with refcnt >= 1.
543  * The intersection of N space maps is the subset with refcnt >= N.
544  *
545  * [It's very much like a Fourier transform.  Unions and intersections
546  * are hard to perform in the 'space map domain', so we convert the maps
547  * into the 'reference count domain', where it's trivial, then invert.]
548  *
549  * vdev_dtl_reassess() uses computations of this form to determine
550  * DTL_MISSING and DTL_OUTAGE for interior vdevs -- e.g. a RAID-Z vdev
551  * has an outage wherever refcnt >= vdev_nparity + 1, and a mirror vdev
552  * has an outage wherever refcnt >= vdev_children.
553  */
554 static int
555 space_map_ref_compare(const void *x1, const void *x2)
418 {
419         space_map_t *sm;
420         int error;
557         const space_ref_t *sr1 = x1;
558         const space_ref_t *sr2 = x2;

422         ASSERT(*smp == NULL);
423         ASSERT(os != NULL);
424         ASSERT(object != 0);
560         if (sr1->sr_offset < sr2->sr_offset)
561                 return (-1);
562         if (sr1->sr_offset > sr2->sr_offset)
563                 return (1);

426         sm = kmem_zalloc(sizeof (space_map_t), KM_SLEEP);
565         if (sr1 < sr2)
566                 return (-1);
567         if (sr1 > sr2)
568                 return (1);

428         sm->sm_start = start;
429         sm->sm_size = size;
430         sm->sm_shift = shift;
431         sm->sm_lock = lp;
432         sm->sm_os = os;
433         sm->sm_object = object;

435         error = space_map_open_impl(sm);
436         if (error != 0) {
437                 space_map_close(sm);
438                 return (error);
439         }
```



```
441         *smp = sm;

443         return (0);
444 }

446 void
447 space_map_close(space_map_t *sm)
574 space_map_ref_create(avl_tree_t *t)
448 {
449         if (sm == NULL)
450                 return;
576         avl_create(t, space_map_ref_compare,
577             sizeof (space_ref_t), offsetof(space_ref_t, sr_node));
578 }

452         if (sm->sm_dbuf != NULL)
453                 dmu_buf_rele(sm->sm_dbuf, sm);
454         sm->sm_dbuf = NULL;
455         sm->sm_phys = NULL;
580 void
581 space_map_ref_destroy(avl_tree_t *t)
582 {
583         space_ref_t *sr;
584         void *cookie = NULL;

457         kmem_free(sm, sizeof (*sm));
586         while ((sr = avl_destroy_nodes(t, &cookie)) != NULL)
587                 kmem_free(sr, sizeof (*sr));

589         avl_destroy(t);
458 }

460 static void
461 space_map_reallocate(space_map_t *sm, dmu_tx_t *tx)
593 space_map_ref_add_node(avl_tree_t *t, uint64_t offset, int64_t refcnt)
462 {
463         ASSERT(dmu_tx_is_syncing(tx));
595         space_ref_t *sr;

465         space_map_free(sm, tx);
466         dmu_buf_rele(sm->sm_dbuf, sm);
597         sr = kmem_alloc(sizeof (*sr), KM_SLEEP);
598         sr->sr_offset = offset;
599         sr->sr_refcnt = refcnt;

468         sm->sm_object = space_map_alloc(sm->sm_os, tx);
469         VERIFY0(space_map_open_impl(sm));
601         avl_add(t, sr);
470 }

472 void
473 space_map_truncate(space_map_t *sm, dmu_tx_t *tx)
605 space_map_ref_add_seg(avl_tree_t *t, uint64_t start, uint64_t end,
606         int64_t refcnt)
474 {
475         objset_t *os = sm->sm_os;
476         spa_t *spa = dmu_objset_spa(os);
477         zfeature_info_t *space_map_histogram =
478             &spa_feature_table[SPA_FEATURE_SPACEMAP_HISTOGRAM];
479         dmu_object_info_t doi;
480         int bonuslen;

482         ASSERT(dsl_pool_sync_context(dmu_objset_pool(os)));
483         ASSERT(dmu_tx_is_syncing(tx));

485         VERIFY0(dmu_free_range(os, space_map_object(sm), 0, -1ULL, tx));
```

```
        dmu_object_info_from_db(sm->sm_dbuf, &doi);

        if (spa_feature_is_enabled(spa, space_map_histogram)) {
                bonuslen = sizeof (space_map_phys_t);
                ASSERT3U(bonuslen, <=, dmu_bonus_max());
        } else {
                bonuslen = SPACE_MAP_SIZE_V0;
        }

        if (bonuslen != doi.doi_bonus_size ||
            doi.doi_data_block_size != SPACE_MAP_INITIAL_BLOCKSIZE) {
                zfs_dbgmsg("txg %llu, spa %s, reallocating: "
                    "old bonus %u, old blocksz %u", dmu_tx_get_txg(tx),
                    spa_name(spa), doi.doi_bonus_size, doi.doi_data_block_size);
                space_map_reallocate(sm, tx);
                VERIFY3U(sm->sm_blksz, ==, SPACE_MAP_INITIAL_BLOCKSIZE);
        }

        dmu_buf_will_dirty(sm->sm_dbuf, tx);
        sm->sm_phys->smp_objsize = 0;
        sm->sm_phys->smp_alloc = 0;
        space_map_ref_add_node(t, start, refcnt);
        space_map_ref_add_node(t, end, -refcnt);
}

/*
 * Update the in-core space_map allocation and length values.
 * Convert (or add) a space map into a reference tree.
 */
void
space_map_update(space_map_t *sm)
space_map_ref_add_map(avl_tree_t *t, space_map_t *sm, int64_t refcnt)
{
        if (sm == NULL)
                return;
        space_seg_t *ss;

        ASSERT(MUTEX_HELD(sm->sm_lock));

        sm->sm_alloc = sm->sm_phys->smp_alloc;
        sm->sm_length = sm->sm_phys->smp_objsize;
        for (ss = avl_first(&sm->sm_root); ss; ss = AVL_NEXT(&sm->sm_root, ss))
                space_map_ref_add_seg(t, ss->ss_start, ss->ss_end, refcnt);
}

uint64_t
space_map_alloc(objset_t *os, dmu_tx_t *tx)
{
        spa_t *spa = dmu_objset_spa(os);
        zfeature_info_t *space_map_histogram =
            &spa_feature_table[SPA_FEATURE_SPACEMAP_HISTOGRAM];
        uint64_t object;
        int bonuslen;

        if (spa_feature_is_enabled(spa, space_map_histogram)) {
                spa_feature_incr(spa, space_map_histogram, tx);
                bonuslen = sizeof (space_map_phys_t);
                ASSERT3U(bonuslen, <=, dmu_bonus_max());
        } else {
                bonuslen = SPACE_MAP_SIZE_V0;
        }

        object = dmu_object_alloc(os,
            DMU_OT_SPACE_MAP, SPACE_MAP_INITIAL_BLOCKSIZE,
            DMU_OT_SPACE_MAP_HEADER, bonuslen, tx);
```



```
        return (object);
}

/*
 * Convert a reference tree into a space map.  The space map will contain
 * all members of the reference tree for which refcnt >= minref.
 */
void
space_map_free(space_map_t *sm, dmu_tx_t *tx)
space_map_ref_generate_map(avl_tree_t *t, space_map_t *sm, int64_t minref)
{
        spa_t *spa;
        zfeature_info_t *space_map_histogram =
            &spa_feature_table[SPA_FEATURE_SPACEMAP_HISTOGRAM];
        uint64_t start = -1ULL;
        int64_t refcnt = 0;
        space_ref_t *sr;

        if (sm == NULL)
                return;
        ASSERT(MUTEX_HELD(sm->sm_lock));

        spa = dmu_objset_spa(sm->sm_os);
        if (spa_feature_is_enabled(spa, space_map_histogram)) {
                dmu_object_info_t doi;
        space_map_vacate(sm, NULL, NULL);

                dmu_object_info_from_db(sm->sm_dbuf, &doi);
                if (doi.doi_bonus_size != SPACE_MAP_SIZE_V0) {
                        VERIFY(spa_feature_is_active(spa, space_map_histogram));
                        spa_feature_decr(spa, space_map_histogram, tx);
        for (sr = avl_first(t); sr != NULL; sr = AVL_NEXT(t, sr)) {
                refcnt += sr->sr_refcnt;
                if (refcnt >= minref) {
                        if (start == -1ULL) {
                                start = sr->sr_offset;
                }
                } else {
                        if (start != -1ULL) {
                                uint64_t end = sr->sr_offset;
                                ASSERT(start <= end);
                                if (end > start)
                                        space_map_add(sm, start, end - start);
                                start = -1ULL;
        }

        VERIFY3U(dmu_object_free(sm->sm_os, space_map_object(sm), tx), ==, 0);
        sm->sm_object = 0;
}

uint64_t
space_map_object(space_map_t *sm)
{
        return (sm != NULL ? sm->sm_object : 0);
}

/*
 * Returns the already synced, on-disk allocated space.
 */
uint64_t
space_map_allocated(space_map_t *sm)
{
        return (sm != NULL ? sm->sm_alloc : 0);
}

/*
```

```
 * Returns the already synced, on-disk length;
 */
uint64_t
space_map_length(space_map_t *sm)
{
        return (sm != NULL ? sm->sm_length : 0);
}

/*
 * Returns the allocated space that is currently syncing.
 */
int64_t
space_map_alloc_delta(space_map_t *sm)
{
        if (sm == NULL)
                return (0);
        ASSERT(sm->sm_dbuf != NULL);
        return (sm->sm_phys->smp_alloc - space_map_allocated(sm));
                }
        }
        ASSERT(refcnt == 0);
        ASSERT(start == -1ULL);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    4308 Tue Sep  3 20:27:05 2013
new/usr/src/uts/common/fs/zfs/space_reftree.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <sys/zfs_context.h>
#include <sys/range_tree.h>
#include <sys/space_reftree.h>

/*
 * Space reference trees.
 *
 * A range tree is a collection of integers.  Every integer is either
 * in the tree, or it's not.  A space reference tree generalizes
 * the idea: it allows its members to have arbitrary reference counts,
 * as opposed to the implicit reference count of 0 or 1 in a range tree.
 * This representation comes in handy when computing the union or
 * intersection of multiple space maps.  For example, the union of
 * N range trees is the subset of the reference tree with refcnt >= 1.
 * The intersection of N range trees is the subset with refcnt >= N.
 *
 * [It's very much like a Fourier transform.  Unions and intersections
 * are hard to perform in the 'range tree domain', so we convert the trees
 * into the 'reference count domain', where it's trivial, then invert.]
 *
 * vdev_dtl_reassess() uses computations of this form to determine
 * DTL_MISSING and DTL_OUTAGE for interior vdevs -- e.g. a RAID-Z vdev
 * has an outage wherever refcnt >= vdev_nparity + 1, and a mirror vdev
 * has an outage wherever refcnt >= vdev_children.
 */
```



```
static int
space_reftree_compare(const void *x1, const void *x2)
{
        const space_ref_t *sr1 = x1;
        const space_ref_t *sr2 = x2;

        if (sr1->sr_offset < sr2->sr_offset)
                return (-1);
        if (sr1->sr_offset > sr2->sr_offset)
                return (1);

        if (sr1 < sr2)
                return (-1);
        if (sr1 > sr2)
                return (1);

        return (0);
}

void
space_reftree_create(avl_tree_t *t)
{
        avl_create(t, space_reftree_compare,
            sizeof (space_ref_t), offsetof(space_ref_t, sr_node));
}

void
space_reftree_destroy(avl_tree_t *t)
{
        space_ref_t *sr;
        void *cookie = NULL;

        while ((sr = avl_destroy_nodes(t, &cookie)) != NULL)
                kmem_free(sr, sizeof (*sr));

        avl_destroy(t);
}

static void
space_reftree_add_node(avl_tree_t *t, uint64_t offset, int64_t refcnt)
{
        space_ref_t *sr;

        sr = kmem_alloc(sizeof (*sr), KM_SLEEP);
        sr->sr_offset = offset;
        sr->sr_refcnt = refcnt;

        avl_add(t, sr);
}

void
space_reftree_add_seg(avl_tree_t *t, uint64_t start, uint64_t end,
        int64_t refcnt)
{
        space_reftree_add_node(t, start, refcnt);
        space_reftree_add_node(t, end, -refcnt);
}

/*
 * Convert (or add) a range tree into a reference tree.
 */
void
space_reftree_add_map(avl_tree_t *t, range_tree_t *rt, int64_t refcnt)
{
        range_seg_t *rs;
```

```
120         ASSERT(MUTEX_HELD(rt->rt_lock));
121
122         for (rs = avl_first(&rt->rt_root); rs; rs = AVL_NEXT(&rt->rt_root, rs))
123                 space_reftree_add_seg(t, rs->rs_start, rs->rs_end, refcnt);
124 }
125
126 /*
127  * Convert a reference tree into a range tree.  The range tree will contain
128  * all members of the reference tree for which refcnt >= minref.
129  */
130 void
131 space_reftree_generate_map(avl_tree_t *t, range_tree_t *rt, int64_t minref)
132 {
133         uint64_t start = -1ULL;
134         int64_t refcnt = 0;
135         space_ref_t *sr;
136
137         ASSERT(MUTEX_HELD(rt->rt_lock));
138
139         range_tree_vacate(rt, NULL, NULL);
140
141         for (sr = avl_first(t); sr != NULL; sr = AVL_NEXT(t, sr)) {
142                 refcnt += sr->sr_refcnt;
143                 if (refcnt >= minref) {
144                         if (start == -1ULL) {
145                                 start = sr->sr_offset;
146                         }
147                 } else {
148                         if (start != -1ULL) {
149                                 uint64_t end = sr->sr_offset;
150                                 ASSERT(start <= end);
151                                 if (end > start)
152                                         range_tree_add(rt, start, end - start);
153                                 start = -1ULL;
154                         }
155                 }
156         }
157         ASSERT(refcnt == 0);
158         ASSERT(start == -1ULL);
159 }
```

```
**********************************************************
    3187 Tue Sep  3 20:27:06 2013
new/usr/src/uts/common/fs/zfs/sys/metaslab.h
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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 (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

#ifndef _SYS_METASLAB_H
#define _SYS_METASLAB_H

#include <sys/spa.h>
#include <sys/space_map.h>
#include <sys/txg.h>
#include <sys/zio.h>
#include <sys/avl.h>

#ifdef  __cplusplus
extern "C" {
#endif

typedef struct metaslab_ops {
        uint64_t (*msop_alloc)(metaslab_t *msp, uint64_t size);
        boolean_t (*msop_fragmented)(metaslab_t *msp);
} metaslab_ops_t;
extern space_map_ops_t *zfs_metaslab_ops;

extern metaslab_ops_t *zfs_metaslab_ops;
extern metaslab_t *metaslab_init(metaslab_group_t *mg, space_map_obj_t *smo,
    uint64_t start, uint64_t size, uint64_t txg);
extern void metaslab_fini(metaslab_t *msp);
extern void metaslab_sync(metaslab_t *msp, uint64_t txg);
extern void metaslab_sync_done(metaslab_t *msp, uint64_t txg);
extern void metaslab_sync_reassess(metaslab_group_t *mg);
```



```
metaslab_t *metaslab_init(metaslab_group_t *mg, uint64_t id,
    uint64_t object, uint64_t txg);
void metaslab_fini(metaslab_t *msp);

void metaslab_load_wait(metaslab_t *msp);
int metaslab_load(metaslab_t *msp);
void metaslab_unload(metaslab_t *msp);

void metaslab_sync(metaslab_t *msp, uint64_t txg);
void metaslab_sync_done(metaslab_t *msp, uint64_t txg);
void metaslab_sync_reassess(metaslab_group_t *mg);
uint64_t metaslab_block_maxsize(metaslab_t *msp);

#define METASLAB_HINTBP_FAVOR   0x0
#define METASLAB_HINTBP_AVOID   0x1
#define METASLAB_GANG_HEADER    0x2
#define METASLAB_GANG_CHILD     0x4
#define METASLAB_GANG_AVOID     0x8

int metaslab_alloc(spa_t *spa, metaslab_class_t *mc, uint64_t psize,
extern int metaslab_alloc(spa_t *spa, metaslab_class_t *mc, uint64_t psize,
    blkptr_t *bp, int ncopies, uint64_t txg, blkptr_t *hintbp, int flags);
void metaslab_free(spa_t *spa, const blkptr_t *bp, uint64_t txg, boolean_t now);
int metaslab_claim(spa_t *spa, const blkptr_t *bp, uint64_t txg);
void metaslab_check_free(spa_t *spa, const blkptr_t *bp);
extern void metaslab_free(spa_t *spa, const blkptr_t *bp, uint64_t txg,
    boolean_t now);
extern int metaslab_claim(spa_t *spa, const blkptr_t *bp, uint64_t txg);
extern void metaslab_check_free(spa_t *spa, const blkptr_t *bp);

metaslab_class_t *metaslab_class_create(spa_t *spa, metaslab_ops_t *ops);
void metaslab_class_destroy(metaslab_class_t *mc);
int metaslab_class_validate(metaslab_class_t *mc);
extern metaslab_class_t *metaslab_class_create(spa_t *spa,
    space_map_ops_t *ops);
extern void metaslab_class_destroy(metaslab_class_t *mc);
extern int metaslab_class_validate(metaslab_class_t *mc);

void metaslab_class_space_update(metaslab_class_t *mc,
extern void metaslab_class_space_update(metaslab_class_t *mc,
    int64_t alloc_delta, int64_t defer_delta,
    int64_t space_delta, int64_t dspace_delta);
uint64_t metaslab_class_get_alloc(metaslab_class_t *mc);
uint64_t metaslab_class_get_space(metaslab_class_t *mc);
uint64_t metaslab_class_get_dspace(metaslab_class_t *mc);
uint64_t metaslab_class_get_deferred(metaslab_class_t *mc);
extern uint64_t metaslab_class_get_alloc(metaslab_class_t *mc);
extern uint64_t metaslab_class_get_space(metaslab_class_t *mc);
extern uint64_t metaslab_class_get_dspace(metaslab_class_t *mc);
extern uint64_t metaslab_class_get_deferred(metaslab_class_t *mc);

metaslab_group_t *metaslab_group_create(metaslab_class_t *mc, vdev_t *vd);
void metaslab_group_destroy(metaslab_group_t *mg);
void metaslab_group_activate(metaslab_group_t *mg);
void metaslab_group_passivate(metaslab_group_t *mg);
extern metaslab_group_t *metaslab_group_create(metaslab_class_t *mc,
    vdev_t *vd);
extern void metaslab_group_destroy(metaslab_group_t *mg);
extern void metaslab_group_activate(metaslab_group_t *mg);
extern void metaslab_group_passivate(metaslab_group_t *mg);

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_METASLAB_H */
```

```
**********************************************************
    6085 Tue Sep  3 20:27:07 2013
new/usr/src/uts/common/fs/zfs/sys/metaslab_impl.h
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _SYS_METASLAB_IMPL_H
#define _SYS_METASLAB_IMPL_H

#include <sys/metaslab.h>
#include <sys/space_map.h>
#include <sys/range_tree.h>
#include <sys/vdev.h>
#include <sys/txg.h>
#include <sys/avl.h>

#ifdef  __cplusplus
extern "C" {
#endif

struct metaslab_class {
        spa_t                   *mc_spa;
        metaslab_group_t        *mc_rotor;
        metaslab_ops_t          *mc_ops;
        space_map_ops_t         *mc_ops;
        uint64_t                mc_aliquot;
        uint64_t                mc_alloc_groups; /* # of allocatable groups */
        uint64_t                mc_alloc;       /* total allocated space */
        uint64_t                mc_deferred;    /* total deferred frees */
        uint64_t                mc_space;       /* total space (alloc + free) */
        uint64_t                mc_dspace;      /* total deflated space */
};

struct metaslab_group {
        kmutex_t                mg_lock;
        avl_tree_t              mg_metaslab_tree;
        uint64_t                mg_aliquot;
        uint64_t                mg_bonus_area;
        uint64_t                mg_alloc_failures;
        boolean_t               mg_allocatable;         /* can we allocate? */
        uint64_t                mg_free_capacity;       /* percentage free */
        int64_t                 mg_bias;
        int64_t                 mg_activation_count;
        metaslab_class_t        *mg_class;
        vdev_t                  *mg_vd;
        taskq_t                 *mg_taskq;
        metaslab_group_t        *mg_prev;
        metaslab_group_t        *mg_next;
};

/*
 * This value defines the number of elements in the ms_lbas array. The value
 * of 64 was chosen as it covers to cover all power of 2 buckets up to
 * UINT64_MAX. This is the equivalent of highbit(UINT64_MAX).
 */
#define MAX_LBAS        64

/*
 * Each metaslab maintains a set of in-core trees to track metaslab operations.
 * The in-core free tree (ms_tree) contains the current list of free segments.
 * As blocks are allocated, the allocated segment are removed from the ms_tree
 * and added to a per txg allocation tree (ms_alloctree). As blocks are freed,
 * they are added to the per txg free tree (ms_freetree). These per txg
 * trees allow us to process all allocations and frees in syncing context
 * where it is safe to update the on-disk space maps. One additional in-core
 * tree is maintained to track deferred frees (ms_defertree). Once a block
 * is freed it will move from the ms_freetree to the ms_defertree. A deferred
 * free means that a block has been freed but cannot be used by the pool
 * until TXG_DEFER_SIZE transactions groups later. For example, a block
 * that is freed in txg 50 will not be available for reallocation until
 * txg 52 (50 + TXG_DEFER_SIZE).  This provides a safety net for uberblock
 * rollback. A pool could be safely rolled back TXG_DEFERS_SIZE
 * transactions groups and ensure that no block has been reallocated.
 * Each metaslab maintains an in-core free map (ms_map) that contains the
 * current list of free segments. As blocks are allocated, the allocated
 * segment is removed from the ms_map and added to a per txg allocation map.
 * As blocks are freed, they are added to the per txg free map. These per
 * txg maps allow us to process all allocations and frees in syncing context
 * where it is safe to update the on-disk space maps.
 *
 * The simplified transition diagram looks like this:
 *
 *
 *      ALLOCATE
 *         |
 *         V
 *    free segment (ms_tree) --------> ms_alloctree ----> (write to space map)
 *         ^
 *         |
 *         |                              ms_freetree <--- FREE
 *         |                                   |
 *         |                                   |
 *         |                                   |
 *         +----------- ms_defertree <-------+---------> (write to space map)
 *
 *
```

```
 * Each metaslab's space is tracked in a single space map in the MOS,
 * Each metaslab's free space is tracked in a space map object in the MOS,
 * which is only updated in syncing context. Each time we sync a txg,
 * we append the allocs and frees from that txg to the space map.
 * The pool space is only updated once all metaslabs have finished syncing.
 * we append the allocs and frees from that txg to the space map object.
 * When the txg is done syncing, metaslab_sync_done() updates ms_smo
 * to ms_smo_syncing. Everything in ms_smo is always safe to allocate.
 *
 * To load the in-core free tree we read the space map from disk.
 * To load the in-core free map we read the space map object from disk.
 * This object contains a series of alloc and free records that are
 * combined to make up the list of all free segments in this metaslab. These
 * segments are represented in-core by the ms_tree and are stored in an
 * segments are represented in-core by the ms_map and are stored in an
 * AVL tree.
 *
 * As the space map grows (as a result of the appends) it will
 * eventually become space-inefficient. When the metaslab's in-core free tree
 * is zfs_condense_pct/100 times the size of the minimal on-disk
 * representation, we rewrite it in its minimized form. If a metaslab
 * needs to condense then we must set the ms_condensing flag to ensure
 * that allocations are not performed on the metaslab that is being written.
 * As the space map objects grows (as a result of the appends) it will
 * eventually become space-inefficient. When the space map object is
 * zfs_condense_pct/100 times the size of the minimal on-disk representation,
 * we rewrite it in its minimized form.
 */
struct metaslab {
        kmutex_t        ms_lock;
        kcondvar_t      ms_load_cv;
        space_map_t     *ms_sm;
        metaslab_ops_t  *ms_ops;
        uint64_t        ms_id;
        uint64_t        ms_start;
        uint64_t        ms_size;

        range_tree_t    *ms_alloctree[TXG_SIZE];
        range_tree_t    *ms_freetree[TXG_SIZE];
        range_tree_t    *ms_defertree[TXG_DEFER_SIZE];
        range_tree_t    *ms_tree;

        boolean_t       ms_condensing;  /* condensing? */
        boolean_t       ms_loaded;
        boolean_t       ms_loading;

        kmutex_t        ms_lock;        /* metaslab lock                */
        space_map_obj_t ms_smo;         /* synced space map object      */
        space_map_obj_t ms_smo_syncing; /* syncing space map object     */
        space_map_t     *ms_allocmap[TXG_SIZE]; /* allocated this txg   */
        space_map_t     *ms_freemap[TXG_SIZE];  /* freed this txg       */
        space_map_t     *ms_defermap[TXG_DEFER_SIZE];   /* deferred frees */
        space_map_t     *ms_map;        /* in-core free space map       */
        int64_t         ms_deferspace;  /* sum of ms_defermap[] space   */
        uint64_t        ms_weight;      /* weight vs. others in group   */
        uint64_t        ms_factor;
        uint64_t        ms_access_txg;

        /*
         * The metaslab block allocators can optionally use a size-ordered
         * range tree and/or an array of LBAs. Not all allocators use
         * this functionality. The ms_size_tree should always contain the
         * same number of segments as the ms_tree. The only difference
         * is that the ms_size_tree is ordered by segment sizes.
         */
        avl_tree_t      ms_size_tree;
```



```
        uint64_t        ms_lbas[MAX_LBAS];

        metaslab_group_t *ms_group;     /* metaslab group               */
        avl_node_t      ms_group_node;  /* node in metaslab group tree  */
        txg_node_t      ms_txg_node;    /* per-txg dirty metaslab links */
};
```

*unchanged_portion_omitted_*

```
**********************************************************
    3197 Tue Sep  3 20:27:08 2013
new/usr/src/uts/common/fs/zfs/sys/range_tree.h
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _SYS_RANGE_TREE_H
#define _SYS_RANGE_TREE_H

#include <sys/avl.h>
#include <sys/dmu.h>

#ifdef  __cplusplus
extern "C" {
#endif

#define RANGE_TREE_HISTOGRAM_SIZE       64

typedef struct range_tree_ops range_tree_ops_t;

typedef struct range_tree {
        avl_tree_t      rt_root;        /* offset-ordered segment AVL tree */
        uint64_t        rt_space;       /* sum of all segments in the map */
        range_tree_ops_t *rt_ops;
        void            *rt_arg;

        /*
         * The rt_histogram maintains a histogram of ranges. Each bucket,
         * rt_histogram[i], contains the number of ranges whose size is:
         * 2^i <= size of range in bytes < 2^(i+1)
```



```
         */
        uint64_t        rt_histogram[RANGE_TREE_HISTOGRAM_SIZE];
        kmutex_t        *rt_lock;       /* pointer to lock that protects map */
} range_tree_t;

typedef struct range_seg {
        avl_node_t      rs_node;        /* AVL node */
        avl_node_t      rs_pp_node;     /* AVL picker-private node */
        uint64_t        rs_start;       /* starting offset of this segment */
        uint64_t        rs_end;         /* ending offset (non-inclusive) */
} range_seg_t;

struct range_tree_ops {
        void    (*rtop_create)(range_tree_t *rt, void *arg);
        void    (*rtop_destroy)(range_tree_t *rt, void *arg);
        void    (*rtop_add)(range_tree_t *rt, range_seg_t *rs, void *arg);
        void    (*rtop_remove)(range_tree_t *rt, range_seg_t *rs, void *arg);
        void    (*rtop_vacate)(range_tree_t *rt, void *arg);
};

typedef void range_tree_func_t(void *arg, uint64_t start, uint64_t size);

void range_tree_init(void);
void range_tree_fini(void);
range_tree_t *range_tree_create(range_tree_ops_t *ops, void *arg, kmutex_t *lp);
void range_tree_destroy(range_tree_t *rt);
boolean_t range_tree_contains(range_tree_t *rt, uint64_t start, uint64_t size);
uint64_t range_tree_space(range_tree_t *rt);
void range_tree_verify(range_tree_t *rt, uint64_t start, uint64_t size);
void range_tree_swap(range_tree_t **rtsrc, range_tree_t **rtdst);
void range_tree_stat_verify(range_tree_t *rt);

void range_tree_add(void *arg, uint64_t start, uint64_t size);
void range_tree_remove(void *arg, uint64_t start, uint64_t size);

void range_tree_vacate(range_tree_t *rt, range_tree_func_t *func, void *arg);
void range_tree_walk(range_tree_t *rt, range_tree_func_t *func, void *arg);

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_RANGE_TREE_H */
```

```
**********************************************************
    6178 Tue Sep  3 20:27:09 2013
new/usr/src/uts/common/fs/zfs/sys/space_map.h
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

#ifndef _SYS_SPACE_MAP_H
#define _SYS_SPACE_MAP_H

#include <sys/avl.h>
#include <sys/range_tree.h>
#include <sys/dmu.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * The size of the space map object has increased to include a histogram.
 * The SPACE_MAP_SIZE_V0 designates the original size and is used to
 * maintain backward compatibility.
 */
#define SPACE_MAP_SIZE_V0       (3 * sizeof (uint64_t))
#define SPACE_MAP_HISTOGRAM_SIZE(sm)                    \
        (sizeof ((sm)->sm_phys->smp_histogram) /        \
        sizeof ((sm)->sm_phys->smp_histogram[0]))
typedef struct space_map_ops space_map_ops_t;

/*
```



```
 * The space_map_phys is the on-disk representation of the space map.
 * Consumers of space maps should never reference any of the members of this
 * structure directly. These members may only be updated in syncing context.
 *
 * Note the smp_object is no longer used but remains in the structure
 * for backward compatibility.
 */
typedef struct space_map_phys {
        uint64_t        smp_object;     /* on-disk space map object */
        uint64_t        smp_objsize;    /* size of the object */
        uint64_t        smp_alloc;      /* space allocated from the map */
        uint64_t        smp_pad[5];     /* reserved */

        /*
         * The smp_histogram maintains a histogram of free regions. Each
         * bucket, smp_histogram[i], contains the number of free regions
         * whose size is:
         * 2^(i+sm_shift) <= size of free region in bytes < 2^(i+sm_shift+1)
         */
        uint64_t        smp_histogram[32]; /* histogram of free space */
} space_map_phys_t;

/*
 * The space map object defines a region of space, its size, how much is
 * allocated, and the on-disk object that stores this information.
 * Consumers of space maps may only access the members of this structure.
 */
typedef struct space_map {
        avl_tree_t      sm_root;        /* offset-ordered segment AVL tree */
        uint64_t        sm_space;       /* sum of all segments in the map */
        uint64_t        sm_start;       /* start of map */
        uint64_t        sm_size;        /* size of map */
        uint8_t         sm_shift;       /* unit shift */
        uint64_t        sm_length;      /* synced length */
        uint64_t        sm_alloc;       /* synced space allocated */
        objset_t        *sm_os;         /* objset for this map */
        uint64_t        sm_object;      /* object id for this map */
        uint32_t        sm_blksz;       /* block size for space map */
        dmu_buf_t       *sm_dbuf;       /* space_map_phys_t dbuf */
        space_map_phys_t *sm_phys;      /* on-disk space map */
        uint8_t         sm_loaded;      /* map loaded? */
        uint8_t         sm_loading;     /* map loading? */
        uint8_t         sm_condensing;  /* map condensing? */
        kcondvar_t      sm_load_cv;     /* map load completion */
        space_map_ops_t *sm_ops;        /* space map block picker ops vector */
        avl_tree_t      *sm_pp_root;    /* size-ordered, picker-private tree */
        void            *sm_ppd;        /* picker-private data */
        kmutex_t        *sm_lock;       /* pointer to lock that protects map */
} space_map_t;

typedef struct space_seg {
        avl_node_t      ss_node;        /* AVL node */
        avl_node_t      ss_pp_node;     /* AVL picker-private node */
        uint64_t        ss_start;       /* starting offset of this segment */
        uint64_t        ss_end;         /* ending offset (non-inclusive) */
} space_seg_t;

typedef struct space_ref {
        avl_node_t      sr_node;        /* AVL node */
        uint64_t        sr_offset;      /* offset (start or end) */
        int64_t         sr_refcnt;      /* associated reference count */
} space_ref_t;

typedef struct space_map_obj {
        uint64_t        smo_object;     /* on-disk space map object */
        uint64_t        smo_objsize;    /* size of the object */
```

```
        uint64_t        smo_alloc;      /* space allocated from the map */
} space_map_obj_t;

struct space_map_ops {
        void    (*smop_load)(space_map_t *sm);
        void    (*smop_unload)(space_map_t *sm);
        uint64_t (*smop_alloc)(space_map_t *sm, uint64_t size);
        void    (*smop_claim)(space_map_t *sm, uint64_t start, uint64_t size);
        void    (*smop_free)(space_map_t *sm, uint64_t start, uint64_t size);
        uint64_t (*smop_max)(space_map_t *sm);
        boolean_t (*smop_fragmented)(space_map_t *sm);
};

/*
 * debug entry
 *
 *    1      3         10                   50
 *  ,---+--------+------------+---------------------------------.
 *  | 1 | action |  syncpass  |        txg (lower bits)         |
 *  '---+--------+------------+---------------------------------'
 *   63  62    60 59        50 49                                0
 *
 *
 * non-debug entry
 *
 *    1               47                   1           15
 *  ,-----------------------------------------------------------.
 *  | 0 |   offset (sm_shift units)    | type |       run       |
 *  '-----------------------------------------------------------'
 *   63  62                          17   16   15               0
 */

/* All this stuff takes and returns bytes */
#define SM_RUN_DECODE(x)        (BF64_DECODE(x, 0, 15) + 1)
#define SM_RUN_ENCODE(x)        BF64_ENCODE((x) - 1, 0, 15)
#define SM_TYPE_DECODE(x)       BF64_DECODE(x, 15, 1)
#define SM_TYPE_ENCODE(x)       BF64_ENCODE(x, 15, 1)
#define SM_OFFSET_DECODE(x)     BF64_DECODE(x, 16, 47)
#define SM_OFFSET_ENCODE(x)     BF64_ENCODE(x, 16, 47)
#define SM_DEBUG_DECODE(x)      BF64_DECODE(x, 63, 1)
#define SM_DEBUG_ENCODE(x)      BF64_ENCODE(x, 63, 1)

#define SM_DEBUG_ACTION_DECODE(x)       BF64_DECODE(x, 60, 3)
#define SM_DEBUG_ACTION_ENCODE(x)       BF64_ENCODE(x, 60, 3)

#define SM_DEBUG_SYNCPASS_DECODE(x)     BF64_DECODE(x, 50, 10)
#define SM_DEBUG_SYNCPASS_ENCODE(x)     BF64_ENCODE(x, 50, 10)

#define SM_DEBUG_TXG_DECODE(x)          BF64_DECODE(x, 0, 50)
#define SM_DEBUG_TXG_ENCODE(x)          BF64_ENCODE(x, 0, 50)

#define SM_RUN_MAX                      SM_RUN_DECODE(~0ULL)

typedef enum {
        SM_ALLOC,
        SM_FREE
} maptype_t;
#define SM_ALLOC        0x0
#define SM_FREE         0x1

/*
 * The data for a given space map can be kept on blocks of any size.
 * Larger blocks entail fewer i/o operations, but they also cause the
 * DMU to keep more data in-core, and also to waste more i/o bandwidth
 * when only a few blocks have changed since the last transaction group.
 * Rather than having a fixed block size for all space maps the block size
```



```
 * can adjust as needed (see space_map_max_blksz). Set the initial block
 * size for the space map to 4k.
 * This could use a lot more research, but for now, set the freelist
 * block size to 4k (2^12).
 */
#define SPACE_MAP_INITIAL_BLOCKSIZE     (1ULL << 12)
#define SPACE_MAP_BLOCKSHIFT    12

int space_map_load(space_map_t *sm, range_tree_t *rt, maptype_t maptype);
typedef void space_map_func_t(space_map_t *sm, uint64_t start, uint64_t size);

void space_map_histogram_clear(space_map_t *sm);
void space_map_histogram_add(space_map_t *sm, range_tree_t *rt,
    dmu_tx_t *tx);
extern void space_map_init(void);
extern void space_map_fini(void);
extern void space_map_create(space_map_t *sm, uint64_t start, uint64_t size,
    uint8_t shift, kmutex_t *lp);
extern void space_map_destroy(space_map_t *sm);
extern void space_map_add(space_map_t *sm, uint64_t start, uint64_t size);
extern void space_map_remove(space_map_t *sm, uint64_t start, uint64_t size);
extern boolean_t space_map_contains(space_map_t *sm,
    uint64_t start, uint64_t size);
extern space_seg_t *space_map_find(space_map_t *sm, uint64_t start,
    uint64_t size, avl_index_t *wherep);
extern void space_map_swap(space_map_t **msrc, space_map_t **mdest);
extern void space_map_vacate(space_map_t *sm,
    space_map_func_t *func, space_map_t *mdest);
extern void space_map_walk(space_map_t *sm,
    space_map_func_t *func, space_map_t *mdest);

void space_map_update(space_map_t *sm);
extern void space_map_load_wait(space_map_t *sm);
extern int space_map_load(space_map_t *sm, space_map_ops_t *ops,
    uint8_t maptype, space_map_obj_t *smo, objset_t *os);
extern void space_map_unload(space_map_t *sm);

uint64_t space_map_object(space_map_t *sm);
uint64_t space_map_allocated(space_map_t *sm);
uint64_t space_map_length(space_map_t *sm);
extern uint64_t space_map_alloc(space_map_t *sm, uint64_t size);
extern void space_map_claim(space_map_t *sm, uint64_t start, uint64_t size);
extern void space_map_free(space_map_t *sm, uint64_t start, uint64_t size);
extern uint64_t space_map_maxsize(space_map_t *sm);

void space_map_write(space_map_t *sm, range_tree_t *rt, maptype_t maptype,
    dmu_tx_t *tx);
void space_map_truncate(space_map_t *sm, dmu_tx_t *tx);
uint64_t space_map_alloc(objset_t *os, dmu_tx_t *tx);
void space_map_free(space_map_t *sm, dmu_tx_t *tx);
extern void space_map_sync(space_map_t *sm, uint8_t maptype,
    space_map_obj_t *smo, objset_t *os, dmu_tx_t *tx);
extern void space_map_truncate(space_map_obj_t *smo,
    objset_t *os, dmu_tx_t *tx);

int space_map_open(space_map_t **smp, objset_t *os, uint64_t object,
    uint64_t start, uint64_t size, uint8_t shift, kmutex_t *lp);
void space_map_close(space_map_t *sm);
extern void space_map_ref_create(avl_tree_t *t);
extern void space_map_ref_destroy(avl_tree_t *t);
extern void space_map_ref_add_seg(avl_tree_t *t,
    uint64_t start, uint64_t end, int64_t refcnt);
extern void space_map_ref_add_map(avl_tree_t *t,
    space_map_t *sm, int64_t refcnt);
extern void space_map_ref_generate_map(avl_tree_t *t,
    space_map_t *sm, int64_t minref);
```

```
171 int64_t space_map_alloc_delta(space_map_t *sm);

173 #ifdef  __cplusplus
174 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1711 Tue Sep  3 20:27:10 2013
new/usr/src/uts/common/fs/zfs/sys/space_reftree.h
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _SYS_SPACE_REFTREE_H
#define _SYS_SPACE_REFTREE_H

#include <sys/range_tree.h>

#ifdef  __cplusplus
extern "C" {
#endif

typedef struct space_ref {
        avl_node_t      sr_node;        /* AVL node */
        uint64_t        sr_offset;      /* range offset (start or end) */
        int64_t         sr_refcnt;      /* associated reference count */
} space_ref_t;

void space_reftree_create(avl_tree_t *t);
void space_reftree_destroy(avl_tree_t *t);
void space_reftree_add_seg(avl_tree_t *t, uint64_t start, uint64_t end,
    int64_t refcnt);
void space_reftree_add_map(avl_tree_t *t, range_tree_t *rt, int64_t refcnt);
void space_reftree_generate_map(avl_tree_t *t, range_tree_t *rt,
    int64_t minref);

#ifdef  __cplusplus
```



```
}
#endif

#endif  /* _SYS_SPACE_REFTREE_H */
```

```
**********************************************************
   11807 Tue Sep  3 20:27:10 2013
new/usr/src/uts/common/fs/zfs/sys/vdev_impl.h
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 121 /*
 122  * Virtual device descriptor
 123  */
 124 struct vdev {
 125         /*
 126          * Common to all vdev types.
 127          */
 128         uint64_t        vdev_id;        /* child number in vdev parent  */
 129         uint64_t        vdev_guid;      /* unique ID for this vdev      */
 130         uint64_t        vdev_guid_sum;  /* self guid + all child guids  */
 131         uint64_t        vdev_orig_guid; /* orig. guid prior to remove   */
 132         uint64_t        vdev_asize;     /* allocatable device capacity  */
 133         uint64_t        vdev_min_asize; /* min acceptable asize         */
 134         uint64_t        vdev_max_asize; /* max acceptable asize         */
 135         uint64_t        vdev_ashift;    /* block alignment shift        */
 136         uint64_t        vdev_state;     /* see VDEV_STATE_* #defines    */
 137         uint64_t        vdev_prevstate; /* used when reopening a vdev   */
 138         vdev_ops_t      *vdev_ops;      /* vdev operations              */
 139         spa_t           *vdev_spa;      /* spa for this vdev            */
 140         void            *vdev_tsd;      /* type-specific data           */
 141         vnode_t         *vdev_name_vp;  /* vnode for pathname           */
 142         vnode_t         *vdev_devid_vp; /* vnode for devid              */
 143         vdev_t          *vdev_top;      /* top-level vdev               */
 144         vdev_t          *vdev_parent;   /* parent vdev                  */
 145         vdev_t          **vdev_child;   /* array of children            */
 146         uint64_t        vdev_children;  /* number of children           */
 147         space_map_t     vdev_dtl[DTL_TYPES]; /* in-core dirty time logs */
 147         vdev_stat_t     vdev_stat;      /* virtual device statistics    */
 148         boolean_t       vdev_expanding; /* expand the vdev?             */
 149         boolean_t       vdev_reopening; /* reopen in progress?          */
 150         int             vdev_open_error; /* error on last open          */
 151         kthread_t       *vdev_open_thread; /* thread opening children   */
 152         uint64_t        vdev_crtxg;     /* txg when top-level was added */

 154         /*
 155          * Top-level vdev state.
 156          */
 157         uint64_t        vdev_ms_array;  /* metaslab array object        */
 158         uint64_t        vdev_ms_shift;  /* metaslab size shift          */
 159         uint64_t        vdev_ms_count;  /* number of metaslabs          */
 160         metaslab_group_t *vdev_mg;      /* metaslab group               */
 161         metaslab_t      **vdev_ms;      /* metaslab array               */
 162         txg_list_t      vdev_ms_list;   /* per-txg dirty metaslab lists */
 163         txg_list_t      vdev_dtl_list;  /* per-txg dirty DTL lists      */
 164         txg_node_t      vdev_txg_node;  /* per-txg dirty vdev linkage   */
 165         boolean_t       vdev_remove_wanted; /* async remove wanted?     */
 166         boolean_t       vdev_probe_wanted; /* async probe wanted?       */
 168         uint64_t        vdev_removing;  /* device is being removed?     */
 167         list_node_t     vdev_config_dirty_node; /* config dirty list    */
 168         list_node_t     vdev_state_dirty_node; /* state dirty list      */
 169         uint64_t        vdev_deflate_ratio; /* deflation ratio (x512)   */
```



```
 170         uint64_t        vdev_islog;     /* is an intent log device      */
 171         uint64_t        vdev_removing;  /* device is being removed?     */
 172         boolean_t       vdev_ishole;    /* is a hole in the namespace   */
 173         uint64_t        vdev_ishole;    /* is a hole in the namespace   */

 174         /*
 175          * Leaf vdev state.
 176          */
 177         range_tree_t    *vdev_dtl[DTL_TYPES]; /* dirty time logs        */
 178         space_map_t     *vdev_dtl_sm;   /* dirty time log space map     */
 179         txg_node_t      vdev_dtl_node;  /* per-txg dirty DTL linkage    */
 180         uint64_t        vdev_dtl_object; /* DTL object                  */
 181         uint64_t        vdev_psize;     /* physical device capacity     */
 179         space_map_obj_t vdev_dtl_smo;   /* dirty time log space map obj */
 180         txg_node_t      vdev_dtl_node;  /* per-txg dirty DTL linkage    */
 182         uint64_t        vdev_wholedisk; /* true if this is a whole disk */
 183         uint64_t        vdev_offline;   /* persistent offline state     */
 184         uint64_t        vdev_faulted;   /* persistent faulted state     */
 185         uint64_t        vdev_degraded;  /* persistent degraded state    */
 186         uint64_t        vdev_removed;   /* persistent removed state     */
 187         uint64_t        vdev_resilver_txg; /* persistent resilvering state */
 188         uint64_t        vdev_nparity;   /* number of parity devices for raidz */
 189         char            *vdev_path;     /* vdev path (if any)           */
 190         char            *vdev_devid;    /* vdev devid (if any)          */
 191         char            *vdev_physpath; /* vdev device path (if any)    */
 192         char            *vdev_fru;      /* physical FRU location        */
 193         uint64_t        vdev_not_present; /* not present during import  */
 194         uint64_t        vdev_unspare;   /* unspare when resilvering done */
 194         hrtime_t        vdev_last_try;  /* last reopen time             */
 195         boolean_t       vdev_nowritecache; /* true if flushwritecache failed */
 196         boolean_t       vdev_checkremove; /* temporary online test      */
 197         boolean_t       vdev_forcefault; /* force online fault          */
 198         boolean_t       vdev_splitting; /* split or repair in progress  */
 199         boolean_t       vdev_delayed_close; /* delayed device close?    */
 200         boolean_t       vdev_tmpoffline; /* device taken offline temporarily? */
 201         boolean_t       vdev_detached;  /* device detached?             */
 202         boolean_t       vdev_cant_read; /* vdev is failing all reads    */
 203         boolean_t       vdev_cant_write; /* vdev is failing all writes  */
 204         boolean_t       vdev_isspare;   /* was a hot spare              */
 205         boolean_t       vdev_isl2cache; /* was a l2cache device         */
 200         uint8_t         vdev_tmpoffline; /* device taken offline temporarily? */
 201         uint8_t         vdev_detached;  /* device detached?             */
 202         uint8_t         vdev_cant_read; /* vdev is failing all reads    */
 203         uint8_t         vdev_cant_write; /* vdev is failing all writes  */
 204         uint64_t        vdev_isspare;   /* was a hot spare              */
 205         uint64_t        vdev_isl2cache; /* was a l2cache device         */
 206         vdev_queue_t    vdev_queue;     /* I/O deadline schedule queue  */
 207         vdev_cache_t    vdev_cache;     /* physical block cache         */
 208         spa_aux_vdev_t  *vdev_aux;      /* for l2cache vdevs            */
 209         zio_t           *vdev_probe_zio; /* root of current probe       */
 210         vdev_aux_t      vdev_label_aux; /* on-disk aux state            */

 212         /*
 213          * For DTrace to work in userland (libzpool) context, these fields must
 214          * remain at the end of the structure.  DTrace will use the kernel's
 215          * CTF definition for 'struct vdev', and since the size of a kmutex_t is
 216          * larger in userland, the offsets for the rest of the fields would be
 217          * incorrect.
 218          */
 219         kmutex_t        vdev_dtl_lock;  /* vdev_dtl_{map,resilver}      */
 220         kmutex_t        vdev_stat_lock; /* vdev_stat                    */
 221         kmutex_t        vdev_probe_lock; /* protects vdev_probe_zio     */
 222 };
______unchanged_portion_omitted_

 252 /*
```

```
 * vdev_dirty() flags
 */
#define VDD_METASLAB    0x01
#define VDD_DTL         0x02

/* Offset of embedded boot loader region on each label */
#define VDEV_BOOT_OFFSET        (2 * sizeof (vdev_label_t))
/*
 * Size of embedded boot loader region on each label.
 * The total size of the first two labels plus the boot area is 4MB.
 */
#define VDEV_BOOT_SIZE          (7ULL << 19)                    /* 3.5M */

/*
 * Size of label regions at the start and end of each leaf device.
 */
#define VDEV_LABEL_START_SIZE   (2 * sizeof (vdev_label_t) + VDEV_BOOT_SIZE)
#define VDEV_LABEL_END_SIZE     (2 * sizeof (vdev_label_t))
#define VDEV_LABELS             4
#define VDEV_BEST_LABEL         VDEV_LABELS

#define VDEV_ALLOC_LOAD         0
#define VDEV_ALLOC_ADD          1
#define VDEV_ALLOC_SPARE        2
#define VDEV_ALLOC_L2CACHE      3
#define VDEV_ALLOC_ROOTPOOL     4
#define VDEV_ALLOC_SPLIT        5
#define VDEV_ALLOC_ATTACH       6

/*
 * Allocate or free a vdev
 */
extern vdev_t *vdev_alloc_common(spa_t *spa, uint_t id, uint64_t guid,
    vdev_ops_t *ops);
extern int vdev_alloc(spa_t *spa, vdev_t **vdp, nvlist_t *config,
    vdev_t *parent, uint_t id, int alloctype);
extern void vdev_free(vdev_t *vd);

/*
 * Add or remove children and parents
 */
extern void vdev_add_child(vdev_t *pvd, vdev_t *cvd);
extern void vdev_remove_child(vdev_t *pvd, vdev_t *cvd);
extern void vdev_compact_children(vdev_t *pvd);
extern vdev_t *vdev_add_parent(vdev_t *cvd, vdev_ops_t *ops);
extern void vdev_remove_parent(vdev_t *cvd);

/*
 * vdev sync load and sync
 */
extern void vdev_load_log_state(vdev_t *nvd, vdev_t *ovd);
extern boolean_t vdev_log_state_valid(vdev_t *vd);
extern void vdev_load(vdev_t *vd);
extern int vdev_dtl_load(vdev_t *vd);
extern void vdev_sync(vdev_t *vd, uint64_t txg);
extern void vdev_sync_done(vdev_t *vd, uint64_t txg);
extern void vdev_dirty(vdev_t *vd, int flags, void *arg, uint64_t txg);
extern void vdev_dirty_leaves(vdev_t *vd, int flags, uint64_t txg);

/*
 * Available vdev types.
 */
extern vdev_ops_t vdev_root_ops;
extern vdev_ops_t vdev_mirror_ops;
extern vdev_ops_t vdev_replacing_ops;
extern vdev_ops_t vdev_raidz_ops;
```



```
extern vdev_ops_t vdev_disk_ops;
extern vdev_ops_t vdev_file_ops;
extern vdev_ops_t vdev_missing_ops;
extern vdev_ops_t vdev_hole_ops;
extern vdev_ops_t vdev_spare_ops;

/*
 * Common size functions
 */
extern uint64_t vdev_default_asize(vdev_t *vd, uint64_t psize);
extern uint64_t vdev_get_min_asize(vdev_t *vd);
extern void vdev_set_min_asize(vdev_t *vd);

/*
 * Global variables
 */
/* zdb uses this tunable, so it must be declared here to make lint happy. */
extern int zfs_vdev_cache_size;

/*
 * The vdev_buf_t is used to translate between zio_t and buf_t, and back again.
 */
typedef struct vdev_buf {
        buf_t   vb_buf;         /* buffer that describes the io */
        zio_t   *vb_io;         /* pointer back to the original zio_t */
} vdev_buf_t;
```

*unchanged_portion_omitted_*

****
1809 Tue Sep 3 20:27:12 2013
new/usr/src/uts/common/fs/zfs/sys/zfeature.h
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.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) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

#ifndef _SYS_ZFEATURE_H
#define _SYS_ZFEATURE_H

#include <sys/nvpair.h>
#include "zfeature_common.h"

#ifdef  __cplusplus
extern "C" {
#endif

struct spa;
struct dmu_tx;
struct objset;

extern boolean_t feature_is_supported(struct objset *os, uint64_t obj,
    uint64_t desc_obj, nvlist_t *unsup_feat, nvlist_t *enabled_feat);

extern void spa_feature_create_zap_objects(struct spa *, struct dmu_tx *);
extern void spa_feature_enable(struct spa *, zfeature_info_t *,
    struct dmu_tx *);
extern void spa_feature_incr(struct spa *, zfeature_info_t *, struct dmu_tx *);
extern void spa_feature_decr(struct spa *, zfeature_info_t *, struct dmu_tx *);
extern boolean_t spa_feature_is_enabled(struct spa *, zfeature_info_t *);
extern boolean_t spa_feature_is_active(struct spa *, zfeature_info_t *);
extern int spa_feature_get_refcount(struct spa *, zfeature_info_t *);

#ifdef  __cplusplus
```



```
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   89088 Tue Sep  3 20:27:13 2013
new/usr/src/uts/common/fs/zfs/vdev.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  * Copyright (c) 2013 by Delphix. All rights reserved.
  26  */

  28 #include <sys/zfs_context.h>
  29 #include <sys/fm/fs/zfs.h>
  30 #include <sys/spa.h>
  31 #include <sys/spa_impl.h>
  32 #include <sys/dmu.h>
  33 #include <sys/dmu_tx.h>
  34 #include <sys/vdev_impl.h>
  35 #include <sys/uberblock_impl.h>
  36 #include <sys/metaslab.h>
  37 #include <sys/metaslab_impl.h>
  38 #include <sys/space_map.h>
  39 #include <sys/space_reftree.h>
  40 #include <sys/zio.h>
  41 #include <sys/zap.h>
  42 #include <sys/fs/zfs.h>
  43 #include <sys/arc.h>
  44 #include <sys/zil.h>
  45 #include <sys/dsl_scan.h>

  47 /*
  48  * Virtual device management.
  49  */

  51 static vdev_ops_t *vdev_ops_table[] = {
  52         &vdev_root_ops,
  53         &vdev_raidz_ops,
```



```
  54         &vdev_mirror_ops,
  55         &vdev_replacing_ops,
  56         &vdev_spare_ops,
  57         &vdev_disk_ops,
  58         &vdev_file_ops,
  59         &vdev_missing_ops,
  60         &vdev_hole_ops,
  61         NULL
  62 };
______unchanged_portion_omitted_

 279 /*
 280  * Allocate and minimally initialize a vdev_t.
 281  */
 282 vdev_t *
 283 vdev_alloc_common(spa_t *spa, uint_t id, uint64_t guid, vdev_ops_t *ops)
 284 {
 285         vdev_t *vd;

 287         vd = kmem_zalloc(sizeof (vdev_t), KM_SLEEP);

 289         if (spa->spa_root_vdev == NULL) {
 290                 ASSERT(ops == &vdev_root_ops);
 291                 spa->spa_root_vdev = vd;
 292                 spa->spa_load_guid = spa_generate_guid(NULL);
 293         }

 295         if (guid == 0 && ops != &vdev_hole_ops) {
 296                 if (spa->spa_root_vdev == vd) {
 297                         /*
 298                          * The root vdev's guid will also be the pool guid,
 299                          * which must be unique among all pools.
 300                          */
 301                         guid = spa_generate_guid(NULL);
 302                 } else {
 303                         /*
 304                          * Any other vdev's guid must be unique within the pool.
 305                          */
 306                         guid = spa_generate_guid(spa);
 307                 }
 308                 ASSERT(!spa_guid_exists(spa_guid(spa), guid));
 309         }

 311         vd->vdev_spa = spa;
 312         vd->vdev_id = id;
 313         vd->vdev_guid = guid;
 314         vd->vdev_guid_sum = guid;
 315         vd->vdev_ops = ops;
 316         vd->vdev_state = VDEV_STATE_CLOSED;
 317         vd->vdev_ishole = (ops == &vdev_hole_ops);

 319         mutex_init(&vd->vdev_dtl_lock, NULL, MUTEX_DEFAULT, NULL);
 320         mutex_init(&vd->vdev_stat_lock, NULL, MUTEX_DEFAULT, NULL);
 321         mutex_init(&vd->vdev_probe_lock, NULL, MUTEX_DEFAULT, NULL);
 322         for (int t = 0; t < DTL_TYPES; t++) {
 323                 vd->vdev_dtl[t] = range_tree_create(NULL, NULL,
 322                 space_map_create(&vd->vdev_dtl[t], 0, -1ULL, 0,
 324                     &vd->vdev_dtl_lock);
 325         }
 326         txg_list_create(&vd->vdev_ms_list,
 327             offsetof(struct metaslab, ms_txg_node));
 328         txg_list_create(&vd->vdev_dtl_list,
 329             offsetof(struct vdev, vdev_dtl_node));
 330         vd->vdev_stat.vs_timestamp = gethrtime();
 331         vdev_queue_init(vd);
 332         vdev_cache_init(vd);
```

```
334         return (vd);
335 }

337 /*
338  * Allocate a new vdev.  The 'alloctype' is used to control whether we are
339  * creating a new vdev or loading an existing one - the behavior is slightly
340  * different for each case.
341  */
342 int
343 vdev_alloc(spa_t *spa, vdev_t **vdp, nvlist_t *nv, vdev_t *parent, uint_t id,
344     int alloctype)
345 {
346         vdev_ops_t *ops;
347         char *type;
348         uint64_t guid = 0, islog, nparity;
349         vdev_t *vd;

351         ASSERT(spa_config_held(spa, SCL_ALL, RW_WRITER) == SCL_ALL);

353         if (nvlist_lookup_string(nv, ZPOOL_CONFIG_TYPE, &type) != 0)
354                 return (SET_ERROR(EINVAL));

356         if ((ops = vdev_getops(type)) == NULL)
357                 return (SET_ERROR(EINVAL));

359         /*
360          * If this is a load, get the vdev guid from the nvlist.
361          * Otherwise, vdev_alloc_common() will generate one for us.
362          */
363         if (alloctype == VDEV_ALLOC_LOAD) {
364                 uint64_t label_id;

366                 if (nvlist_lookup_uint64(nv, ZPOOL_CONFIG_ID, &label_id) ||
367                     label_id != id)
368                         return (SET_ERROR(EINVAL));

370                 if (nvlist_lookup_uint64(nv, ZPOOL_CONFIG_GUID, &guid) != 0)
371                         return (SET_ERROR(EINVAL));
372         } else if (alloctype == VDEV_ALLOC_SPARE) {
373                 if (nvlist_lookup_uint64(nv, ZPOOL_CONFIG_GUID, &guid) != 0)
374                         return (SET_ERROR(EINVAL));
375         } else if (alloctype == VDEV_ALLOC_L2CACHE) {
376                 if (nvlist_lookup_uint64(nv, ZPOOL_CONFIG_GUID, &guid) != 0)
377                         return (SET_ERROR(EINVAL));
378         } else if (alloctype == VDEV_ALLOC_ROOTPOOL) {
379                 if (nvlist_lookup_uint64(nv, ZPOOL_CONFIG_GUID, &guid) != 0)
380                         return (SET_ERROR(EINVAL));
381         }

383         /*
384          * The first allocated vdev must be of type 'root'.
385          */
386         if (ops != &vdev_root_ops && spa->spa_root_vdev == NULL)
387                 return (SET_ERROR(EINVAL));

389         /*
390          * Determine whether we're a log vdev.
391          */
392         islog = 0;
393         (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_IS_LOG, &islog);
394         if (islog && spa_version(spa) < SPA_VERSION_SLOGS)
395                 return (SET_ERROR(ENOTSUP));

397         if (ops == &vdev_hole_ops && spa_version(spa) < SPA_VERSION_HOLES)
398                 return (SET_ERROR(ENOTSUP));
```



```
400         /*
401          * Set the nparity property for RAID-Z vdevs.
402          */
403         nparity = -1ULL;
404         if (ops == &vdev_raidz_ops) {
405                 if (nvlist_lookup_uint64(nv, ZPOOL_CONFIG_NPARITY,
406                     &nparity) == 0) {
407                         if (nparity == 0 || nparity > VDEV_RAIDZ_MAXPARITY)
408                                 return (SET_ERROR(EINVAL));
409                         /*
410                          * Previous versions could only support 1 or 2 parity
411                          * device.
412                          */
413                         if (nparity > 1 &&
414                             spa_version(spa) < SPA_VERSION_RAIDZ2)
415                                 return (SET_ERROR(ENOTSUP));
416                         if (nparity > 2 &&
417                             spa_version(spa) < SPA_VERSION_RAIDZ3)
418                                 return (SET_ERROR(ENOTSUP));
419                 } else {
420                         /*
421                          * We require the parity to be specified for SPAs that
422                          * support multiple parity levels.
423                          */
424                         if (spa_version(spa) >= SPA_VERSION_RAIDZ2)
425                                 return (SET_ERROR(EINVAL));
426                         /*
427                          * Otherwise, we default to 1 parity device for RAID-Z.
428                          */
429                         nparity = 1;
430                 }
431         } else {
432                 nparity = 0;
433         }
434         ASSERT(nparity != -1ULL);

436         vd = vdev_alloc_common(spa, id, guid, ops);

438         vd->vdev_islog = islog;
439         vd->vdev_nparity = nparity;

441         if (nvlist_lookup_string(nv, ZPOOL_CONFIG_PATH, &vd->vdev_path) == 0)
442                 vd->vdev_path = spa_strdup(vd->vdev_path);
443         if (nvlist_lookup_string(nv, ZPOOL_CONFIG_DEVID, &vd->vdev_devid) == 0)
444                 vd->vdev_devid = spa_strdup(vd->vdev_devid);
445         if (nvlist_lookup_string(nv, ZPOOL_CONFIG_PHYS_PATH,
446             &vd->vdev_physpath) == 0)
447                 vd->vdev_physpath = spa_strdup(vd->vdev_physpath);
448         if (nvlist_lookup_string(nv, ZPOOL_CONFIG_FRU, &vd->vdev_fru) == 0)
449                 vd->vdev_fru = spa_strdup(vd->vdev_fru);

451         /*
452          * Set the whole_disk property.  If it's not specified, leave the value
453          * as -1.
454          */
455         if (nvlist_lookup_uint64(nv, ZPOOL_CONFIG_WHOLE_DISK,
456             &vd->vdev_wholedisk) != 0)
457                 vd->vdev_wholedisk = -1ULL;

459         /*
460          * Look for the 'not present' flag.  This will only be set if the device
461          * was not present at the time of import.
462          */
463         (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_NOT_PRESENT,
464             &vd->vdev_not_present);
```

```
466         /*
467          * Get the alignment requirement.
468          */
469         (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_ASHIFT, &vd->vdev_ashift);

471         /*
472          * Retrieve the vdev creation time.
473          */
474         (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_CREATE_TXG,
475             &vd->vdev_crtxg);

477         /*
478          * If we're a top-level vdev, try to load the allocation parameters.
479          */
480         if (parent && !parent->vdev_parent &&
481             (alloctype == VDEV_ALLOC_LOAD || alloctype == VDEV_ALLOC_SPLIT)) {
482                 (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_METASLAB_ARRAY,
483                     &vd->vdev_ms_array);
484                 (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_METASLAB_SHIFT,
485                     &vd->vdev_ms_shift);
486                 (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_ASIZE,
487                     &vd->vdev_asize);
488                 (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_REMOVING,
489                     &vd->vdev_removing);
490         }

492         if (parent && !parent->vdev_parent && alloctype != VDEV_ALLOC_ATTACH) {
493                 ASSERT(alloctype == VDEV_ALLOC_LOAD ||
494                     alloctype == VDEV_ALLOC_ADD ||
495                     alloctype == VDEV_ALLOC_SPLIT ||
496                     alloctype == VDEV_ALLOC_ROOTPOOL);
497                 vd->vdev_mg = metaslab_group_create(islog ?
498                     spa_log_class(spa) : spa_normal_class(spa), vd);
499         }

501         /*
502          * If we're a leaf vdev, try to load the DTL object and other state.
503          */
504         if (vd->vdev_ops->vdev_op_leaf &&
505             (alloctype == VDEV_ALLOC_LOAD || alloctype == VDEV_ALLOC_L2CACHE ||
506             alloctype == VDEV_ALLOC_ROOTPOOL)) {
507                 if (alloctype == VDEV_ALLOC_LOAD) {
508                         (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_DTL,
509                             &vd->vdev_dtl_object);
508                             &vd->vdev_dtl_smo.smo_object);
510                         (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_UNSPARE,
511                             &vd->vdev_unspare);
512                 }

514                 if (alloctype == VDEV_ALLOC_ROOTPOOL) {
515                         uint64_t spare = 0;

517                         if (nvlist_lookup_uint64(nv, ZPOOL_CONFIG_IS_SPARE,
518                             &spare) == 0 && spare)
519                                 spa_spare_add(vd);
520                 }

522                 (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_OFFLINE,
523                     &vd->vdev_offline);

525                 (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_RESILVER_TXG,
526                     &vd->vdev_resilver_txg);

528                 /*
529                  * When importing a pool, we want to ignore the persistent fault
```



```
530                  * state, as the diagnosis made on another system may not be
531                  * valid in the current context.  Local vdevs will
532                  * remain in the faulted state.
533                  */
534                 if (spa_load_state(spa) == SPA_LOAD_OPEN) {
535                         (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_FAULTED,
536                             &vd->vdev_faulted);
537                         (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_DEGRADED,
538                             &vd->vdev_degraded);
539                         (void) nvlist_lookup_uint64(nv, ZPOOL_CONFIG_REMOVED,
540                             &vd->vdev_removed);

542                         if (vd->vdev_faulted || vd->vdev_degraded) {
543                                 char *aux;

545                                 vd->vdev_label_aux =
546                                     VDEV_AUX_ERR_EXCEEDED;
547                                 if (nvlist_lookup_string(nv,
548                                     ZPOOL_CONFIG_AUX_STATE, &aux) == 0 &&
549                                     strcmp(aux, "external") == 0)
550                                         vd->vdev_label_aux = VDEV_AUX_EXTERNAL;
551                         }
552                 }
553         }

555         /*
556          * Add ourselves to the parent's list of children.
557          */
558         vdev_add_child(parent, vd);

560         *vdp = vd;

562         return (0);
563 }

565 void
566 vdev_free(vdev_t *vd)
567 {
568         spa_t *spa = vd->vdev_spa;

570         /*
571          * vdev_free() implies closing the vdev first.  This is simpler than
572          * trying to ensure complicated semantics for all callers.
573          */
574         vdev_close(vd);

576         ASSERT(!list_link_active(&vd->vdev_config_dirty_node));
577         ASSERT(!list_link_active(&vd->vdev_state_dirty_node));

579         /*
580          * Free all children.
581          */
582         for (int c = 0; c < vd->vdev_children; c++)
583                 vdev_free(vd->vdev_child[c]);

585         ASSERT(vd->vdev_child == NULL);
586         ASSERT(vd->vdev_guid_sum == vd->vdev_guid);

588         /*
589          * Discard allocation state.
590          */
591         if (vd->vdev_mg != NULL) {
592                 vdev_metaslab_fini(vd);
593                 metaslab_group_destroy(vd->vdev_mg);
594         }
```

```
596         ASSERT0(vd->vdev_stat.vs_space);
597         ASSERT0(vd->vdev_stat.vs_dspace);
598         ASSERT0(vd->vdev_stat.vs_alloc);

600         /*
601          * Remove this vdev from its parent's child list.
602          */
603         vdev_remove_child(vd->vdev_parent, vd);

605         ASSERT(vd->vdev_parent == NULL);

607         /*
608          * Clean up vdev structure.
609          */
610         vdev_queue_fini(vd);
611         vdev_cache_fini(vd);

613         if (vd->vdev_path)
614                 spa_strfree(vd->vdev_path);
615         if (vd->vdev_devid)
616                 spa_strfree(vd->vdev_devid);
617         if (vd->vdev_physpath)
618                 spa_strfree(vd->vdev_physpath);
619         if (vd->vdev_fru)
620                 spa_strfree(vd->vdev_fru);

622         if (vd->vdev_isspare)
623                 spa_spare_remove(vd);
624         if (vd->vdev_isl2cache)
625                 spa_l2cache_remove(vd);

627         txg_list_destroy(&vd->vdev_ms_list);
628         txg_list_destroy(&vd->vdev_dtl_list);

630         mutex_enter(&vd->vdev_dtl_lock);
631         space_map_close(vd->vdev_dtl_sm);
632         for (int t = 0; t < DTL_TYPES; t++) {
633                 range_tree_vacate(vd->vdev_dtl[t], NULL, NULL);
634                 range_tree_destroy(vd->vdev_dtl[t]);
631                 space_map_unload(&vd->vdev_dtl[t]);
632                 space_map_destroy(&vd->vdev_dtl[t]);
635         }
636         mutex_exit(&vd->vdev_dtl_lock);

638         mutex_destroy(&vd->vdev_dtl_lock);
639         mutex_destroy(&vd->vdev_stat_lock);
640         mutex_destroy(&vd->vdev_probe_lock);

642         if (vd == spa->spa_root_vdev)
643                 spa->spa_root_vdev = NULL;

645         kmem_free(vd, sizeof (vdev_t));
646 }
______unchanged_portion_omitted_

802 int
803 vdev_metaslab_init(vdev_t *vd, uint64_t txg)
804 {
805         spa_t *spa = vd->vdev_spa;
806         objset_t *mos = spa->spa_meta_objset;
807         uint64_t m;
808         uint64_t oldc = vd->vdev_ms_count;
809         uint64_t newc = vd->vdev_asize >> vd->vdev_ms_shift;
810         metaslab_t **mspp;
811         int error;
```



```
813         ASSERT(txg == 0 || spa_config_held(spa, SCL_ALLOC, RW_WRITER));

815         /*
816          * This vdev is not being allocated from yet or is a hole.
817          */
818         if (vd->vdev_ms_shift == 0)
819                 return (0);

821         ASSERT(!vd->vdev_ishole);

823         /*
824          * Compute the raidz-deflation ratio.  Note, we hard-code
825          * in 128k (1 << 17) because it is the current "typical" blocksize.
826          * Even if SPA_MAXBLOCKSIZE changes, this algorithm must never change,
827          * or we will inconsistently account for existing bp's.
828          */
829         vd->vdev_deflate_ratio = (1 << 17) /
830             (vdev_psize_to_asize(vd, 1 << 17) >> SPA_MINBLOCKSHIFT);

832         ASSERT(oldc <= newc);

834         mspp = kmem_zalloc(newc * sizeof (*mspp), KM_SLEEP);

836         if (oldc != 0) {
837                 bcopy(vd->vdev_ms, mspp, oldc * sizeof (*mspp));
838                 kmem_free(vd->vdev_ms, oldc * sizeof (*mspp));
839         }

841         vd->vdev_ms = mspp;
842         vd->vdev_ms_count = newc;

844         for (m = oldc; m < newc; m++) {
845                 uint64_t object = 0;

843                 space_map_obj_t smo = { 0, 0, 0 };
847                 if (txg == 0) {
845                         uint64_t object = 0;
848                         error = dmu_read(mos, vd->vdev_ms_array,
849                             m * sizeof (uint64_t), sizeof (uint64_t), &object,
850                             DMU_READ_PREFETCH);
851                         if (error)
852                                 return (error);
851                         if (object != 0) {
852                                 dmu_buf_t *db;
853                                 error = dmu_bonus_hold(mos, object, FTAG, &db);
854                                 if (error)
855                                         return (error);
856                                 ASSERT3U(db->db_size, >=, sizeof (smo));
857                                 bcopy(db->db_data, &smo, sizeof (smo));
858                                 ASSERT3U(smo.smo_object, ==, object);
859                                 dmu_buf_rele(db, FTAG);
853                         }
854                         vd->vdev_ms[m] = metaslab_init(vd->vdev_mg, m, object, txg);
855                 }
862                 vd->vdev_ms[m] = metaslab_init(vd->vdev_mg, &smo,
863                     m << vd->vdev_ms_shift, 1ULL << vd->vdev_ms_shift, txg);
864         }

857         if (txg == 0)
858                 spa_config_enter(spa, SCL_ALLOC, FTAG, RW_WRITER);

860         /*
861          * If the vdev is being removed we don't activate
862          * the metaslabs since we want to ensure that no new
863          * allocations are performed on this device.
864          */
```

```
 865         if (oldc == 0 && !vd->vdev_removing)
 866                 metaslab_group_activate(vd->vdev_mg);
 867
 868         if (txg == 0)
 869                 spa_config_exit(spa, SCL_ALLOC, FTAG);
 870
 871         return (0);
 872 }
 873
 874 void
 875 vdev_metaslab_fini(vdev_t *vd)
 876 {
 877         uint64_t m;
 878         uint64_t count = vd->vdev_ms_count;
 879
 880         if (vd->vdev_ms != NULL) {
 881                 metaslab_group_passivate(vd->vdev_mg);
 882                 for (m = 0; m < count; m++) {
 883                         metaslab_t *msp = vd->vdev_ms[m];
 884
 885                         if (msp != NULL)
 886                                 metaslab_fini(msp);
 887                 }
 891                 for (m = 0; m < count; m++)
 892                         if (vd->vdev_ms[m] != NULL)
 893                                 metaslab_fini(vd->vdev_ms[m]);
 888                 kmem_free(vd->vdev_ms, count * sizeof (metaslab_t *));
 889                 vd->vdev_ms = NULL;
 890         }
 891 }
_____unchanged_portion_omitted_

1520 int
1521 vdev_create(vdev_t *vd, uint64_t txg, boolean_t isreplacing)
1522 {
1523         int error;
1524
1525         /*
1526          * Normally, partial opens (e.g. of a mirror) are allowed.
1527          * For a create, however, we want to fail the request if
1528          * there are any components we can't open.
1529          */
1530         error = vdev_open(vd);
1531
1532         if (error || vd->vdev_state != VDEV_STATE_HEALTHY) {
1533                 vdev_close(vd);
1534                 return (error ? error : ENXIO);
1535         }
1536
1537         /*
1538          * Recursively load DTLs and initialize all labels.
1544          * Recursively initialize all labels.
1539          */
1540         if ((error = vdev_dtl_load(vd)) != 0 ||
1541             (error = vdev_label_init(vd, txg, isreplacing ?
1546         if ((error = vdev_label_init(vd, txg, isreplacing ?
1542             VDEV_LABEL_REPLACE : VDEV_LABEL_CREATE)) != 0) {
1543                 vdev_close(vd);
1544                 return (error);
1545         }
1546
1547         return (0);
1548 }
_____unchanged_portion_omitted_

1577 void
```



```
1578 vdev_dirty_leaves(vdev_t *vd, int flags, uint64_t txg)
1579 {
1580         for (int c = 0; c < vd->vdev_children; c++)
1581                 vdev_dirty_leaves(vd->vdev_child[c], flags, txg);
1582
1583         if (vd->vdev_ops->vdev_op_leaf)
1584                 vdev_dirty(vd->vdev_top, flags, vd, txg);
1585 }
1586
1587 /*
1588  * DTLs.
1589  *
1590  * A vdev's DTL (dirty time log) is the set of transaction groups for which
1591  * the vdev has less than perfect replication.  There are four kinds of DTL:
1592  *
1593  * DTL_MISSING: txgs for which the vdev has no valid copies of the data
1594  *
1595  * DTL_PARTIAL: txgs for which data is available, but not fully replicated
1596  *
1597  * DTL_SCRUB: the txgs that could not be repaired by the last scrub; upon
1598  *      scrub completion, DTL_SCRUB replaces DTL_MISSING in the range of
1599  *      txgs that was scrubbed.
1600  *
1601  * DTL_OUTAGE: txgs which cannot currently be read, whether due to
1602  *      persistent errors or just some device being offline.
1603  *      Unlike the other three, the DTL_OUTAGE map is not generally
1604  *      maintained; it's only computed when needed, typically to
1605  *      determine whether a device can be detached.
1606  *
1607  * For leaf vdevs, DTL_MISSING and DTL_PARTIAL are identical: the device
1608  * either has the data or it doesn't.
1609  *
1610  * For interior vdevs such as mirror and RAID-Z the picture is more complex.
1611  * A vdev's DTL_PARTIAL is the union of its children's DTL_PARTIALs, because
1612  * if any child is less than fully replicated, then so is its parent.
1613  * A vdev's DTL_MISSING is a modified union of its children's DTL_MISSINGs,
1614  * comprising only those txgs which appear in 'maxfaults' or more children;
1615  * those are the txgs we don't have enough replication to read.  For example,
1616  * double-parity RAID-Z can tolerate up to two missing devices (maxfaults == 2);
1617  * thus, its DTL_MISSING consists of the set of txgs that appear in more than
1618  * two child DTL_MISSING maps.
1619  *
1620  * It should be clear from the above that to compute the DTLs and outage maps
1621  * for all vdevs, it suffices to know just the leaf vdevs' DTL_MISSING maps.
1622  * Therefore, that is all we keep on disk.  When loading the pool, or after
1623  * a configuration change, we generate all other DTLs from first principles.
1624  */
1625 void
1626 vdev_dtl_dirty(vdev_t *vd, vdev_dtl_type_t t, uint64_t txg, uint64_t size)
1627 {
1628         range_tree_t *rt = vd->vdev_dtl[t];
1623         space_map_t *sm = &vd->vdev_dtl[t];
1629
1630         ASSERT(t < DTL_TYPES);
1631         ASSERT(vd != vd->vdev_spa->spa_root_vdev);
1632         ASSERT(spa_writeable(vd->vdev_spa));
1633
1634         mutex_enter(rt->rt_lock);
1635         if (!range_tree_contains(rt, txg, size))
1636                 range_tree_add(rt, txg, size);
1637         mutex_exit(rt->rt_lock);
1629         mutex_enter(sm->sm_lock);
1630         if (!space_map_contains(sm, txg, size))
1631                 space_map_add(sm, txg, size);
1632         mutex_exit(sm->sm_lock);
1638 }
```

```
boolean_t
vdev_dtl_contains(vdev_t *vd, vdev_dtl_type_t t, uint64_t txg, uint64_t size)
{
        range_tree_t *rt = vd->vdev_dtl[t];
        space_map_t *sm = &vd->vdev_dtl[t];
        boolean_t dirty = B_FALSE;

        ASSERT(t < DTL_TYPES);
        ASSERT(vd != vd->vdev_spa->spa_root_vdev);

        mutex_enter(rt->rt_lock);
        if (range_tree_space(rt) != 0)
                dirty = range_tree_contains(rt, txg, size);
        mutex_exit(rt->rt_lock);
        mutex_enter(sm->sm_lock);
        if (sm->sm_space != 0)
                dirty = space_map_contains(sm, txg, size);
        mutex_exit(sm->sm_lock);

        return (dirty);
}

boolean_t
vdev_dtl_empty(vdev_t *vd, vdev_dtl_type_t t)
{
        range_tree_t *rt = vd->vdev_dtl[t];
        space_map_t *sm = &vd->vdev_dtl[t];
        boolean_t empty;

        mutex_enter(rt->rt_lock);
        empty = (range_tree_space(rt) == 0);
        mutex_exit(rt->rt_lock);
        mutex_enter(sm->sm_lock);
        empty = (sm->sm_space == 0);
        mutex_exit(sm->sm_lock);

        return (empty);
}

/*
 * Returns the lowest txg in the DTL range.
 */
static uint64_t
vdev_dtl_min(vdev_t *vd)
{
        range_seg_t *rs;
        space_seg_t *ss;

        ASSERT(MUTEX_HELD(&vd->vdev_dtl_lock));
        ASSERT3U(range_tree_space(vd->vdev_dtl[DTL_MISSING]), !=, 0);
        ASSERT3U(vd->vdev_dtl[DTL_MISSING].sm_space, !=, 0);
        ASSERT0(vd->vdev_children);

        rs = avl_first(&vd->vdev_dtl[DTL_MISSING]->rt_root);
        return (rs->rs_start - 1);
        ss = avl_first(&vd->vdev_dtl[DTL_MISSING].sm_root);
        return (ss->ss_start - 1);
}

/*
 * Returns the highest txg in the DTL.
 */
static uint64_t
vdev_dtl_max(vdev_t *vd)
{
```



```
        range_seg_t *rs;
        space_seg_t *ss;

        ASSERT(MUTEX_HELD(&vd->vdev_dtl_lock));
        ASSERT3U(range_tree_space(vd->vdev_dtl[DTL_MISSING]), !=, 0);
        ASSERT3U(vd->vdev_dtl[DTL_MISSING].sm_space, !=, 0);
        ASSERT0(vd->vdev_children);

        rs = avl_last(&vd->vdev_dtl[DTL_MISSING]->rt_root);
        return (rs->rs_end);
        ss = avl_last(&vd->vdev_dtl[DTL_MISSING].sm_root);
        return (ss->ss_end);
}

/*
 * Determine if a resilvering vdev should remove any DTL entries from
 * its range. If the vdev was resilvering for the entire duration of the
 * scan then it should excise that range from its DTLs. Otherwise, this
 * vdev is considered partially resilvered and should leave its DTL
 * entries intact. The comment in vdev_dtl_reassess() describes how we
 * excise the DTLs.
 */
static boolean_t
vdev_dtl_should_excise(vdev_t *vd)
{
        spa_t *spa = vd->vdev_spa;
        dsl_scan_t *scn = spa->spa_dsl_pool->dp_scan;

        ASSERT0(scn->scn_phys.scn_errors);
        ASSERT0(vd->vdev_children);

        if (vd->vdev_resilver_txg == 0 ||
            range_tree_space(vd->vdev_dtl[DTL_MISSING]) == 0)
            vd->vdev_dtl[DTL_MISSING].sm_space == 0)
                return (B_TRUE);

        /*
         * When a resilver is initiated the scan will assign the scn_max_txg
         * value to the highest txg value that exists in all DTLs. If this
         * device's max DTL is not part of this scan (i.e. it is not in
         * the range (scn_min_txg, scn_max_txg] then it is not eligible
         * for excision.
         */
        if (vdev_dtl_max(vd) <= scn->scn_phys.scn_max_txg) {
                ASSERT3U(scn->scn_phys.scn_min_txg, <=, vdev_dtl_min(vd));
                ASSERT3U(scn->scn_phys.scn_min_txg, <, vd->vdev_resilver_txg);
                ASSERT3U(vd->vdev_resilver_txg, <=, scn->scn_phys.scn_max_txg);
                return (B_TRUE);
        }
        return (B_FALSE);
}

/*
 * Reassess DTLs after a config change or scrub completion.
 */
void
vdev_dtl_reassess(vdev_t *vd, uint64_t txg, uint64_t scrub_txg, int scrub_done)
{
        spa_t *spa = vd->vdev_spa;
        avl_tree_t reftree;
        int minref;

        ASSERT(spa_config_held(spa, SCL_ALL, RW_READER) != 0);

        for (int c = 0; c < vd->vdev_children; c++)
                vdev_dtl_reassess(vd->vdev_child[c], txg,
```

```
1753                    scrub_txg, scrub_done);

1755        if (vd == spa->spa_root_vdev || vd->vdev_ishole || vd->vdev_aux)
1756                return;

1758        if (vd->vdev_ops->vdev_op_leaf) {
1759                dsl_scan_t *scn = spa->spa_dsl_pool->dp_scan;

1761                mutex_enter(&vd->vdev_dtl_lock);

1763                /*
1764                 * If we've completed a scan cleanly then determine
1765                 * if this vdev should remove any DTLs. We only want to
1766                 * excise regions on vdevs that were available during
1767                 * the entire duration of this scan.
1768                 */
1769                if (scrub_txg != 0 &&
1770                    (spa->spa_scrub_started ||
1771                    (scn != NULL && scn->scn_phys.scn_errors == 0)) &&
1772                    vdev_dtl_should_excise(vd)) {
1773                        /*
1774                         * We completed a scrub up to scrub_txg.  If we
1775                         * did it without rebooting, then the scrub dtl
1776                         * will be valid, so excise the old region and
1777                         * fold in the scrub dtl.  Otherwise, leave the
1778                         * dtl as-is if there was an error.
1779                         *
1780                         * There's little trick here: to excise the beginning
1781                         * of the DTL_MISSING map, we put it into a reference
1782                         * tree and then add a segment with refcnt -1 that
1783                         * covers the range [0, scrub_txg).  This means
1784                         * that each txg in that range has refcnt -1 or 0.
1785                         * We then add DTL_SCRUB with a refcnt of 2, so that
1786                         * entries in the range [0, scrub_txg) will have a
1787                         * positive refcnt -- either 1 or 2.  We then convert
1788                         * the reference tree into the new DTL_MISSING map.
1789                         */
1790                        space_reftree_create(&reftree);
1791                        space_reftree_add_map(&reftree,
1792                            vd->vdev_dtl[DTL_MISSING], 1);
1793                        space_reftree_add_seg(&reftree, 0, scrub_txg, -1);
1794                        space_reftree_add_map(&reftree,
1795                            vd->vdev_dtl[DTL_SCRUB], 2);
1796                        space_reftree_generate_map(&reftree,
1797                            vd->vdev_dtl[DTL_MISSING], 1);
1798                        space_reftree_destroy(&reftree);
1785                        space_map_ref_create(&reftree);
1786                        space_map_ref_add_map(&reftree,
1787                            &vd->vdev_dtl[DTL_MISSING], 1);
1788                        space_map_ref_add_seg(&reftree, 0, scrub_txg, -1);
1789                        space_map_ref_add_map(&reftree,
1790                            &vd->vdev_dtl[DTL_SCRUB], 2);
1791                        space_map_ref_generate_map(&reftree,
1792                            &vd->vdev_dtl[DTL_MISSING], 1);
1793                        space_map_ref_destroy(&reftree);
1799                }
1800                range_tree_vacate(vd->vdev_dtl[DTL_PARTIAL], NULL, NULL);
1801                range_tree_walk(vd->vdev_dtl[DTL_MISSING],
1802                    range_tree_add, vd->vdev_dtl[DTL_PARTIAL]);
1795                space_map_vacate(&vd->vdev_dtl[DTL_PARTIAL], NULL, NULL);
1796                space_map_walk(&vd->vdev_dtl[DTL_MISSING],
1797                    space_map_add, &vd->vdev_dtl[DTL_PARTIAL]);
1803                if (scrub_done)
1804                        range_tree_vacate(vd->vdev_dtl[DTL_SCRUB], NULL, NULL);
1805                range_tree_vacate(vd->vdev_dtl[DTL_OUTAGE], NULL, NULL);
1799                        space_map_vacate(&vd->vdev_dtl[DTL_SCRUB], NULL, NULL);
```



```
1800                space_map_vacate(&vd->vdev_dtl[DTL_OUTAGE], NULL, NULL);
1806                if (!vdev_readable(vd))
1807                        range_tree_add(vd->vdev_dtl[DTL_OUTAGE], 0, -1ULL);
1802                        space_map_add(&vd->vdev_dtl[DTL_OUTAGE], 0, -1ULL);
1808                else
1809                        range_tree_walk(vd->vdev_dtl[DTL_MISSING],
1810                            range_tree_add, vd->vdev_dtl[DTL_OUTAGE]);
1804                        space_map_walk(&vd->vdev_dtl[DTL_MISSING],
1805                            space_map_add, &vd->vdev_dtl[DTL_OUTAGE]);

1812                /*
1813                 * If the vdev was resilvering and no longer has any
1814                 * DTLs then reset its resilvering flag.
1815                 */
1816                if (vd->vdev_resilver_txg != 0 &&
1817                    range_tree_space(vd->vdev_dtl[DTL_MISSING]) == 0 &&
1818                    range_tree_space(vd->vdev_dtl[DTL_OUTAGE]) == 0)
1812                    vd->vdev_dtl[DTL_MISSING].sm_space == 0 &&
1813                    vd->vdev_dtl[DTL_OUTAGE].sm_space == 0)
1819                        vd->vdev_resilver_txg = 0;

1821                mutex_exit(&vd->vdev_dtl_lock);

1823                if (txg != 0)
1824                        vdev_dirty(vd->vdev_top, VDD_DTL, vd, txg);
1825                return;
1826        }

1828        mutex_enter(&vd->vdev_dtl_lock);
1829        for (int t = 0; t < DTL_TYPES; t++) {
1830                /* account for child's outage in parent's missing map */
1831                int s = (t == DTL_MISSING) ? DTL_OUTAGE: t;
1832                if (t == DTL_SCRUB)
1833                        continue;                       /* leaf vdevs only */
1834                if (t == DTL_PARTIAL)
1835                        minref = 1;                     /* i.e. non-zero */
1836                else if (vd->vdev_nparity != 0)
1837                        minref = vd->vdev_nparity + 1;  /* RAID-Z */
1838                else
1839                        minref = vd->vdev_children;     /* any kind of mirror */
1840                space_reftree_create(&reftree);
1835                space_map_ref_create(&reftree);
1841                for (int c = 0; c < vd->vdev_children; c++) {
1842                        vdev_t *cvd = vd->vdev_child[c];
1843                        mutex_enter(&cvd->vdev_dtl_lock);
1844                        space_reftree_add_map(&reftree, cvd->vdev_dtl[s], 1);
1839                        space_map_ref_add_map(&reftree, &cvd->vdev_dtl[s], 1);
1845                        mutex_exit(&cvd->vdev_dtl_lock);
1846                }
1847                space_reftree_generate_map(&reftree, vd->vdev_dtl[t], minref);
1848                space_reftree_destroy(&reftree);
1842                space_map_ref_generate_map(&reftree, &vd->vdev_dtl[t], minref);
1843                space_map_ref_destroy(&reftree);
1849        }
1850        mutex_exit(&vd->vdev_dtl_lock);
1851 }

1853 int
1848 static int
1854 vdev_dtl_load(vdev_t *vd)
1855 {
1856        spa_t *spa = vd->vdev_spa;
1852        space_map_obj_t *smo = &vd->vdev_dtl_smo;
1857        objset_t *mos = spa->spa_meta_objset;
1858        int error = 0;
1854        dmu_buf_t *db;
```

```
1855         int error;

1860         if (vd->vdev_ops->vdev_op_leaf && vd->vdev_dtl_object != 0) {
1857         ASSERT(vd->vdev_children == 0);

1859         if (smo->smo_object == 0)
1860                 return (0);

1861                 ASSERT(!vd->vdev_ishole);

1863                 error = space_map_open(&vd->vdev_dtl_sm, mos,
1864                     vd->vdev_dtl_object, 0, -1ULL, 0, &vd->vdev_dtl_lock);
1865                 if (error)
1864         if ((error = dmu_bonus_hold(mos, smo->smo_object, FTAG, &db)) != 0)
1866                         return (error);
1867                 ASSERT(vd->vdev_dtl_sm != NULL);

1869                 mutex_enter(&vd->vdev_dtl_lock);
1867         ASSERT3U(db->db_size, >=, sizeof (*smo));
1868         bcopy(db->db_data, smo, sizeof (*smo));
1869         dmu_buf_rele(db, FTAG);

1871                 /*
1872                  * Now that we've opened the space_map we need to update
1873                  * the in-core DTL.
1874                  */
1875                 space_map_update(vd->vdev_dtl_sm);

1877                 error = space_map_load(vd->vdev_dtl_sm,
1878                     vd->vdev_dtl[DTL_MISSING], SM_ALLOC);
1871         mutex_enter(&vd->vdev_dtl_lock);
1872         error = space_map_load(&vd->vdev_dtl[DTL_MISSING],
1873             NULL, SM_ALLOC, smo, mos);
1879                 mutex_exit(&vd->vdev_dtl_lock);

1881                 return (error);
1882         }

1884         for (int c = 0; c < vd->vdev_children; c++) {
1885                 error = vdev_dtl_load(vd->vdev_child[c]);
1886                 if (error != 0)
1887                         break;
1888         }

1890         return (error);
1891 }

1893 void
1894 vdev_dtl_sync(vdev_t *vd, uint64_t txg)
1895 {
1896         spa_t *spa = vd->vdev_spa;
1897         range_tree_t *rt = vd->vdev_dtl[DTL_MISSING];
1883         space_map_obj_t *smo = &vd->vdev_dtl_smo;
1884         space_map_t *sm = &vd->vdev_dtl[DTL_MISSING];
1898         objset_t *mos = spa->spa_meta_objset;
1899         range_tree_t *rtsync;
1900         kmutex_t rtlock;
1886         space_map_t smsync;
1887         kmutex_t smlock;
1888         dmu_buf_t *db;
1901         dmu_tx_t *tx;
1902         uint64_t object = space_map_object(vd->vdev_dtl_sm);

1904         ASSERT(!vd->vdev_ishole);
1905         ASSERT(vd->vdev_ops->vdev_op_leaf);
```



```
1907         tx = dmu_tx_create_assigned(spa->spa_dsl_pool, txg);

1909         if (vd->vdev_detached || vd->vdev_top->vdev_removing) {
1910                 mutex_enter(&vd->vdev_dtl_lock);
1911                 space_map_free(vd->vdev_dtl_sm, tx);
1912                 space_map_close(vd->vdev_dtl_sm);
1913                 vd->vdev_dtl_sm = NULL;
1914                 mutex_exit(&vd->vdev_dtl_lock);
1895         if (vd->vdev_detached) {
1896                 if (smo->smo_object != 0) {
1897                         int err = dmu_object_free(mos, smo->smo_object, tx);
1898                         ASSERT0(err);
1899                         smo->smo_object = 0;
1900                 }
1915                 dmu_tx_commit(tx);
1916                 return;
1917         }

1919         if (vd->vdev_dtl_sm == NULL) {
1920                 uint64_t new_object;

1922                 new_object = space_map_alloc(mos, tx);
1923                 VERIFY3U(new_object, !=, 0);

1925                 VERIFY0(space_map_open(&vd->vdev_dtl_sm, mos, new_object,
1926                     0, -1ULL, 0, &vd->vdev_dtl_lock));
1927                 ASSERT(vd->vdev_dtl_sm != NULL);
1905         if (smo->smo_object == 0) {
1906                 ASSERT(smo->smo_objsize == 0);
1907                 ASSERT(smo->smo_alloc == 0);
1908                 smo->smo_object = dmu_object_alloc(mos,
1909                     DMU_OT_SPACE_MAP, 1 << SPACE_MAP_BLOCKSHIFT,
1910                     DMU_OT_SPACE_MAP_HEADER, sizeof (*smo), tx);
1911                 ASSERT(smo->smo_object != 0);
1912                 vdev_config_dirty(vd->vdev_top);
1928         }

1930         mutex_init(&rtlock, NULL, MUTEX_DEFAULT, NULL);
1915         mutex_init(&smlock, NULL, MUTEX_DEFAULT, NULL);

1932         rtsync = range_tree_create(NULL, NULL, &rtlock);
1917         space_map_create(&smsync, sm->sm_start, sm->sm_size, sm->sm_shift,
1918             &smlock);

1934         mutex_enter(&rtlock);
1920         mutex_enter(&smlock);

1936         mutex_enter(&vd->vdev_dtl_lock);
1937         range_tree_walk(rt, range_tree_add, rtsync);
1923         space_map_walk(sm, space_map_add, &smsync);
1938         mutex_exit(&vd->vdev_dtl_lock);

1940         space_map_truncate(vd->vdev_dtl_sm, tx);
1941         space_map_write(vd->vdev_dtl_sm, rtsync, SM_ALLOC, tx);
1942         range_tree_vacate(rtsync, NULL, NULL);
1926         space_map_truncate(smo, mos, tx);
1927         space_map_sync(&smsync, SM_ALLOC, smo, mos, tx);
1928         space_map_vacate(&smsync, NULL, NULL);

1944         range_tree_destroy(rtsync);
1930         space_map_destroy(&smsync);

1946         mutex_exit(&rtlock);
1947         mutex_destroy(&rtlock);
1932         mutex_exit(&smlock);
1933         mutex_destroy(&smlock);
```

```
1949         /*
1950          * If the object for the space map has changed then dirty
1951          * the top level so that we update the config.
1952          */
1953         if (object != space_map_object(vd->vdev_dtl_sm)) {
1954                 zfs_dbgmsg("txg %llu, spa %s, DTL old object %llu, "
1955                     "new object %llu", txg, spa_name(spa), object,
1956                     space_map_object(vd->vdev_dtl_sm));
1957                 vdev_config_dirty(vd->vdev_top);
1958         }
1935         VERIFY(0 == dmu_bonus_hold(mos, smo->smo_object, FTAG, &db));
1936         dmu_buf_will_dirty(db, tx);
1937         ASSERT3U(db->db_size, >=, sizeof (*smo));
1938         bcopy(smo, db->db_data, sizeof (*smo));
1939         dmu_buf_rele(db, FTAG);

1960         dmu_tx_commit(tx);

1962         mutex_enter(&vd->vdev_dtl_lock);
1963         space_map_update(vd->vdev_dtl_sm);
1964         mutex_exit(&vd->vdev_dtl_lock);
1965 }
```

_____unchanged_portion_omitted_

```
2001 /*
2002  * Determine if resilver is needed, and if so the txg range.
2003  */
2004 boolean_t
2005 vdev_resilver_needed(vdev_t *vd, uint64_t *minp, uint64_t *maxp)
2006 {
2007         boolean_t needed = B_FALSE;
2008         uint64_t thismin = UINT64_MAX;
2009         uint64_t thismax = 0;

2011         if (vd->vdev_children == 0) {
2012                 mutex_enter(&vd->vdev_dtl_lock);
2013                 if (range_tree_space(vd->vdev_dtl[DTL_MISSING]) != 0 &&
1990                 if (vd->vdev_dtl[DTL_MISSING].sm_space != 0 &&
2014                     vdev_writeable(vd)) {

2016                         thismin = vdev_dtl_min(vd);
2017                         thismax = vdev_dtl_max(vd);
2018                         needed = B_TRUE;
2019                 }
2020                 mutex_exit(&vd->vdev_dtl_lock);
2021         } else {
2022                 for (int c = 0; c < vd->vdev_children; c++) {
2023                         vdev_t *cvd = vd->vdev_child[c];
2024                         uint64_t cmin, cmax;

2026                         if (vdev_resilver_needed(cvd, &cmin, &cmax)) {
2027                                 thismin = MIN(thismin, cmin);
2028                                 thismax = MAX(thismax, cmax);
2029                                 needed = B_TRUE;
2030                         }
2031                 }
2032         }

2034         if (needed && minp) {
2035                 *minp = thismin;
2036                 *maxp = thismax;
2037         }
2038         return (needed);
2039 }
```

_____unchanged_portion_omitted_


```
2109 void
2110 vdev_remove(vdev_t *vd, uint64_t txg)
2111 {
2112         spa_t *spa = vd->vdev_spa;
2113         objset_t *mos = spa->spa_meta_objset;
2114         dmu_tx_t *tx;

2116         tx = dmu_tx_create_assigned(spa_get_dsl(spa), txg);

2095         if (vd->vdev_dtl_smo.smo_object) {
2096                 ASSERT0(vd->vdev_dtl_smo.smo_alloc);
2097                 (void) dmu_object_free(mos, vd->vdev_dtl_smo.smo_object, tx);
2098                 vd->vdev_dtl_smo.smo_object = 0;
2099         }

2118         if (vd->vdev_ms != NULL) {
2119                 for (int m = 0; m < vd->vdev_ms_count; m++) {
2120                         metaslab_t *msp = vd->vdev_ms[m];

2122                         if (msp == NULL || msp->ms_sm == NULL)
2105                         if (msp == NULL || msp->ms_smo.smo_object == 0)
2123                                 continue;

2125                         mutex_enter(&msp->ms_lock);
2126                         VERIFY0(space_map_allocated(msp->ms_sm));
2127                         space_map_free(msp->ms_sm, tx);
2128                         space_map_close(msp->ms_sm);
2129                         msp->ms_sm = NULL;
2130                         mutex_exit(&msp->ms_lock);
2108                         ASSERT0(msp->ms_smo.smo_alloc);
2109                         (void) dmu_object_free(mos, msp->ms_smo.smo_object, tx);
2110                         msp->ms_smo.smo_object = 0;
2131                 }
2132         }

2134         if (vd->vdev_ms_array) {
2135                 (void) dmu_object_free(mos, vd->vdev_ms_array, tx);
2136                 vd->vdev_ms_array = 0;
2117                 vd->vdev_ms_shift = 0;
2137         }
2138         dmu_tx_commit(tx);
2139 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   37492 Tue Sep  3 20:27:14 2013
new/usr/src/uts/common/fs/zfs/vdev_label.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
______unchanged_portion_omitted_

 210 /*
 211  * Generate the nvlist representing this vdev's config.
 212  */
 213 nvlist_t *
 214 vdev_config_generate(spa_t *spa, vdev_t *vd, boolean_t getstats,
 215     vdev_config_flag_t flags)
 216 {
 217         nvlist_t *nv = NULL;

 219         nv = fnvlist_alloc();

 221         fnvlist_add_string(nv, ZPOOL_CONFIG_TYPE, vd->vdev_ops->vdev_op_type);
 222         if (!(flags & (VDEV_CONFIG_SPARE | VDEV_CONFIG_L2CACHE)))
 223                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_ID, vd->vdev_id);
 224         fnvlist_add_uint64(nv, ZPOOL_CONFIG_GUID, vd->vdev_guid);

 226         if (vd->vdev_path != NULL)
 227                 fnvlist_add_string(nv, ZPOOL_CONFIG_PATH, vd->vdev_path);

 229         if (vd->vdev_devid != NULL)
 230                 fnvlist_add_string(nv, ZPOOL_CONFIG_DEVID, vd->vdev_devid);

 232         if (vd->vdev_physpath != NULL)
 233                 fnvlist_add_string(nv, ZPOOL_CONFIG_PHYS_PATH,
 234                     vd->vdev_physpath);

 236         if (vd->vdev_fru != NULL)
 237                 fnvlist_add_string(nv, ZPOOL_CONFIG_FRU, vd->vdev_fru);

 239         if (vd->vdev_nparity != 0) {
 240                 ASSERT(strcmp(vd->vdev_ops->vdev_op_type,
 241                     VDEV_TYPE_RAIDZ) == 0);

 243                 /*
 244                  * Make sure someone hasn't managed to sneak a fancy new vdev
 245                  * into a crufty old storage pool.
 246                  */
 247                 ASSERT(vd->vdev_nparity == 1 ||
 248                     (vd->vdev_nparity <= 2 &&
 249                     spa_version(spa) >= SPA_VERSION_RAIDZ2) ||
 250                     (vd->vdev_nparity <= 3 &&
 251                     spa_version(spa) >= SPA_VERSION_RAIDZ3));

 253                 /*
 254                  * Note that we'll add the nparity tag even on storage pools
 255                  * that only support a single parity device -- older software
 256                  * will just ignore it.
 257                  */
 258                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_NPARITY, vd->vdev_nparity);
 259         }
```



```
 261         if (vd->vdev_wholedisk != -1ULL)
 262                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_WHOLE_DISK,
 263                     vd->vdev_wholedisk);

 265         if (vd->vdev_not_present)
 266                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_NOT_PRESENT, 1);

 268         if (vd->vdev_isspare)
 269                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_IS_SPARE, 1);

 271         if (!(flags & (VDEV_CONFIG_SPARE | VDEV_CONFIG_L2CACHE)) &&
 272             vd == vd->vdev_top) {
 273                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_METASLAB_ARRAY,
 274                     vd->vdev_ms_array);
 275                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_METASLAB_SHIFT,
 276                     vd->vdev_ms_shift);
 277                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_ASHIFT, vd->vdev_ashift);
 278                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_ASIZE,
 279                     vd->vdev_asize);
 280                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_IS_LOG, vd->vdev_islog);
 281                 if (vd->vdev_removing)
 282                         fnvlist_add_uint64(nv, ZPOOL_CONFIG_REMOVING,
 283                             vd->vdev_removing);
 284         }

 286         if (vd->vdev_dtl_sm != NULL) {
 286         if (vd->vdev_dtl_smo.smo_object != 0)
 287                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_DTL,
 288                     space_map_object(vd->vdev_dtl_sm));
 289         }
 288                     vd->vdev_dtl_smo.smo_object);

 291         if (vd->vdev_crtxg)
 292                 fnvlist_add_uint64(nv, ZPOOL_CONFIG_CREATE_TXG, vd->vdev_crtxg);

 294         if (getstats) {
 295                 vdev_stat_t vs;
 296                 pool_scan_stat_t ps;

 298                 vdev_get_stats(vd, &vs);
 299                 fnvlist_add_uint64_array(nv, ZPOOL_CONFIG_VDEV_STATS,
 300                     (uint64_t *)&vs, sizeof (vs) / sizeof (uint64_t));

 302                 /* provide either current or previous scan information */
 303                 if (spa_scan_get_stats(spa, &ps) == 0) {
 304                         fnvlist_add_uint64_array(nv,
 305                             ZPOOL_CONFIG_SCAN_STATS, (uint64_t *)&ps,
 306                             sizeof (pool_scan_stat_t) / sizeof (uint64_t));
 307                 }
 308         }

 310         if (!vd->vdev_ops->vdev_op_leaf) {
 311                 nvlist_t **child;
 312                 int c, idx;

 314                 ASSERT(!vd->vdev_ishole);

 316                 child = kmem_alloc(vd->vdev_children * sizeof (nvlist_t *),
 317                     KM_SLEEP);

 319                 for (c = 0, idx = 0; c < vd->vdev_children; c++) {
 320                         vdev_t *cvd = vd->vdev_child[c];

 322                         /*
 323                          * If we're generating an nvlist of removing
 324                          * vdevs then skip over any device which is
```

```
325                          * not being removed.
326                          */
327                         if ((flags & VDEV_CONFIG_REMOVING) &&
328                             !cvd->vdev_removing)
329                                 continue;
330
331                         child[idx++] = vdev_config_generate(spa, cvd,
332                             getstats, flags);
333                 }
334
335                 if (idx) {
336                         fnvlist_add_nvlist_array(nv, ZPOOL_CONFIG_CHILDREN,
337                             child, idx);
338                 }
339
340                 for (c = 0; c < idx; c++)
341                         nvlist_free(child[c]);
342
343                 kmem_free(child, vd->vdev_children * sizeof (nvlist_t *));
344
345         } else {
346                 const char *aux = NULL;
347
348                 if (vd->vdev_offline && !vd->vdev_tmpoffline)
349                         fnvlist_add_uint64(nv, ZPOOL_CONFIG_OFFLINE, B_TRUE);
350                 if (vd->vdev_resilver_txg != 0)
351                         fnvlist_add_uint64(nv, ZPOOL_CONFIG_RESILVER_TXG,
352                             vd->vdev_resilver_txg);
353                 if (vd->vdev_faulted)
354                         fnvlist_add_uint64(nv, ZPOOL_CONFIG_FAULTED, B_TRUE);
355                 if (vd->vdev_degraded)
356                         fnvlist_add_uint64(nv, ZPOOL_CONFIG_DEGRADED, B_TRUE);
357                 if (vd->vdev_removed)
358                         fnvlist_add_uint64(nv, ZPOOL_CONFIG_REMOVED, B_TRUE);
359                 if (vd->vdev_unspare)
360                         fnvlist_add_uint64(nv, ZPOOL_CONFIG_UNSPARE, B_TRUE);
361                 if (vd->vdev_ishole)
362                         fnvlist_add_uint64(nv, ZPOOL_CONFIG_IS_HOLE, B_TRUE);
363
364                 switch (vd->vdev_stat.vs_aux) {
365                 case VDEV_AUX_ERR_EXCEEDED:
366                         aux = "err_exceeded";
367                         break;
368
369                 case VDEV_AUX_EXTERNAL:
370                         aux = "external";
371                         break;
372                 }
373
374                 if (aux != NULL)
375                         fnvlist_add_string(nv, ZPOOL_CONFIG_AUX_STATE, aux);
376
377                 if (vd->vdev_splitting && vd->vdev_orig_guid != 0LL) {
378                         fnvlist_add_uint64(nv, ZPOOL_CONFIG_ORIG_GUID,
379                             vd->vdev_orig_guid);
380                 }
381         }
382
383         return (nv);
384 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   14617 Tue Sep  3 20:27:15 2013
new/usr/src/uts/common/fs/zfs/zfeature.c
4101 metaslab_debug should allow for fine-grained control
4102 space_maps should store more information about themselves
4103 space map object blocksize should be increased
4104 ::spa_space no longer works
4105 removing a mirrored log device results in a leaked object
4106 asynchronously load metaslab
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Sebastien Roy <seb@delphix.com>
**********************************************************
______unchanged_portion_omitted_
```

```
364 /*
365  * If the specified feature has not yet been enabled, this function returns
366  * ENOTSUP; otherwise, this function increments the feature's refcount (or
367  * returns EOVERFLOW if the refcount cannot be incremented). This function must
368  * be called from syncing context.
369  */
364 void
365 spa_feature_incr(spa_t *spa, zfeature_info_t *feature, dmu_tx_t *tx)
366 {
367         ASSERT(dmu_tx_is_syncing(tx));
368         ASSERT3U(spa_version(spa), >=, SPA_VERSION_FEATURES);
369         VERIFY3U(0, ==, feature_do_action(spa->spa_meta_objset,
370             spa->spa_feat_for_read_obj, spa->spa_feat_for_write_obj,
371             spa->spa_feat_desc_obj, feature, FEATURE_ACTION_INCR, tx));
372 }

379 /*
380  * If the specified feature has not yet been enabled, this function returns
381  * ENOTSUP; otherwise, this function decrements the feature's refcount (or
382  * returns EOVERFLOW if the refcount is already 0). This function must
383  * be called from syncing context.
384  */
374 void
375 spa_feature_decr(spa_t *spa, zfeature_info_t *feature, dmu_tx_t *tx)
376 {
377         ASSERT(dmu_tx_is_syncing(tx));
378         ASSERT3U(spa_version(spa), >=, SPA_VERSION_FEATURES);
379         VERIFY3U(0, ==, feature_do_action(spa->spa_meta_objset,
380             spa->spa_feat_for_read_obj, spa->spa_feat_for_write_obj,
381             spa->spa_feat_desc_obj, feature, FEATURE_ACTION_DECR, tx));
382 }

384 /*
385  * This interface is for debugging only. Normal consumers should use
386  * spa_feature_is_enabled/spa_feature_is_active.
387  */
388 int
389 spa_feature_get_refcount(spa_t *spa, zfeature_info_t *feature)
390 {
391         int err;
392         uint64_t refcount;

394         if (spa_version(spa) < SPA_VERSION_FEATURES)
395                 return (B_FALSE);

397         err = feature_get_refcount(spa->spa_meta_objset,
398             spa->spa_feat_for_read_obj, spa->spa_feat_for_write_obj,
399             feature, &refcount);
400         ASSERT(err == 0 || err == ENOTSUP);
401         return (err == 0 ? refcount : 0);
402 }
```



```
404 boolean_t
405 spa_feature_is_enabled(spa_t *spa, zfeature_info_t *feature)
406 {
407         int err;
408         uint64_t refcount;

410         if (spa_version(spa) < SPA_VERSION_FEATURES)
411                 return (B_FALSE);

413         err = feature_get_refcount(spa->spa_meta_objset,
414             spa->spa_feat_for_read_obj, spa->spa_feat_for_write_obj,
415             feature, &refcount);
416         ASSERT(err == 0 || err == ENOTSUP);
417         return (err == 0);
418 }
______unchanged_portion_omitted_
```