

new/usr/src/cmd/ztest/Makefile.com

1

```
*****
1819 Wed Apr 24 10:02:42 2013
new/usr/src/cmd/ztest/Makefile.com
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectrallogic.com>
Submitted by: Will Andrews <willa@spectrallogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.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 #
21 #
22 # Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
23 # Copyright (c) 2012 by Delphix. All rights reserved.
24 #
```

```
26 PROG= ztest
27 OBJS= $(PROG).o ztest_hdrck.o
28 SRCS= $(OBJS:%.o=./%.c) ztest_hdrck.cpp
27 OBJS= $(PROG).o
28 SRCS= $(OBJS:%.o=./%.c)
```

```
30 include ../../Makefile.cmd
31 include ../../Makefile.ctf
```

```
33 INCS += -I../../lib/libzpool/common
34 INCS += -I../../uts/common/fs/zfs
35 INCS += -I../../common/zfs
```

```
37 LDLIBS += -lumem -lzpool -lcmdutils -lm -lnvpair
```

```
39 C99MODE= -xc99=%all
40 C99LMODE= -Xc99=%all
41 CFLAGS += -g $(CCVERBOSE)
42 CFLAGS64 += -g $(CCVERBOSE)
43 CPPFLAGS += -D_LARGEFILE64_SOURCE=1 -D_REENTRANT $(INCS) -DDEBUG
```

```
45 # lint complains about unused _umem_* functions
46 LINTFLAGS += -xeroff=E_NAME_DEF_NOT_USED2
47 LINTFLAGS64 += -xeroff=E_NAME_DEF_NOT_USED2
```

```
49 CERRWARN += -_gcc=-Wno-switch
```

```
51 .KEEP_STATE:
```

```
53 all: $(PROG)
```

new/usr/src/cmd/ztest/Makefile.com

2

```
55 $(PROG): $(OBJS)
56     $(LINK.c) -o $(PROG) $(OBJS) $(LDLIBS)
57     $(POST_PROCESS)
```

```
59 clean:
```

```
61 lint: lint_SRCS
```

```
63 include ../../Makefile.targ
```

```
65 %.o: ../../%.cpp
66     $(COMPILE.cc) $<
67     $(POST_PROCESS_O)
```

```
69 #endif /* ! codereview */
70 %.o: ../../%.c
71     $(COMPILE.c) $<
72     $(POST_PROCESS_O)
```

new/usr/src/cmd/ztest/ztest\_hdrck.cpp

1

```
*****
2170 Wed Apr 24 10:02:42 2013
new/usr/src/cmd/ztest/ztest_hdrck.cpp
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectralogic.com>
Submitted by: Will Andrews <willa@spectralogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.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 */
21 /*
22 * Copyright (c) 2013 Spectra Logic Corporation. All rights reserved.
23 */

25 /*
26 * This file's primary purpose is to build-time enforce C++ compatibility of
27 * the ZFS headers. However, if desired, it can also be used for utility
28 * functions to be used by ztest.c.
29 */

31 /* Not sure why these are necessary? */
32 #ifndef B_TRUE
33 #define B_TRUE _B_TRUE
34 #endif
35 #ifndef B_FALSE
36 #define B_FALSE _B_FALSE
37 #endif

39 /* from ztest.c */
40 #include <sys/zfs_context.h>
41 #include <sys/spa.h>
42 #include <sys/dmu.h>
43 #include <sys/txg.h>
44 #include <sys/dbuf.h>
45 #include <sys/zap.h>
46 #include <sys/dmu_objset.h>
47 #include <sys/poll.h>
48 #include <sys/stat.h>
49 #include <sys/time.h>
50 #include <sys/wait.h>
51 #include <sys/mman.h>
52 #include <sys/resource.h>
53 #include <sys/zio.h>
54 #include <sys/zil.h>
55 #include <sys/zil_impl.h>
56 #include <sys/vdev_impl.h>
```

new/usr/src/cmd/ztest/ztest\_hdrck.cpp

2

```
57 #include <sys/vdev_file.h>
58 #include <sys/spa_impl.h>
59 #include <sys/metaslab_impl.h>
60 #include <sys/dsl_prop.h>
61 #include <sys/dsl_dataset.h>
62 #include <sys/dsl_destroy.h>
63 #include <sys/dsl_scan.h>
64 #include <sys/zio_checksum.h>
65 #include <sys/refcount.h>
66 #include <sys/zfeature.h>
67 #include <sys/dsl_userhold.h>
68 #include <stdio.h>
69 #include <stdio_ext.h>
70 #include <stdlib.h>
71 #include <unistd.h>
72 #include <signal.h>
73 #include <umem.h>
74 #include <dlfcn.h>
75 #include <ctype.h>
76 #include <math.h>
77 #include <sys/fs/zfs.h>
78 #include <libnvpair.h>
79 #endif /* !codereview */
```

```

*****
134667 Wed Apr 24 10:02:42 2013
new/usr/src/uts/common/fs/zfs/arc.c
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectralogic.com>
Submitted by: Will Andrews <willa@spectralogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
_____unchanged_portion_omitted_____

2769 /*
2770  * "Read" the block at the specified DVA (in bp) via the
2771  * cache. If the block is found in the cache, invoke the provided
2772  * callback immediately and return. Note that the 'zio' parameter
2773  * in the callback will be NULL in this case, since no IO was
2774  * required. If the block is not in the cache pass the read request
2775  * on to the spa with a substitute callback function, so that the
2776  * requested block will be added to the cache.
2777  *
2778  * If a read request arrives for a block that has a read in-progress,
2779  * either wait for the in-progress read to complete (and return the
2780  * results); or, if this is a read with a "done" func, add a record
2781  * to the read to invoke the "done" func when the read completes,
2782  * and return; or just return.
2783  *
2784  * arc_read_done() will invoke all the requested "done" functions
2785  * for readers of this block.
2786  */
2787 int
2788 arc_read(zio_t *pio, spa_t *spa, const blkptr_t *bp, arc_done_func_t *done,
2789 void *cb_private, int priority, int zio_flags, uint32_t *arc_flags,
2790 void *private, int priority, int zio_flags, uint32_t *arc_flags,
2791 const zbookmark_t *zb)
2792 {
2793     arc_buf_hdr_t *hdr;
2794     arc_buf_t *buf = NULL;
2795     kmutex_t *hash_lock;
2796     zio_t *rzio;
2797     uint64_t guid = spa_load_guid(spa);
2798
2799     top:
2800     hdr = buf_hash_find(guid, BP_IDENTITY(bp), BP_PHYSICAL_BIRTH(bp),
2801 &hash_lock);
2802     if (hdr && hdr->b_datacnt > 0) {
2803
2804         *arc_flags |= ARC_CACHED;
2805
2806         if (HDR_IO_IN_PROGRESS(hdr)) {
2807
2808             if (*arc_flags & ARC_WAIT) {
2809                 cv_wait(&hdr->b_cv, hash_lock);
2810                 mutex_exit(hash_lock);
2811                 goto top;
2812             }
2813             ASSERT(*arc_flags & ARC_NOWAIT);
2814
2815             if (done) {
2816                 arc_callback_t *acb = NULL;
2817
2818                 acb = kmem_zalloc(sizeof (arc_callback_t),
2819 KM_SLEEP);
2820                 acb->acb_done = done;
2821                 acb->acb_private = cb_private;
2822                 acb->acb_private = private;

```

```

2823         if (pio != NULL)
2824             acb->acb_zio_dummy = zio_null(pio,
2825 spa, NULL, NULL, NULL, zio_flags);
2826
2827         ASSERT(acb->acb_done != NULL);
2828         acb->acb_next = hdr->b_acb;
2829         hdr->b_acb = acb;
2830         add_reference(hdr, hash_lock, cb_private);
2831         add_reference(hdr, hash_lock, private);
2832         mutex_exit(hash_lock);
2833         return (0);
2834     }
2835
2836     ASSERT(hdr->b_state == arc_mru || hdr->b_state == arc_mfu);
2837
2838     if (done) {
2839         add_reference(hdr, hash_lock, cb_private);
2840         add_reference(hdr, hash_lock, private);
2841         /*
2842          * If this block is already in use, create a new
2843          * copy of the data so that we will be guaranteed
2844          * that arc_release() will always succeed.
2845          */
2846         buf = hdr->b_buf;
2847         ASSERT(buf);
2848         ASSERT(buf->b_data);
2849         if (HDR_BUF_AVAILABLE(hdr)) {
2850             ASSERT(buf->b_efunc == NULL);
2851             hdr->b_flags &= ~ARC_BUF_AVAILABLE;
2852         } else {
2853             buf = arc_buf_clone(buf);
2854         }
2855     } else if (*arc_flags & ARC_PREFETCH &&
2856 refcount_count(&hdr->b_refcnt) == 0) {
2857         hdr->b_flags |= ARC_PREFETCH;
2858     }
2859     DTRACE_PROBE1(arc_hit, arc_buf_hdr_t *, hdr);
2860     arc_access(hdr, hash_lock);
2861     if (*arc_flags & ARC_L2CACHE)
2862         hdr->b_flags |= ARC_L2CACHE;
2863     mutex_exit(hash_lock);
2864     ARCSTAT_BUMP(arcstat_hits);
2865     ARCSTAT_CONDSTAT(! (hdr->b_flags & ARC_PREFETCH),
2866 demand, prefetch, hdr->b_type != ARC_BUF_METADATA,
2867 data, metadata, hits);
2868
2869     if (done)
2870         done(NULL, buf, cb_private);
2871         done(NULL, buf, private);
2872     } else {
2873         uint64_t size = BP_GET_LSIZE(bp);
2874         arc_callback_t *acb;
2875         vdev_t *vd = NULL;
2876         uint64_t addr = 0;
2877         boolean_t devw = B_FALSE;
2878
2879         if (hdr == NULL) {
2880             /* this block is not in the cache */
2881             arc_buf_hdr_t *exists;
2882             arc_buf_contents_t type = BP_GET_BUF_TYPE(bp);
2883             buf = arc_buf_alloc(spa, size, cb_private, type);
2884             buf = arc_buf_alloc(spa, size, private, type);

```

```

2883     hdr = buf->b_hdr;
2884     hdr->b_dva = *BP_IDENTITY(bp);
2885     hdr->b_birth = BP_PHYSICAL_BIRTH(bp);
2886     hdr->b_cksum0 = bp->blk_cksum.zc_word[0];
2887     exists = buf_hash_insert(hdr, &hash_lock);
2888     if (exists) {
2889         /* somebody beat us to the hash insert */
2890         mutex_exit(hash_lock);
2891         buf_discard_identity(hdr);
2892         (void) arc_buf_remove_ref(buf, cb_private);
2892         (void) arc_buf_remove_ref(buf, private);
2893         goto top; /* restart the IO request */
2894     }
2895     /* if this is a prefetch, we don't have a reference */
2896     if (*arc_flags & ARC_PREFETCH) {
2897         (void) remove_reference(hdr, hash_lock,
2898             cb_private);
2898         private);
2899         hdr->b_flags |= ARC_PREFETCH;
2900     }
2901     if (*arc_flags & ARC_L2CACHE)
2902         hdr->b_flags |= ARC_L2CACHE;
2903     if (BP_GET_LEVEL(bp) > 0)
2904         hdr->b_flags |= ARC_INDIRECT;
2905 } else {
2906     /* this block is in the ghost cache */
2907     ASSERT(GHOST_STATE(hdr->b_state));
2908     ASSERT(!HDR_IO_IN_PROGRESS(hdr));
2909     ASSERT0(refcount_count(&hdr->b_refcnt));
2910     ASSERT(hdr->b_buf == NULL);
2911
2912     /* if this is a prefetch, we don't have a reference */
2913     if (*arc_flags & ARC_PREFETCH)
2914         hdr->b_flags |= ARC_PREFETCH;
2915     else
2916         add_reference(hdr, hash_lock, cb_private);
2916         add_reference(hdr, hash_lock, private);
2917     if (*arc_flags & ARC_L2CACHE)
2918         hdr->b_flags |= ARC_L2CACHE;
2919     buf = kmem_cache_alloc(buf_cache, KM_PUSHPAGE);
2920     buf->b_hdr = hdr;
2921     buf->b_data = NULL;
2922     buf->b_efunc = NULL;
2923     buf->b_private = NULL;
2924     buf->b_next = NULL;
2925     hdr->b_buf = buf;
2926     ASSERT(hdr->b_datacnt == 0);
2927     hdr->b_datacnt = 1;
2928     arc_get_data_buf(buf);
2929     arc_access(hdr, hash_lock);
2930 }
2931
2932     ASSERT(!GHOST_STATE(hdr->b_state));
2933
2934     acb = kmem_zalloc(sizeof(arc_callback_t), KM_SLEEP);
2935     acb->acb_done = done;
2936     acb->acb_private = cb_private;
2936     acb->acb_private = private;
2937
2938     ASSERT(hdr->b_acb == NULL);
2939     hdr->b_acb = acb;
2940     hdr->b_flags |= ARC_IO_IN_PROGRESS;
2941
2942     if (HDR_L2CACHE(hdr) && hdr->b_l2hdr != NULL &&
2943         (vd = hdr->b_l2hdr->b_dev->l2ad_vdev) != NULL) {
2944         devw = hdr->b_l2hdr->b_dev->l2ad_writing;

```

```

2945         addr = hdr->b_l2hdr->b_daddr;
2946         /*
2947          * Lock out device removal.
2948          */
2949         if (vdev_is_dead(vd) ||
2950             !spa_config_tryenter(spa, SCL_L2ARC, vd, RW_READER))
2951             vd = NULL;
2952     }
2953
2954     mutex_exit(hash_lock);
2955
2956     ASSERT3U(hdr->b_size, ==, size);
2957     DTRACE_PROBE4(arc__miss, arc_buf_hdr_t *, hdr, blkptr_t *, bp,
2958         uint64_t, size, zbookmark_t *, zb);
2959     ARCSTAT_BUMP(arcstat_misses);
2960     ARCSTAT_CONDSTAT(!{hdr->b_flags & ARC_PREFETCH},
2961         demand, prefetch, hdr->b_type != ARC_BUFC_METADATA,
2962         data, metadata, misses);
2963
2964     if (vd != NULL && l2arc_ndev != 0 && !(l2arc_norw && devw)) {
2965         /*
2966          * Read from the L2ARC if the following are true:
2967          * 1. The L2ARC vdev was previously cached.
2968          * 2. This buffer still has L2ARC metadata.
2969          * 3. This buffer isn't currently writing to the L2ARC.
2970          * 4. The L2ARC entry wasn't evicted, which may
2971          *    also have invalidated the vdev.
2972          * 5. This isn't prefetch and l2arc_noprefetch is set.
2973          */
2974         if (hdr->b_l2hdr != NULL &&
2975             !HDR_L2_WRITING(hdr) && !HDR_L2_EVICTED(hdr) &&
2976             !(l2arc_noprefetch && HDR_PREFETCH(hdr))) {
2977             l2arc_read_callback_t *cb;
2978
2979             DTRACE_PROBE1(l2arc_hit, arc_buf_hdr_t *, hdr);
2980             ARCSTAT_BUMP(arcstat_l2_hits);
2981
2982             cb = kmem_zalloc(sizeof(l2arc_read_callback_t),
2983                 KM_SLEEP);
2984             cb->l2rcb_buf = buf;
2985             cb->l2rcb_spa = spa;
2986             cb->l2rcb_bp = *bp;
2987             cb->l2rcb_zb = *zb;
2988             cb->l2rcb_flags = zio_flags;
2989
2990             ASSERT(addr >= VDEV_LABEL_START_SIZE &&
2991                 addr + size < vd->vdev_psize -
2992                 VDEV_LABEL_END_SIZE);
2993
2994             /*
2995              * l2arc read. The SCL_L2ARC lock will be
2996              * released by l2arc_read_done().
2997              */
2998             rzio = zio_read_phys(pio, vd, addr, size,
2999                 buf->b_data, ZIO_CHECKSUM_OFF,
3000                 l2arc_read_done, cb, priority, zio_flags |
3001                 ZIO_FLAG_DONT_CACHE | ZIO_FLAG_CANFAIL |
3002                 ZIO_FLAG_DONT_PROPAGATE |
3003                 ZIO_FLAG_DONT_RETRY, B_FALSE);
3004             DTRACE_PROBE2(l2arc__read, vdev_t *, vd,
3005                 zio_t *, rzio);
3006             ARCSTAT_INCR(arcstat_l2_read_bytes, size);
3007
3008             if (*arc_flags & ARC_NOWAIT) {
3009                 zio_nowait(rzio);
3010                 return (0);

```

```

3011     }
3013     ASSERT(*arc_flags & ARC_WAIT);
3014     if (zio_wait(rzio) == 0)
3015         return (0);
3017     /* l2arc read error; goto zio_read() */
3018     } else {
3019         DTRACE_PROBE1(l2arc__miss,
3020             arc_buf_hdr_t *, hdr);
3021         ARCSTAT_BUMP(arcstat_l2_misses);
3022         if (HDR_L2_WRITING(hdr))
3023             ARCSTAT_BUMP(arcstat_l2_rw_clash);
3024         spa_config_exit(spa, SCL_L2ARC, vd);
3025     }
3026     } else {
3027         if (vd != NULL)
3028             spa_config_exit(spa, SCL_L2ARC, vd);
3029         if (l2arc_ndev != 0) {
3030             DTRACE_PROBE1(l2arc__miss,
3031                 arc_buf_hdr_t *, hdr);
3032             ARCSTAT_BUMP(arcstat_l2_misses);
3033         }
3034     }
3036     rzio = zio_read(pio, spa, bp, buf->b_data, size,
3037         arc_read_done, buf, priority, zio_flags, zb);
3039     if (*arc_flags & ARC_WAIT)
3040         return (zio_wait(rzio));
3042     ASSERT(*arc_flags & ARC_NOWAIT);
3043     zio_nowait(rzio);
3044 }
3045 return (0);
3046 }

```

```

3048 void
3049 arc_set_callback(arc_buf_t *buf, arc_evict_func_t *func, void *cb_private)
3049 arc_set_callback(arc_buf_t *buf, arc_evict_func_t *func, void *private)
3050 {

```

```

3051     ASSERT(buf->b_hdr != NULL);
3052     ASSERT(buf->b_hdr->b_state != arc_anon);
3053     ASSERT(!refcount_is_zero(&buf->b_hdr->b_refcnt) || func == NULL);
3054     ASSERT(buf->b_efunc == NULL);
3055     ASSERT(!HDR_BUF_AVAILABLE(buf->b_hdr));

```

```

3057     buf->b_efunc = func;
3058     buf->b_private = cb_private;
3058     buf->b_private = private;
3059 }

```

unchanged portion omitted

```

3414 zio_t *
3415 arc_write(zio_t *pio, spa_t *spa, uint64_t txg,
3416     blkptr_t *bp, arc_buf_t *buf, boolean_t l2arc, const zio_prop_t *zp,
3417     arc_done_func_t *ready, arc_done_func_t *done, void *cb_private,
3417     arc_done_func_t *ready, arc_done_func_t *done, void *private,
3418     int priority, int zio_flags, const zbookmark_t *zb)
3419 {
3420     arc_buf_hdr_t *hdr = buf->b_hdr;
3421     arc_write_callback_t *callback;
3422     zio_t *zio;
3424     ASSERT(ready != NULL);
3425     ASSERT(done != NULL);

```

```

3426     ASSERT(!HDR_IO_ERROR(hdr));
3427     ASSERT((hdr->b_flags & ARC_IO_IN_PROGRESS) == 0);
3428     ASSERT(hdr->b_acb == NULL);
3429     if (l2arc)
3430         hdr->b_flags |= ARC_L2CACHE;
3431     callback = kmem_zalloc(sizeof (arc_write_callback_t), KM_SLEEP);
3432     callback->awcb_ready = ready;
3433     callback->awcb_done = done;
3434     callback->awcb_private = cb_private;
3434     callback->awcb_private = private;
3435     callback->awcb_buf = buf;
3437     zio = zio_write(pio, spa, txg, bp, buf->b_data, hdr->b_size, zp,
3438         arc_write_ready, arc_write_done, callback, priority, zio_flags, zb);
3440     return (zio);
3441 }

```

unchanged portion omitted

```

*****
27528 Wed Apr 24 10:02:42 2013
new/usr/src/uts/common/fs/zfs/ddt.c
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectralogic.com>
Submitted by: Will Andrews <willa@spectralogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
    unchanged_portion_omitted_

180 int
181 ddt_object_update(ddt_t *ddt, enum ddt_type type, enum ddt_class ddt_class,
181 ddt_object_update(ddt_t *ddt, enum ddt_type type, enum ddt_class class,
182     ddt_entry_t *dde, dmu_tx_t *tx)
183 {
184     ASSERT(ddt_object_exists(ddt, type, ddt_class));
184     ASSERT(ddt_object_exists(ddt, type, class));

186     return (ddt_ops[type]->ddt_op_update(ddt->ddt_os,
187         ddt->ddt_object[type][ddt_class], dde, tx));
187     ddt->ddt_object[type][class], dde, tx));
188 }
    unchanged_portion_omitted_

200 int
201 ddt_object_walk(ddt_t *ddt, enum ddt_type type, enum ddt_class ddt_class,
201 ddt_object_walk(ddt_t *ddt, enum ddt_type type, enum ddt_class class,
202     uint64_t *walk, ddt_entry_t *dde)
203 {
204     ASSERT(ddt_object_exists(ddt, type, ddt_class));
204     ASSERT(ddt_object_exists(ddt, type, class));

206     return (ddt_ops[type]->ddt_op_walk(ddt->ddt_os,
207         ddt->ddt_object[type][ddt_class], dde, walk));
207     ddt->ddt_object[type][class], dde, walk));
208 }

210 uint64_t
211 ddt_object_count(ddt_t *ddt, enum ddt_type type, enum ddt_class ddt_class)
211 ddt_object_count(ddt_t *ddt, enum ddt_type type, enum ddt_class class)
212 {
213     ASSERT(ddt_object_exists(ddt, type, ddt_class));
213     ASSERT(ddt_object_exists(ddt, type, class));

215     return (ddt_ops[type]->ddt_op_count(ddt->ddt_os,
216         ddt->ddt_object[type][ddt_class]));
216     ddt->ddt_object[type][class]));
217 }

219 int
220 ddt_object_info(ddt_t *ddt, enum ddt_type type, enum ddt_class ddt_class,
220 ddt_object_info(ddt_t *ddt, enum ddt_type type, enum ddt_class class,
221     dmu_object_info_t *doi)
222 {
223     if (!ddt_object_exists(ddt, type, ddt_class))
223     if (!ddt_object_exists(ddt, type, class))
224         return (SET_ERROR(ENOENT));

226     return (dmu_object_info(ddt->ddt_os, ddt->ddt_object[type][ddt_class],
226     return (dmu_object_info(ddt->ddt_os, ddt->ddt_object[type][class],
227         doi));
227 }
228 }

230 boolean_t

```

```

231 ddt_object_exists(ddt_t *ddt, enum ddt_type type, enum ddt_class ddt_class)
231 ddt_object_exists(ddt_t *ddt, enum ddt_type type, enum ddt_class class)
232 {
233     return (!ddt->ddt_object[type][ddt_class]);
233     return (!ddt->ddt_object[type][class]);
234 }

236 void
237 ddt_object_name(ddt_t *ddt, enum ddt_type type, enum ddt_class ddt_class,
237 ddt_object_name(ddt_t *ddt, enum ddt_type type, enum ddt_class class,
238     char *name)
239 {
240     (void) sprintf(name, DMU_POOL_DDT,
240         zio_checksum_table[ddt->ddt_checksum].ci_name,
241         ddt_ops[type]->ddt_op_name, ddt_class_name[ddt_class]);
242     ddt_ops[type]->ddt_op_name, ddt_class_name[class]);
242 }
    unchanged_portion_omitted_

```

```

*****
80615 Wed Apr 24 10:02:43 2013
new/usr/src/uts/common/fs/zfs/dsl_dataset.c
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectrallogic.com>
Submitted by: Will Andrews <willa@spectrallogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
_____unchanged_portion_omitted_____

2740 /*
2741  * Return (in *usedp) the amount of space written in new that is not
2742  * present in oldsnap. New may be a snapshot or the head. Old must be
2743  * a snapshot before new, in new's filesystem (or its origin). If not then
2744  * fail and return EINVAL.
2745  *
2746  * The written space is calculated by considering two components: First, we
2747  * ignore any freed space, and calculate the written as new's used space
2748  * minus old's used space. Next, we add in the amount of space that was freed
2749  * between the two snapshots, thus reducing new's used space relative to old's.
2750  * Specifically, this is the space that was born before old->ds_creation_txg,
2751  * and freed before new (ie. on new's deadlist or a previous deadlist).
2752  *
2753  * space freed
2754  * snapshots
2755  *
2756  */
2757 int
2758 dsl_dataset_space_written(dsl_dataset_t *oldsnap, dsl_dataset_t *newds,
2759 dsl_dataset_space_written(dsl_dataset_t *oldsnap, dsl_dataset_t *new,
2760 uint64_t *usedp, uint64_t *compp, uint64_t *uncompp)
2761 {
2762     int err = 0;
2763     uint64_t snapobj;
2764     dsl_pool_t *dp = newds->ds_dir->dd_pool;
2765     dsl_pool_t *dp = new->ds_dir->dd_pool;
2766
2767     ASSERT(dsl_pool_config_held(dp));
2768
2769     *usedp = 0;
2770     *usedp += newds->ds_phys->ds_referenced_bytes;
2771     *usedp += new->ds_phys->ds_referenced_bytes;
2772     *usedp -= oldsnap->ds_phys->ds_referenced_bytes;
2773
2774     *compp = 0;
2775     *compp += newds->ds_phys->ds_compressed_bytes;
2776     *compp += new->ds_phys->ds_compressed_bytes;
2777     *compp -= oldsnap->ds_phys->ds_compressed_bytes;
2778
2779     *uncompp = 0;
2780     *uncompp += newds->ds_phys->ds_uncompressed_bytes;
2781     *uncompp += new->ds_phys->ds_uncompressed_bytes;
2782     *uncompp -= oldsnap->ds_phys->ds_uncompressed_bytes;
2783
2784     snapobj = newds->ds_object;
2785     snapobj = new->ds_object;
2786     while (snapobj != oldsnap->ds_object) {
2787         dsl_dataset_t *snap;
2788         uint64_t used, comp, uncomp;
2789
2790         if (snapobj == newds->ds_object) {
2791             snap = newds;
2792         }
2793         if (snapobj == new->ds_object) {
2794             snap = new;

```

```

2786     } else {
2787         err = dsl_dataset_hold_obj(dp, snapobj, FTAG, &snap);
2788         if (err != 0)
2789             break;
2790     }
2791
2792     if (snap->ds_phys->ds_prev_snap_txg ==
2793         oldsnap->ds_phys->ds_creation_txg) {
2794         /*
2795          * The blocks in the deadlist can not be born after
2796          * ds_prev_snap_txg, so get the whole deadlist space,
2797          * which is more efficient (especially for old-format
2798          * deadlists). Unfortunately the deadlist code
2799          * doesn't have enough information to make this
2800          * optimization itself.
2801          */
2802         dsl_deadlist_space(&snap->ds_deadlist,
2803             &used, &comp, &uncomp);
2804     } else {
2805         dsl_deadlist_space_range(&snap->ds_deadlist,
2806             0, oldsnap->ds_phys->ds_creation_txg,
2807             &used, &comp, &uncomp);
2808     }
2809     *usedp += used;
2810     *compp += comp;
2811     *uncompp += uncomp;
2812
2813     /*
2814     * If we get to the beginning of the chain of snapshots
2815     * (ds_prev_snap_obj == 0) before oldsnap, then oldsnap
2816     * was not a snapshot of/before newds.
2817     * was not a snapshot of/before new.
2818     */
2819     snapobj = snap->ds_phys->ds_prev_snap_obj;
2820     if (snap != newds)
2821         if (snap != new)
2822             dsl_dataset_rele(snap, FTAG);
2823     if (snapobj == 0) {
2824         err = SET_ERROR(EINVAL);
2825         break;
2826     }
2827     return (err);
2828 }
_____unchanged_portion_omitted_____

```

```

*****
4325 Wed Apr 24 10:02:43 2013
new/usr/src/uts/common/fs/zfs/sys/arc.h
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectrallogic.com>
Submitted by: Will Andrews <willa@spectrallogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.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 */
21 /*
22 * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright (c) 2012 by Delphix. All rights reserved.
24 */

26 #ifndef _SYS_ARC_H
27 #define _SYS_ARC_H

29 #include <sys/zfs_context.h>

31 #ifdef __cplusplus
32 extern "C" {
33 #endif

35 #include <sys/zio.h>
36 #include <sys/dmu.h>
37 #include <sys/spa.h>

39 typedef struct arc_buf_hdr arc_buf_hdr_t;
40 typedef struct arc_buf arc_buf_t;
41 typedef void arc_done_func_t(zio_t *zio, arc_buf_t *buf, void *cb_private);
42 typedef int arc_evict_func_t(void *cb_private);
41 typedef void arc_done_func_t(zio_t *zio, arc_buf_t *buf, void *private);
42 typedef int arc_evict_func_t(void *private);

44 /* generic arc_done_func_t's which you can use */
45 arc_done_func_t arc_bcopy_func;
46 arc_done_func_t arc_getbuf_func;

48 struct arc_buf {
49     arc_buf_hdr_t      *b_hdr;
50     arc_buf_t          *b_next;
51     kmutex_t           b_evict_lock;
52     void               *b_data;
53     arc_evict_func_t   *b_efunc;
54     void               *b_private;

```

```

55 };
    unchanged_portion_omitted

82 void arc_space_consume(uint64_t space, arc_space_type_t type);
83 void arc_space_return(uint64_t space, arc_space_type_t type);
84 void *arc_data_buf_alloc(uint64_t space);
85 void arc_data_buf_free(void *buf, uint64_t space);
86 arc_buf_t *arc_buf_alloc(spa_t *spa, int size, void *tag,
87     arc_buf_contents_t type);
88 arc_buf_t *arc_loan_buf(spa_t *spa, int size);
89 void arc_return_buf(arc_buf_t *buf, void *tag);
90 void arc_loan_inuse_buf(arc_buf_t *buf, void *tag);
91 void arc_buf_add_ref(arc_buf_t *buf, void *tag);
92 boolean_t arc_buf_remove_ref(arc_buf_t *buf, void *tag);
93 int arc_buf_size(arc_buf_t *buf);
94 void arc_release(arc_buf_t *buf, void *tag);
95 int arc_released(arc_buf_t *buf);
96 int arc_has_callback(arc_buf_t *buf);
97 void arc_buf_freeze(arc_buf_t *buf);
98 void arc_buf_thaw(arc_buf_t *buf);
99 boolean_t arc_buf_eviction_needed(arc_buf_t *buf);
100 #ifdef ZFS_DEBUG
101 int arc_referenced(arc_buf_t *buf);
102 #endif

104 int arc_read(zio_t *pio, spa_t *spa, const blkptr_t *bp,
105     arc_done_func_t *done, void *cb_private, int priority, int flags,
106     arc_done_func_t *done, void *private, int priority, int flags,
107     uint32_t *arc_flags, const zbookmark_t *zb);
108 zio_t *arc_write(zio_t *pio, spa_t *spa, uint64_t txg,
109     blkptr_t *bp, arc_buf_t *buf, boolean_t l2arc, const zio_prop_t *zp,
110     arc_done_func_t *ready, arc_done_func_t *done, void *cb_private,
111     arc_done_func_t *ready, arc_done_func_t *done, void *private,
112     int priority, int zio_flags, const zbookmark_t *zb);

112 void arc_set_callback(arc_buf_t *buf, arc_evict_func_t *func,
113     void *cb_private);
112 void arc_set_callback(arc_buf_t *buf, arc_evict_func_t *func, void *private);
114 int arc_buf_evict(arc_buf_t *buf);

116 void arc_flush(spa_t *spa);
117 void arc_tempreserve_clear(uint64_t reserve);
118 int arc_tempreserve_space(uint64_t reserve, uint64_t txg);

120 void arc_init(void);
121 void arc_fini(void);

123 /*
124  * Level 2 ARC
125  */

127 void l2arc_add_vdev(spa_t *spa, vdev_t *vd);
128 void l2arc_remove_vdev(vdev_t *vd);
129 boolean_t l2arc_vdev_present(vdev_t *vd);
130 void l2arc_init(void);
131 void l2arc_fini(void);
132 void l2arc_start(void);
133 void l2arc_stop(void);

135 #ifndef _KERNEL
136 extern boolean_t arc_watch;
137 extern int arc_procfid;
138 #endif

140 #ifdef __cplusplus
141 }
    unchanged_portion_omitted

```



```

*****
7742 Wed Apr 24 10:02:43 2013
new/usr/src/uts/common/fs/zfs/sys/ddt.h
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectrallogic.com>
Submitted by: Will Andrews <willa@spectrallogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
_____unchanged_portion_omitted_____

```

```

169 #define DDT_NAMELEN      80

171 extern void ddt_object_name(ddt_t *ddt, enum ddt_type type,
172     enum ddt_class ddt_class, char *name);
172     enum ddt_class class, char *name);
173 extern int ddt_object_walk(ddt_t *ddt, enum ddt_type type,
174     enum ddt_class ddt_class, uint64_t *walk, ddt_entry_t *dde);
174     enum ddt_class class, uint64_t *walk, ddt_entry_t *dde);
175 extern uint64_t ddt_object_count(ddt_t *ddt, enum ddt_type type,
176     enum ddt_class ddt_class);
176     enum ddt_class class);
177 extern int ddt_object_info(ddt_t *ddt, enum ddt_type type,
178     enum ddt_class ddt_class, dmu_object_info_t *);
178     enum ddt_class class, dmu_object_info_t *);
179 extern boolean_t ddt_object_exists(ddt_t *ddt, enum ddt_type type,
180     enum ddt_class ddt_class);
180     enum ddt_class class);

182 extern void ddt_bp_fill(const ddt_phys_t *ddp, blkptr_t *bp,
183     uint64_t txg);
184 extern void ddt_bp_create(enum zio_checksum checksum, const ddt_key_t *ddk,
185     const ddt_phys_t *ddp, blkptr_t *bp);

187 extern void ddt_key_fill(ddt_key_t *ddk, const blkptr_t *bp);

189 extern void ddt_phys_fill(ddt_phys_t *ddp, const blkptr_t *bp);
190 extern void ddt_phys_clear(ddt_phys_t *ddp);
191 extern void ddt_phys_addrf(ddt_phys_t *ddp);
192 extern void ddt_phys_decref(ddt_phys_t *ddp);
193 extern void ddt_phys_free(ddt_t *ddt, ddt_key_t *ddk, ddt_phys_t *ddp,
194     uint64_t txg);
195 extern ddt_phys_t *ddt_phys_select(const ddt_entry_t *dde, const blkptr_t *bp);
196 extern uint64_t ddt_phys_total_refcnt(const ddt_entry_t *dde);

198 extern void ddt_stat_add(ddt_stat_t *dst, const ddt_stat_t *src, uint64_t neg);

200 extern void ddt_histogram_add(ddt_histogram_t *dst, const ddt_histogram_t *src);
201 extern void ddt_histogram_stat(ddt_stat_t *dds, const ddt_histogram_t *ddh);
202 extern boolean_t ddt_histogram_empty(const ddt_histogram_t *ddh);
203 extern void ddt_get_dedup_object_stats(spa_t *spa, ddt_object_t *ddo);
204 extern void ddt_get_dedup_histogram(spa_t *spa, ddt_histogram_t *ddh);
205 extern void ddt_get_dedup_stats(spa_t *spa, ddt_stat_t *dds_total);

207 extern uint64_t ddt_get_dedup_dspace(spa_t *spa);
208 extern uint64_t ddt_get_pool_dedup_ratio(spa_t *spa);

210 extern int ddt_ditto_copies_needed(ddt_t *ddt, ddt_entry_t *dde,
211     ddt_phys_t *ddp_willref);
212 extern int ddt_ditto_copies_present(ddt_entry_t *dde);

214 extern size_t ddt_compress(void *src, uchar_t *dst, size_t s_len, size_t d_len);
215 extern void ddt_decompress(uchar_t *src, void *dst, size_t s_len, size_t d_len);

217 extern ddt_t *ddt_select(spa_t *spa, const blkptr_t *bp);

```

```

218 extern void ddt_enter(ddt_t *ddt);
219 extern void ddt_exit(ddt_t *ddt);
220 extern ddt_entry_t *ddt_lookup(ddt_t *ddt, const blkptr_t *bp, boolean_t add);
221 extern void ddt_prefetch(spa_t *spa, const blkptr_t *bp);
222 extern void ddt_remove(ddt_t *ddt, ddt_entry_t *dde);

224 extern boolean_t ddt_class_contains(spa_t *spa, enum ddt_class max_class,
225     const blkptr_t *bp);

227 extern ddt_entry_t *ddt_repair_start(ddt_t *ddt, const blkptr_t *bp);
228 extern void ddt_repair_done(ddt_t *ddt, ddt_entry_t *dde);

230 extern int ddt_entry_compare(const void *x1, const void *x2);

232 extern void ddt_create(spa_t *spa);
233 extern int ddt_load(spa_t *spa);
234 extern void ddt_unload(spa_t *spa);
235 extern void ddt_sync(spa_t *spa, uint64_t txg);
236 extern int ddt_walk(spa_t *spa, ddt_bookmark_t *ddb, ddt_entry_t *dde);
237 extern int ddt_object_update(ddt_t *ddt, enum ddt_type type,
238     enum ddt_class ddt_class, ddt_entry_t *dde, dmu_tx_t *tx);
238     enum ddt_class class, ddt_entry_t *dde, dmu_tx_t *tx);

240 extern const ddt_ops_t ddt_zap_ops;

242 #ifdef __cplusplus
243 }
_____unchanged_portion_omitted_____

```

```

*****
10244 Wed Apr 24 10:02:44 2013
new/usr/src/uts/common/fs/zfs/sys/dsl_dataset.h
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectralogic.com>
Submitted by: Will Andrews <willa@spectralogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
_____unchanged_portion_omitted_____

166 /*
167  * The max length of a temporary tag prefix is the number of hex digits
168  * required to express UINIT64_MAX plus one for the hyphen.
169  */
170 #define MAX_TAG_PREFIX_LEN    17

172 #define dsl_dataset_is_snapshot(ds) \
173     ((ds)->ds_phys->ds_num_children != 0)

175 #define DS_UNIQUE_IS_ACCURATE(ds) \
176     (((ds)->ds_phys->ds_flags & DS_FLAG_UNIQUE_ACCURATE) != 0)

178 int dsl_dataset_hold(struct dsl_pool *dp, const char *name, void *tag,
179     dsl_dataset_t **dsp);
180 int dsl_dataset_hold_obj(struct dsl_pool *dp, uint64_t dsobj, void *tag,
181     dsl_dataset_t **);
182 void dsl_dataset_rele(dsl_dataset_t *ds, void *tag);
183 int dsl_dataset_own(struct dsl_pool *dp, const char *name,
184     void *tag, dsl_dataset_t **dsp);
185 int dsl_dataset_own_obj(struct dsl_pool *dp, uint64_t dsobj,
186     void *tag, dsl_dataset_t **dsp);
187 void dsl_dataset_disown(dsl_dataset_t *ds, void *tag);
188 void dsl_dataset_name(dsl_dataset_t *ds, char *name);
189 boolean_t dsl_dataset_tryown(dsl_dataset_t *ds, void *tag);
190 void dsl_register_onexit_hold_cleanup(dsl_dataset_t *ds, const char *htag,
191     minor_t minor);
192 uint64_t dsl_dataset_create_sync(dsl_dir_t *pds, const char *lastname,
193     dsl_dataset_t *origin, uint64_t flags, cred_t *, dmu_tx_t *);
194 uint64_t dsl_dataset_create_sync_dd(dsl_dir_t *dd, dsl_dataset_t *origin,
195     uint64_t flags, dmu_tx_t *tx);
196 int dsl_dataset_snapshot(nvlist_t *snaps, nvlist_t *props, nvlist_t *errors);
197 int dsl_dataset_promote(const char *name, char *conflsnap);
198 int dsl_dataset_clone_swap(dsl_dataset_t *clone, dsl_dataset_t *origin_head,
199     boolean_t force);
200 int dsl_dataset_rename_snapshot(const char *fsname,
201     const char *oldsnapname, const char *newsnapname, boolean_t recursive);
202 int dsl_dataset_snapshot_tmp(const char *fsname, const char *snapname,
203     minor_t cleanup_minor, const char *htag);

205 blkptr_t *dsl_dataset_get_blkptr(dsl_dataset_t *ds);
206 void dsl_dataset_set_blkptr(dsl_dataset_t *ds, blkptr_t *bp, dmu_tx_t *tx);

208 spa_t *dsl_dataset_get_spa(dsl_dataset_t *ds);

210 boolean_t dsl_dataset_modified_since_lastsnap(dsl_dataset_t *ds);

212 void dsl_dataset_sync(dsl_dataset_t *os, zio_t *zio, dmu_tx_t *tx);

214 void dsl_dataset_block_born(dsl_dataset_t *ds, const blkptr_t *bp,
215     dmu_tx_t *tx);
216 int dsl_dataset_block_kill(dsl_dataset_t *ds, const blkptr_t *bp,
217     dmu_tx_t *tx, boolean_t async);
218 boolean_t dsl_dataset_block_freeable(dsl_dataset_t *ds, const blkptr_t *bp,
219     uint64_t blk_birth);

```

```

220 uint64_t dsl_dataset_prev_snap_txg(dsl_dataset_t *ds);

222 void dsl_dataset_dirty(dsl_dataset_t *ds, dmu_tx_t *tx);
223 void dsl_dataset_stats(dsl_dataset_t *os, nvlist_t *nv);
224 void dsl_dataset_fast_stat(dsl_dataset_t *ds, dmu_objset_stats_t *stat);
225 void dsl_dataset_space(dsl_dataset_t *ds,
226     uint64_t *refdbytesp, uint64_t *availbytesp,
227     uint64_t *usedobjsp, uint64_t *availobjsp);
228 uint64_t dsl_dataset_fsid_guid(dsl_dataset_t *ds);
229 int dsl_dataset_space_written(dsl_dataset_t *oldsnap, dsl_dataset_t *newsd,
230     int dsl_dataset_space_written(dsl_dataset_t *oldsnap, dsl_dataset_t *new,
231     uint64_t *usedp, uint64_t *compp, uint64_t *uncompp);
232 int dsl_dataset_space_wouldfree(dsl_dataset_t *firstsnap, dsl_dataset_t *last,
233     uint64_t *usedp, uint64_t *compp, uint64_t *uncompp);
234 boolean_t dsl_dataset_is_dirty(dsl_dataset_t *ds);

235 int dsl_dsobj_to_dsname(char *pname, uint64_t obj, char *buf);

237 int dsl_dataset_check_quota(dsl_dataset_t *ds, boolean_t check_quota,
238     uint64_t asize, uint64_t inflight, uint64_t *used,
239     uint64_t *ref_rsrv);
240 int dsl_dataset_set_refquota(const char *dsname, zprop_source_t source,
241     uint64_t quota);
242 int dsl_dataset_set_refreservation(const char *dsname, zprop_source_t source,
243     uint64_t reservation);

245 boolean_t dsl_dataset_is_before(dsl_dataset_t *later, dsl_dataset_t *earlier);
246 void dsl_dataset_long_hold(dsl_dataset_t *ds, void *tag);
247 void dsl_dataset_long_rele(dsl_dataset_t *ds, void *tag);
248 boolean_t dsl_dataset_long_held(dsl_dataset_t *ds);

250 int dsl_dataset_clone_swap_check_impl(dsl_dataset_t *clone,
251     dsl_dataset_t *origin_head, boolean_t force);
252 void dsl_dataset_clone_swap_sync_impl(dsl_dataset_t *clone,
253     dsl_dataset_t *origin_head, dmu_tx_t *tx);
254 int dsl_dataset_snapshot_check_impl(dsl_dataset_t *ds, const char *snapname,
255     dmu_tx_t *tx);
256 void dsl_dataset_snapshot_sync_impl(dsl_dataset_t *ds, const char *snapname,
257     dmu_tx_t *tx);

259 void dsl_dataset_remove_from_next_clones(dsl_dataset_t *ds, uint64_t obj,
260     dmu_tx_t *tx);
261 void dsl_dataset_recalc_head_uniq(dsl_dataset_t *ds);
262 int dsl_dataset_get_snapname(dsl_dataset_t *ds);
263 int dsl_dataset_snap_lookup(dsl_dataset_t *ds, const char *name,
264     uint64_t *value);
265 int dsl_dataset_snap_remove(dsl_dataset_t *ds, const char *name, dmu_tx_t *tx);
266 void dsl_dataset_set_refreservation_sync_impl(dsl_dataset_t *ds,
267     zprop_source_t source, uint64_t value, dmu_tx_t *tx);
268 int dsl_dataset_rollback(const char *fsname);

270 #ifdef ZFS_DEBUG
271 #define dprintf_ds(ds, fmt, ...) do { \
272     if (zfs_flags & ZFS_DEBUG_DPRINTF) { \
273         char *ds_name = kmem_alloc(MAXNAMELEN, KM_SLEEP); \
274         dsl_dataset_name(ds, ds_name); \
275         dprintf("ds=%s " fmt, ds_name, __VA_ARGS__); \
276         kmem_free(ds_name, MAXNAMELEN); \
277     } \
278     _NOTE(CONSTCOND) } while (0)
279 #else
280 #define dprintf_ds(ds, fmt, ...)
281 #endif

283 #ifdef __cplusplus
284 }
_____unchanged_portion_omitted_____

```

new/usr/src/uts/common/fs/zfs/sys/spa.h

1

\*\*\*\*\*

25602 Wed Apr 24 10:02:44 2013

new/usr/src/uts/common/fs/zfs/sys/spa.h

3748 zfs headers should be C++ compatible

Submitted by: Justin Gibbs <justing@spectralllogic.com>

Submitted by: Will Andrews <willa@spectralllogic.com>

Reviewed by: Matthew Ahrens <mahrens@delphix.com>

Reviewed by: Eric Schrock <eric.schrock@delphix.com>

Reviewed by: Robert Mustacchi <rm@joyent.com>

\*\*\*\*\*

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_

```
559 extern spa_log_state_t spa_get_log_state(spa_t *spa);
560 extern void spa_set_log_state(spa_t *spa, spa_log_state_t state);
561 extern int spa_offline_log(spa_t *spa);
```

```
563 /* Log claim callback */
564 extern void spa_claim_notify(zio_t *zio);
```

```
566 /* Accessor functions */
567 extern boolean_t spa_shutting_down(spa_t *spa);
568 extern struct dsl_pool *spa_get_dsl(spa_t *spa);
569 extern boolean_t spa_is_initializing(spa_t *spa);
570 extern blkptr_t *spa_get_rootblkptr(spa_t *spa);
571 extern void spa_set_rootblkptr(spa_t *spa, const blkptr_t *bp);
572 extern void spa_altroot(spa_t *, char *, size_t);
573 extern int spa_sync_pass(spa_t *spa);
574 extern char *spa_name(spa_t *spa);
575 extern uint64_t spa_guid(spa_t *spa);
576 extern uint64_t spa_load_guid(spa_t *spa);
577 extern uint64_t spa_last_synced_txg(spa_t *spa);
578 extern uint64_t spa_first_txg(spa_t *spa);
579 extern uint64_t spa_syncing_txg(spa_t *spa);
580 extern uint64_t spa_version(spa_t *spa);
581 extern pool_state_t spa_state(spa_t *spa);
582 extern spa_load_state_t spa_load_state(spa_t *spa);
583 extern uint64_t spa_freeze_txg(spa_t *spa);
584 extern uint64_t spa_get_asize(spa_t *spa, uint64_t lsize);
585 extern uint64_t spa_get_dspace(spa_t *spa);
586 extern void spa_update_dspace(spa_t *spa);
587 extern uint64_t spa_version(spa_t *spa);
588 extern boolean_t spa_deflate(spa_t *spa);
589 extern metaslab_class_t *spa_normal_class(spa_t *spa);
590 extern metaslab_class_t *spa_log_class(spa_t *spa);
591 extern int spa_max_replication(spa_t *spa);
592 extern int spa_prev_software_version(spa_t *spa);
593 extern int spa_busy(void);
594 extern uint8_t spa_get_failmode(spa_t *spa);
595 extern boolean_t spa_suspended(spa_t *spa);
596 extern uint64_t spa_bootfs(spa_t *spa);
597 extern uint64_t spa_delegation(spa_t *spa);
598 extern objset_t *spa_meta_objset(spa_t *spa);
599 extern uint64_t spa_deadman_synctime(spa_t *spa);
```

```
601 /* Miscellaneous support routines */
602 extern void spa_activate_mos_feature(spa_t *spa, const char *feature);
603 extern void spa_deactivate_mos_feature(spa_t *spa, const char *feature);
604 extern int spa_rename(const char *oldname, const char *newname);
605 extern spa_t *spa_by_guid(uint64_t pool_guid, uint64_t device_guid);
606 extern boolean_t spa_guid_exists(uint64_t pool_guid, uint64_t device_guid);
607 extern char *spa_strdup(const char *);
608 extern void spa_strfree(char *);
609 extern uint64_t spa_get_random(uint64_t range);
610 extern uint64_t spa_generate_guid(spa_t *spa);
611 extern void sprintf_blkptr(char *buf, const blkptr_t *bp);
612 extern void spa_freeze(spa_t *spa);
```

new/usr/src/uts/common/fs/zfs/sys/spa.h

2

```
613 extern int spa_change_guid(spa_t *spa);
614 extern void spa_upgrade(spa_t *spa, uint64_t version);
615 extern void spa_evict_all(void);
616 extern vdev_t *spa_lookup_by_guid(spa_t *spa, uint64_t guid,
617     boolean_t l2cache);
618 extern boolean_t spa_has_spare(spa_t *, uint64_t guid);
619 extern uint64_t dva_get_dsize_sync(spa_t *spa, const dva_t *dva);
620 extern uint64_t bp_get_dsize_sync(spa_t *spa, const blkptr_t *bp);
621 extern uint64_t bp_get_dsize(spa_t *spa, const blkptr_t *bp);
622 extern boolean_t spa_has_slogs(spa_t *spa);
623 extern boolean_t spa_is_root(spa_t *spa);
624 extern boolean_t spa_writeable(spa_t *spa);

626 extern int spa_mode(spa_t *spa);
627 extern uint64_t strtonum(const char *str, char **nptr);

629 extern char *spa_his_ievent_table[];

631 extern void spa_history_create_obj(spa_t *spa, dmu_tx_t *tx);
632 extern int spa_history_get(spa_t *spa, uint64_t *offset, uint64_t *len_read,
633     char *his_buf);
634 extern int spa_history_log(spa_t *spa, const char *his_buf);
635 extern int spa_history_log_nvl(spa_t *spa, nvlist_t *nvl);
636 extern void spa_history_log_version(spa_t *spa, const char *operation);
637 extern void spa_history_log_internal(spa_t *spa, const char *operation,
638     dmu_tx_t *tx, const char *fmt, ...);
639 extern void spa_history_log_internal_ds(struct dsl_dataset *ds, const char *op,
640     dmu_tx_t *tx, const char *fmt, ...);
641 extern void spa_history_log_internal_dd(dsl_dir_t *dd, const char *operation,
642     dmu_tx_t *tx, const char *fmt, ...);

644 /* error handling */
645 struct zbookmark;
646 extern void spa_log_error(spa_t *spa, zio_t *zio);
647 extern void zfs_ereport_post(const char *subclass, spa_t *spa, vdev_t *vd,
648     zio_t *zio, uint64_t stateoffset, uint64_t length);
649 extern void zfs_post_remove(spa_t *spa, vdev_t *vd);
650 extern void zfs_post_state_change(spa_t *spa, vdev_t *vd);
651 extern void zfs_post_autoreplace(spa_t *spa, vdev_t *vd);
652 extern uint64_t spa_get_errlog_size(spa_t *spa);
653 extern int spa_get_errlog(spa_t *spa, void *uaddr, size_t *count);
654 extern void spa_errlog_rotate(spa_t *spa);
655 extern void spa_errlog_drain(spa_t *spa);
656 extern void spa_errlog_sync(spa_t *spa, uint64_t txg);
657 extern void spa_get_errlists(spa_t *spa, avl_tree_t *last, avl_tree_t *scrub);

659 /* vdev cache */
660 extern void vdev_cache_stat_init(void);
661 extern void vdev_cache_stat_fini(void);

663 /* Initialization and termination */
664 extern void spa_init(int flags);
665 extern void spa_fini(void);
666 extern void spa_boot_init();

668 /* properties */
669 extern int spa_prop_set(spa_t *spa, nvlist_t *nvp);
670 extern int spa_prop_get(spa_t *spa, nvlist_t **nvp);
671 extern void spa_prop_clear_bootfs(spa_t *spa, uint64_t obj, dmu_tx_t *tx);
672 extern void spa_configfile_set(spa_t *, nvlist_t *, boolean_t);

674 /* asynchronous event notification */
675 extern void spa_event_notify(spa_t *spa, vdev_t *vdev, const char *name);

677 #ifndef ZFS_DEBUG
```

```
678 #define dprintf_bp(bp, fmt, ...) do { \
679     if (zfs_flags & ZFS_DEBUG_DPRINTF) { \
680         char *__blkbuf = kmem_alloc(BP_SPRINTF_LEN, KM_SLEEP); \
681         sprintf_blkptr(__blkbuf, (bp)); \
682         dprintf(fmt " %s\n", __VA_ARGS__, __blkbuf); \
683         kmem_free(__blkbuf, BP_SPRINTF_LEN); \
684     } \
685     _NOTE(CONSTCOND) } while (0)
686 #else
687 #define dprintf_bp(bp, fmt, ...)
688 #endif

690 extern boolean_t spa_debug_enabled(spa_t *spa);
691 #define spa_dbgmsg(spa, ...) \
692 { \
693     if (spa_debug_enabled(spa)) \
694         zfs_dbgmsg(__VA_ARGS__); \
695 }
unchanged_portion_omitted
```

```

*****
10168 Wed Apr 24 10:02:44 2013
new/usr/src/uts/common/fs/zfs/sys/zfs_ioctl.h
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectrallogic.com>
Submitted by: Will Andrews <willa@spectrallogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
unchanged portion omitted

72 #define DMU_GET_STREAM_HDRTYPE(vi) BF64_GET((vi), 0, 2)
73 #define DMU_SET_STREAM_HDRTYPE(vi, x) BF64_SET((vi), 0, 2, x)

75 #define DMU_GET_FEATUREFLAGS(vi) BF64_GET((vi), 2, 30)
76 #define DMU_SET_FEATUREFLAGS(vi, x) BF64_SET((vi), 2, 30, x)

78 /*
79  * Feature flags for zfs send streams (flags in drr_versioninfo)
80  */

82 #define DMU_BACKUP_FEATURE_DEDUP (0x1)
83 #define DMU_BACKUP_FEATURE_DEDUPPROPS (0x2)
84 #define DMU_BACKUP_FEATURE_SA_SPILL (0x4)

86 /*
87  * Mask of all supported backup features
88  */
89 #define DMU_BACKUP_FEATURE_MASK (DMU_BACKUP_FEATURE_DEDUP | \
90  DMU_BACKUP_FEATURE_DEDUPPROPS | DMU_BACKUP_FEATURE_SA_SPILL)

92 /* Are all features in the given flag word currently supported? */
93 #define DMU_STREAM_SUPPORTED(x) (!(x) & ~DMU_BACKUP_FEATURE_MASK)

95 /*
96  * The drr_versioninfo field of the dmu_replay_record has the
97  * following layout:
98  *
99  *      64      56      48      40      32      24      16      8      0
100  *  +-----+-----+-----+-----+-----+-----+-----+-----+
101  *  |           reserved           | feature-flags |C|S|
102  *  +-----+-----+-----+-----+-----+-----+-----+
103  *
104  * The low order two bits indicate the header type: SUBSTREAM (0x1)
105  * or COMPOUNDSTREAM (0x2). Using two bits for this is historical:
106  * this field used to be a version number, where the two version types
107  * were 1 and 2. Using two bits for this allows earlier versions of
108  * the code to be able to recognize send streams that don't use any
109  * of the features indicated by feature flags.
110  */

95 #define DMU_BACKUP_MAGIC 0x2F5bacbacULL

97 #define DRR_FLAG_CLONE (1<<0)
98 #define DRR_FLAG_CI_DATA (1<<1)

100 /*
101  * flags in the drr_checksumflags field in the DRR_WRITE and
102  * DRR_WRITE_BYREF blocks
103  */
104 #define DRR_CHECKSUM_DEDUP (1<<0)

106 #define DRR_IS_DEDUP_CAPABLE(flags) ((flags) & DRR_CHECKSUM_DEDUP)

108 /*

```

```

109  * zfs ioctl command structure
110  */
111  enum drr_type {
128  typedef struct dmu_replay_record {
129      enum {
112          DRR_BEGIN, DRR_OBJECT, DRR_FREEOBJECTS,
113          DRR_WRITE, DRR_FREE, DRR_END, DRR_WRITE_BYREF,
114          DRR_SPILL, DRR_NUMTYPES
115  };

117  struct drr_begin {
133      } drr_type;
134      uint32_t drr_payloadlen;
135      union {
136          struct drr_begin {
118          uint64_t drr_magic;
119          /*
120          * Formerly named drr_version, this field has the following layout:
121          *
122          *      64      56      48      40      32      24      16      8      0
123          *  +-----+-----+-----+-----+-----+-----+-----+
124          *  |           reserved           | feature-flags |C|S|
125          *  +-----+-----+-----+-----+-----+-----+
126          *
127          * The low order two bits indicate the header type: SUBSTREAM (0x1)
128          * or COMPOUNDSTREAM (0x2). Using two bits for this is historical:
129          * this field used to be a version number, where the two version types
130          * were 1 and 2. Using two bits for this allows earlier versions of
131          * the code to be able to recognize send streams that don't use any
132          * of the features indicated by feature flags.
133          */
134          uint64_t drr_versioninfo;
138          uint64_t drr_versioninfo; /* was drr_version */
135          uint64_t drr_creation_time;
136          dmu_objset_type_t drr_type;
137          uint32_t drr_flags;
138          uint64_t drr_toguid;
139          uint64_t drr_fromguid;
140          char drr_toname[MAXNAMELEN];
141  };

143  struct drr_end {
145      } drr_begin;
146      struct drr_end {
144          zio_cksum_t drr_checksum;
145          uint64_t drr_toguid;
146  };

148  struct drr_object {
149      } drr_end;
150      struct drr_object {
149          uint64_t drr_object;
150          dmu_object_type_t drr_type;
151          dmu_object_type_t drr_bonustype;
152          uint32_t drr_blkisz;
153          uint32_t drr_bonuslen;
154          uint8_t drr_checksumtype;
155          uint8_t drr_compress;
156          uint8_t drr_pad[6];
157          uint64_t drr_toguid;
158          /* bonus content follows */
159  };

161  struct drr_freeobjects {
161      } drr_object;
162      struct drr_freeobjects {

```

```

162     uint64_t drr_firstobj;
163     uint64_t drr_numobjs;
164     uint64_t drr_toguid;
165 };

167 struct drr_write {
168     } drr_freeobjects;
169     struct drr_write {
170         uint64_t drr_object;
171         dmu_object_type_t drr_type;
172         uint32_t drr_pad;
173         uint64_t drr_offset;
174         uint64_t drr_length;
175         uint64_t drr_toguid;
176         uint8_t drr_checksumtype;
177         uint8_t drr_checksumflags;
178         uint8_t drr_pad2[6];
179         ddt_key_t drr_key; /* deduplication key */
180     /* content follows */
181 };

182 struct drr_free {
183     } drr_write;
184     struct drr_free {
185         uint64_t drr_object;
186         uint64_t drr_offset;
187         uint64_t drr_length;
188         uint64_t drr_toguid;
189 };

190 struct drr_write_byref {
191     uint64_t drr_object; /* where to put the data */
192     } drr_free;
193     struct drr_write_byref {
194         /* where to put the data */
195         uint64_t drr_object;
196         uint64_t drr_offset;
197         uint64_t drr_length;
198         uint64_t drr_toguid; /* where to find the prior copy of the data */
199         uint64_t drr_toguid;
200         /* where to find the prior copy of the data */
201         uint64_t drr_refguid;
202         uint64_t drr_refobject;
203         uint64_t drr_refoffset; /* properties of the data */
204         uint64_t drr_refoffset;
205         /* properties of the data */
206         uint8_t drr_checksumtype;
207         uint8_t drr_checksumflags;
208         uint8_t drr_pad2[6];
209         ddt_key_t drr_key; /* deduplication key */
210 };

211 struct drr_spill {
212     } drr_write_byref;
213     struct drr_spill {
214         uint64_t drr_object;
215         uint64_t drr_length;
216         uint64_t drr_toguid;
217         uint64_t drr_pad[4]; /* needed for crypto */
218     /* spill data follows */
219 };

220 typedef struct dmu_replay_record {
221     enum drr_type drr_type;
222     uint32_t drr_payloadlen;
223     union {

```

```

214     struct drr_begin drr_begin;
215     struct drr_end drr_end;
216     struct drr_object drr_object;
217     struct drr_freeobjects drr_freeobjects;
218     struct drr_write drr_write;
219     struct drr_free drr_free;
220     struct drr_write_byref drr_write_byref;
221     struct drr_spill drr_spill;
222     } drr_spill;
223 } dmu_replay_record_t;
224 unchanged_portion_omitted

```

```

*****
18003 Wed Apr 24 10:02:45 2013
new/usr/src/uts/common/fs/zfs/sys/zio.h
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectrallogic.com>
Submitted by: Will Andrews <willa@spectrallogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
unchanged portion omitted

444 extern zio_t *zio_null(zio_t *pio, spa_t *spa, vdev_t *vd,
445     zio_done_func_t *done, void *io_private, enum zio_flag flags);
445     zio_done_func_t *done, void *private, enum zio_flag flags);

447 extern zio_t *zio_root(spa_t *spa,
448     zio_done_func_t *done, void *io_private, enum zio_flag flags);
448     zio_done_func_t *done, void *private, enum zio_flag flags);

450 extern zio_t *zio_read(zio_t *pio, spa_t *spa, const blkptr_t *bp, void *data,
451     uint64_t size, zio_done_func_t *done, void *io_private,
451     uint64_t size, zio_done_func_t *done, void *private,
452     int priority, enum zio_flag flags, const zbookmark_t *zb);

454 extern zio_t *zio_write(zio_t *pio, spa_t *spa, uint64_t txg, blkptr_t *bp,
455     void *data, uint64_t size, const zio_prop_t *zp,
456     zio_done_func_t *ready, zio_done_func_t *done, void *io_private,
456     zio_done_func_t *ready, zio_done_func_t *done, void *private,
457     int priority, enum zio_flag flags, const zbookmark_t *zb);

459 extern zio_t *zio_rewrite(zio_t *pio, spa_t *spa, uint64_t txg, blkptr_t *bp,
460     void *data, uint64_t size, zio_done_func_t *done, void *io_private,
460     void *data, uint64_t size, zio_done_func_t *done, void *private,
461     int priority, enum zio_flag flags, zbookmark_t *zb);

463 extern void zio_write_override(zio_t *zio, blkptr_t *bp, int copies,
464     boolean_t nopwrite);

466 extern void zio_free(spa_t *spa, uint64_t txg, const blkptr_t *bp);

468 extern zio_t *zio_claim(zio_t *pio, spa_t *spa, uint64_t txg,
469     const blkptr_t *bp,
470     zio_done_func_t *done, void *io_private, enum zio_flag flags);
470     zio_done_func_t *done, void *private, enum zio_flag flags);

472 extern zio_t *zio_ioctl(zio_t *pio, spa_t *spa, vdev_t *vd, int cmd,
473     zio_done_func_t *done, void *io_private, int priority,
474     enum zio_flag flags);
473     zio_done_func_t *done, void *private, int priority, enum zio_flag flags);

476 extern zio_t *zio_read_phys(zio_t *pio, vdev_t *vd, uint64_t offset,
477     uint64_t size, void *data, int checksum,
478     zio_done_func_t *done, void *io_private, int priority, enum zio_flag flags,
477     zio_done_func_t *done, void *private, int priority, enum zio_flag flags,
479     boolean_t labels);

481 extern zio_t *zio_write_phys(zio_t *pio, vdev_t *vd, uint64_t offset,
482     uint64_t size, void *data, int checksum,
483     zio_done_func_t *done, void *io_private, int priority, enum zio_flag flags,
482     zio_done_func_t *done, void *private, int priority, enum zio_flag flags,
484     boolean_t labels);

486 extern zio_t *zio_free_sync(zio_t *pio, spa_t *spa, uint64_t txg,
487     const blkptr_t *bp, enum zio_flag flags);

```

```

489 extern int zio_alloc zil(spa_t *spa, uint64_t txg, blkptr_t *new_bp,
490     blkptr_t *old_bp, uint64_t size, boolean_t use_slog);
491 extern void zio_free zil(spa_t *spa, uint64_t txg, blkptr_t *bp);
492 extern void zio_flush(zio_t *zio, vdev_t *vd);
493 extern void zio_shrink(zio_t *zio, uint64_t size);

495 extern int zio_wait(zio_t *zio);
496 extern void zio_nowait(zio_t *zio);
497 extern void zio_execute(zio_t *zio);
498 extern void zio_interrupt(zio_t *zio);

500 extern zio_t *zio_walk_parents(zio_t *cio);
501 extern zio_t *zio_walk_children(zio_t *pio);
502 extern zio_t *zio_unique_parent(zio_t *cio);
503 extern void zio_add_child(zio_t *pio, zio_t *cio);

505 extern void *zio_buf_alloc(size_t size);
506 extern void zio_buf_free(void *buf, size_t size);
507 extern void *zio_data_buf_alloc(size_t size);
508 extern void zio_data_buf_free(void *buf, size_t size);

510 extern void zio_resubmit_stage_async(void *);

512 extern zio_t *zio_vdev_child_io(zio_t *zio, blkptr_t *bp, vdev_t *vd,
513     uint64_t offset, void *data, uint64_t size, int type, int priority,
514     enum zio_flag flags, zio_done_func_t *done, void *io_private);
513     enum zio_flag flags, zio_done_func_t *done, void *private);

516 extern zio_t *zio_vdev_delegated_io(vdev_t *vd, uint64_t offset,
517     void *data, uint64_t size, int type, int priority,
518     enum zio_flag flags, zio_done_func_t *done, void *io_private);
517     enum zio_flag flags, zio_done_func_t *done, void *private);

520 extern void zio_vdev_io_bypass(zio_t *zio);
521 extern void zio_vdev_io_reissue(zio_t *zio);
522 extern void zio_vdev_io_redone(zio_t *zio);

524 extern void zio_checksum_verified(zio_t *zio);
525 extern int zio_worst_error(int e1, int e2);

527 extern enum zio_checksum zio_checksum_select(enum zio_checksum child,
528     enum zio_checksum parent);
529 extern enum zio_checksum zio_checksum_dedup_select(spa_t *spa,
530     enum zio_checksum child, enum zio_checksum parent);
531 extern enum zio_compress zio_compress_select(enum zio_compress child,
532     enum zio_compress parent);

534 extern void zio_suspend(spa_t *spa, zio_t *zio);
535 extern int zio_resume(spa_t *spa);
536 extern void zio_resume_wait(spa_t *spa);

538 /*
539  * Initial setup and teardown.
540  */
541 extern void zio_init(void);
542 extern void zio_fini(void);

544 /*
545  * Fault injection
546  */
547 struct zinject_record;
548 extern uint32_t zio_injection_enabled;
549 extern int zio_inject_fault(char *name, int flags, int *id,
550     struct zinject_record *record);
551 extern int zio_inject_list_next(int *id, char *name, size_t buflen,
552     struct zinject_record *record);

```

```
553 extern int zio_clear_fault(int id);
554 extern void zio_handle_panic_injection(spa_t *spa, char *tag, uint64_t type);
555 extern int zio_handle_fault_injection(zio_t *zio, int error);
556 extern int zio_handle_device_injection(vdev_t *vd, zio_t *zio, int error);
557 extern int zio_handle_label_injection(zio_t *zio, int error);
558 extern void zio_handle_ignored_writes(zio_t *zio);
559 extern uint64_t zio_handle_io_delay(zio_t *zio);

561 /*
562  * Checksum ereport functions
563  */
564 extern void zfs_ereport_start_checksum(spa_t *spa, vdev_t *vd, struct zio *zio,
565     uint64_t offset, uint64_t length, void *arg, struct zio_bad_cksum *info);
566 extern void zfs_ereport_finish_checksum(zio_cksum_report_t *report,
567     const void *good_data, const void *bad_data, boolean_t drop_if_identical);

569 extern void zfs_ereport_send_interim_checksum(zio_cksum_report_t *report);
570 extern void zfs_ereport_free_checksum(zio_cksum_report_t *report);

572 /* If we have the good data in hand, this function can be used */
573 extern void zfs_ereport_post_checksum(spa_t *spa, vdev_t *vd,
574     struct zio *zio, uint64_t offset, uint64_t length,
575     const void *good_data, const void *bad_data, struct zio_bad_cksum *info);

577 /* Called from spa_sync(), but primarily an injection handler */
578 extern void spa_handle_ignored_writes(spa_t *spa);

580 /* zbookmark functions */
581 boolean_t zbookmark_is_before(const struct dnode_phys *dnp,
582     const zbookmark_t *zb1, const zbookmark_t *zb2);

584 #ifdef __cplusplus
585 }
_____ unchanged portion omitted
```



```

*****
89700 Wed Apr 24 10:02:45 2013
new/usr/src/uts/common/fs/zfs/zio.c
3748 zfs headers should be C++ compatible
Submitted by: Justin Gibbs <justing@spectrallogic.com>
Submitted by: Will Andrews <willa@spectrallogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
unchanged portion omitted

505 /*
506 * =====
507 * Create the various types of I/O (read, write, free, etc)
508 * =====
509 */
510 static zio_t *
511 zio_create(zio_t *pio, spa_t *spa, uint64_t txg, const blkptr_t *bp,
512 void *data, uint64_t size, zio_done_func_t *done, void *private,
513 zio_type_t type, int priority, enum zio_flag flags,
514 vdev_t *vd, uint64_t offset, const zbookmark_t *zb,
515 enum zio_stage stage, enum zio_stage pipeline)
516 {
517     zio_t *zio;

519     ASSERT3U(size, <=, SPA_MAXBLOCKSIZE);
520     ASSERT(P2PHASE(size, SPA_MINBLOCKSIZE) == 0);
521     ASSERT(P2PHASE(offset, SPA_MINBLOCKSIZE) == 0);

523     ASSERT(!vd || spa_config_held(spa, SCL_STATE_ALL, RW_READER));
524     ASSERT(!bp || !(flags & ZIO_FLAG_CONFIG_WRITER));
525     ASSERT(vd || stage == ZIO_STAGE_OPEN);

527     zio = kmem_cache_alloc(zio_cache, KM_SLEEP);
528     bzero(zio, sizeof (zio_t));

530     mutex_init(&zio->io_lock, NULL, MUTEX_DEFAULT, NULL);
531     cv_init(&zio->io_cv, NULL, CV_DEFAULT, NULL);

533     list_create(&zio->io_parent_list, sizeof (zio_link_t),
534         offsetof(zio_link_t, zl_parent_node));
535     list_create(&zio->io_child_list, sizeof (zio_link_t),
536         offsetof(zio_link_t, zl_child_node));

538     if (vd != NULL)
539         zio->io_child_type = ZIO_CHILD_VDEV;
540     else if (flags & ZIO_FLAG_GANG_CHILD)
541         zio->io_child_type = ZIO_CHILD_GANG;
542     else if (flags & ZIO_FLAG_DDT_CHILD)
543         zio->io_child_type = ZIO_CHILD_DDT;
544     else
545         zio->io_child_type = ZIO_CHILD_LOGICAL;

547     if (bp != NULL) {
548         zio->io_bp = (blkptr_t *)bp;
549         zio->io_bp_copy = *bp;
550         zio->io_bp_orig = *bp;
551         if (type != ZIO_TYPE_WRITE ||
552             zio->io_child_type == ZIO_CHILD_DDT)
553             zio->io_bp = &zio->io_bp_copy; /* so caller can free */
554         if (zio->io_child_type == ZIO_CHILD_LOGICAL)
555             zio->io_logical = zio;
556         if (zio->io_child_type > ZIO_CHILD_GANG && BP_IS_GANG(bp))
557             pipeline |= ZIO_GANG_STAGES;

```

```

558     }

560     zio->io_spa = spa;
561     zio->io_txg = txg;
562     zio->io_done = done;
563     zio->io_private = io_private;
564     zio->io_private = private;
565     zio->io_type = type;
566     zio->io_priority = priority;
567     zio->io_vd = vd;
568     zio->io_offset = offset;
569     zio->io_orig_data = zio->io_data = data;
570     zio->io_orig_size = zio->io_size = size;
571     zio->io_orig_flags = zio->io_flags = flags;
572     zio->io_orig_stage = zio->io_stage = stage;
573     zio->io_orig_pipeline = zio->io_pipeline = pipeline;

574     zio->io_state[ZIO_WAIT_READY] = (stage >= ZIO_STAGE_READY);
575     zio->io_state[ZIO_WAIT_DONE] = (stage >= ZIO_STAGE_DONE);

577     if (zb != NULL)
578         zio->io_bookmark = *zb;

580     if (pio != NULL) {
581         if (zio->io_logical == NULL)
582             zio->io_logical = pio->io_logical;
583         if (zio->io_child_type == ZIO_CHILD_GANG)
584             zio->io_gang_leader = pio->io_gang_leader;
585         zio_add_child(pio, zio);
586     }

588     return (zio);
589 }
unchanged portion omitted

601 zio_t *
602 zio_null(zio_t *pio, spa_t *spa, vdev_t *vd, zio_done_func_t *done,
603 void *private, enum zio_flag flags)
604 {
605     zio_t *zio;

607     zio = zio_create(pio, spa, 0, NULL, NULL, 0, done, io_private,
608         ZIO_TYPE_NULL, ZIO_PRIORITY_NOW, flags, vd, 0, NULL,
609         ZIO_STAGE_OPEN, ZIO_INTERLOCK_PIPELINE);

611     return (zio);
612 }

614 zio_t *
615 zio_root(spa_t *spa, zio_done_func_t *done, void *private,
616     enum zio_flag flags)
617 {
618     return (zio_null(NULL, spa, NULL, done, io_private, flags));
619 }

621 zio_t *
622 zio_read(zio_t *pio, spa_t *spa, const blkptr_t *bp,
623 void *data, uint64_t size, zio_done_func_t *done, void *private,
624 void *data, uint64_t size, zio_done_func_t *done, void *private,
625 int priority, enum zio_flag flags, const zbookmark_t *zb)
626     zio_t *zio;

```

```

628     zio = zio_create(pio, spa, BP_PHYSICAL_BIRTH(bp), bp,
629     data, size, done, io_private,
628     data, size, done, private,
630     ZIO_TYPE_READ, priority, flags, NULL, 0, zb,
631     ZIO_STAGE_OPEN, (flags & ZIO_FLAG_DDT_CHILD) ?
632     ZIO_DDT_CHILD_READ_PIPELINE : ZIO_READ_PIPELINE);

634     return (zio);
635 }

637 zio_t *
638 zio_write(zio_t *pio, spa_t *spa, uint64_t txg, blkptr_t *bp,
639     void *data, uint64_t size, const zio_prop_t *zp,
640     zio_done_func_t *ready, zio_done_func_t *done, void *io_private,
639     zio_done_func_t *ready, zio_done_func_t *done, void *private,
641     int priority, enum zio_flag flags, const zbookmark_t *zb)
642 {
643     zio_t *zio;

645     ASSERT(zp->zp_checksum >= ZIO_CHECKSUM_OFF &&
646     zp->zp_checksum < ZIO_CHECKSUM_FUNCTIONS &&
647     zp->zp_compress >= ZIO_COMPRESS_OFF &&
648     zp->zp_compress < ZIO_COMPRESS_FUNCTIONS &&
649     DMU_OT_IS_VALID(zp->zp_type) &&
650     zp->zp_level < 32 &&
651     zp->zp_copies > 0 &&
652     zp->zp_copies <= spa_max_replication(spa));

654     zio = zio_create(pio, spa, txg, bp, data, size, done, io_private,
653     zio = zio_create(pio, spa, txg, bp, data, size, done, private,
655     ZIO_TYPE_WRITE, priority, flags, NULL, 0, zb,
656     ZIO_STAGE_OPEN, (flags & ZIO_FLAG_DDT_CHILD) ?
657     ZIO_DDT_CHILD_WRITE_PIPELINE : ZIO_WRITE_PIPELINE);

659     zio->io_ready = ready;
660     zio->io_prop = *zp;

662     return (zio);
663 }

665 zio_t *
666 zio_rewrite(zio_t *pio, spa_t *spa, uint64_t txg, blkptr_t *bp, void *data,
667     uint64_t size, zio_done_func_t *done, void *io_private, int priority,
666     uint64_t size, zio_done_func_t *done, void *private, int priority,
668     enum zio_flag flags, zbookmark_t *zb)
669 {
670     zio_t *zio;

672     zio = zio_create(pio, spa, txg, bp, data, size, done, io_private,
671     zio = zio_create(pio, spa, txg, bp, data, size, done, private,
673     ZIO_TYPE_WRITE, priority, flags, NULL, 0, zb,
674     ZIO_STAGE_OPEN, ZIO_REWRITE_PIPELINE);

676     return (zio);
677 }
unchanged_portion_omitted

727 zio_t *
728 zio_claim(zio_t *pio, spa_t *spa, uint64_t txg, const blkptr_t *bp,
729     zio_done_func_t *done, void *io_private, enum zio_flag flags)
728     zio_done_func_t *done, void *private, enum zio_flag flags)
730 {
731     zio_t *zio;

733     /*

```

```

734     * A claim is an allocation of a specific block. Claims are needed
735     * to support immediate writes in the intent log. The issue is that
736     * immediate writes contain committed data, but in a txg that was
737     * *not* committed. Upon opening the pool after an unclean shutdown,
738     * the intent log claims all blocks that contain immediate write data
739     * so that the SPA knows they're in use.
740     *
741     * All claims *must* be resolved in the first txg -- before the SPA
742     * starts allocating blocks -- so that nothing is allocated twice.
743     * If txg == 0 we just verify that the block is claimable.
744     */
745     ASSERT3U(spa->spa_uberblock.ub_rootbp.blk_birth, <, spa_first_txg(spa));
746     ASSERT(txg == spa_first_txg(spa) || txg == 0);
747     ASSERT(!BP_GET_DEDUP(bp) || !spa_writeable(spa)); /* zdb(1M) */

749     zio = zio_create(pio, spa, txg, bp, NULL, BP_GET_PSIZE(bp),
750     done, io_private, ZIO_TYPE_CLAIM, ZIO_PRIORITY_NOW, flags,
749     done, private, ZIO_TYPE_CLAIM, ZIO_PRIORITY_NOW, flags,
751     NULL, 0, NULL, ZIO_STAGE_OPEN, ZIO_CLAIM_PIPELINE);

753     return (zio);
754 }

756 zio_t *
757 zio_ioctl(zio_t *pio, spa_t *spa, vdev_t *vd, int cmd,
758     zio_done_func_t *done, void *io_private, int priority, enum zio_flag flags)
757     zio_done_func_t *done, void *private, int priority, enum zio_flag flags)
759 {
760     zio_t *zio;
761     int c;

763     if (vd->vdev_children == 0) {
764         zio = zio_create(pio, spa, 0, NULL, NULL, 0, done, io_private,
763         zio = zio_create(pio, spa, 0, NULL, NULL, 0, done, private,
765         ZIO_TYPE_IOCTL, priority, flags, vd, 0, NULL,
766         ZIO_STAGE_OPEN, ZIO_IOCTL_PIPELINE);

768         zio->io_cmd = cmd;
769     } else {
770         zio = zio_null(pio, spa, NULL, NULL, NULL, flags);

772         for (c = 0; c < vd->vdev_children; c++)
773             zio_nowait(zio_ioctl(zio, spa, vd->vdev_child[c], cmd,
774                 done, io_private, priority, flags));
773                 done, private, priority, flags));
775     }

777     return (zio);
778 }

780 zio_t *
781 zio_read_phys(zio_t *pio, vdev_t *vd, uint64_t offset, uint64_t size,
782     void *data, int checksum, zio_done_func_t *done, void *io_private,
781     void *data, int checksum, zio_done_func_t *done, void *private,
783     int priority, enum zio_flag flags, boolean_t labels)
784 {
785     zio_t *zio;

787     ASSERT(vd->vdev_children == 0);
788     ASSERT(!labels || offset + size <= VDEV_LABEL_START_SIZE ||
789         offset >= vd->vdev_psize - VDEV_LABEL_END_SIZE);
790     ASSERT3U(offset + size, <=, vd->vdev_psize);

792     zio = zio_create(pio, vd->vdev_spa, 0, NULL, data, size, done,
793     io_private, ZIO_TYPE_READ, priority, flags, vd, offset, NULL,
791     zio = zio_create(pio, vd->vdev_spa, 0, NULL, data, size, done, private,

```

```

792     ZIO_TYPE_READ, priority, flags, vd, offset, NULL,
794     ZIO_STAGE_OPEN, ZIO_READ_PHYS_PIPELINE);

796     zio->io_prop.zp_checksum = checksum;

798     return (zio);
799 }

801 zio_t *
802 zio_write_phys(zio_t *pio, vdev_t *vd, uint64_t offset, uint64_t size,
803 void *data, int checksum, zio_done_func_t *done, void *io_private,
804 void *data, int checksum, zio_done_func_t *done, void *private,
805 int priority, enum zio_flag flags, boolean_t labels)
806 {
807     zio_t *zio;

808     ASSERT(vd->vdev_children == 0);
809     ASSERT(!labels || offset + size <= VDEV_LABEL_START_SIZE ||
810         offset >= vd->vdev_psize - VDEV_LABEL_END_SIZE);
811     ASSERT3U(offset + size, <=, vd->vdev_psize);

813     zio = zio_create(pio, vd->vdev_spa, 0, NULL, data, size, done,
814         io_private, ZIO_TYPE_WRITE, priority, flags, vd, offset, NULL,
815         ZIO_TYPE_WRITE, priority, flags, vd, offset, NULL,
816         ZIO_STAGE_OPEN, ZIO_WRITE_PHYS_PIPELINE);

817     zio->io_prop.zp_checksum = checksum;

819     if (zio_checksum_table[checksum].ci_eck) {
820         /*
821          * zec checksums are necessarily destructive -- they modify
822          * the end of the write buffer to hold the verifier/checksum.
823          * Therefore, we must make a local copy in case the data is
824          * being written to multiple places in parallel.
825          */
826         void *wbuf = zio_buf_alloc(size);
827         bcopy(data, wbuf, size);
828         zio_push_transform(zio, wbuf, size, size, NULL);
829     }

831     return (zio);
832 }

834 /*
835  * Create a child I/O to do some work for us.
836  */
837 zio_t *
838 zio_vdev_child_io(zio_t *pio, blkptr_t *bp, vdev_t *vd, uint64_t offset,
839 void *data, uint64_t size, int type, int priority, enum zio_flag flags,
840 zio_done_func_t *done, void *io_private)
841 {
842     enum zio_stage pipeline = ZIO_VDEV_CHILD_PIPELINE;
843     zio_t *zio;

845     ASSERT(vd->vdev_parent ==
846         (pio->io_vd ? pio->io_vd : pio->io_spa->spa_root_vdev));

848     if (type == ZIO_TYPE_READ && bp != NULL) {
849         /*
850          * If we have the bp, then the child should perform the
851          * checksum and the parent need not. This pushes error
852          * detection as close to the leaves as possible and
853          * eliminates redundant checksums in the interior nodes.
854          */

```

```

855         pipeline |= ZIO_STAGE_CHECKSUM_VERIFY;
856         pio->io_pipeline &= ~ZIO_STAGE_CHECKSUM_VERIFY;
857     }

859     if (vd->vdev_children == 0)
860         offset += VDEV_LABEL_START_SIZE;

862     flags |= ZIO_VDEV_CHILD_FLAGS(pio) | ZIO_FLAG_DONT_PROPAGATE;

864     /*
865      * If we've decided to do a repair, the write is not speculative --
866      * even if the original read was.
867      */
868     if (flags & ZIO_FLAG_IO_REPAIR)
869         flags &= ~ZIO_FLAG_SPECULATIVE;

871     zio = zio_create(pio, pio->io_spa, pio->io_txg, bp, data, size,
872         done, io_private, type, priority, flags, vd, offset,
873         &pio->io_bookmark, ZIO_STAGE_VDEV_IO_START >> 1, pipeline);
874     zio->io_bookmark->bookmark = &pio->io_bookmark;

875     return (zio);
876 }

878 zio_t *
879 zio_vdev_delegated_io(vdev_t *vd, uint64_t offset, void *data, uint64_t size,
880 int type, int priority, enum zio_flag flags,
881 zio_done_func_t *done, void *io_private)
882 {
883     zio_t *zio;

885     ASSERT(vd->vdev_ops->vdev_op_leaf);

887     zio = zio_create(NULL, vd->vdev_spa, 0, NULL,
888         data, size, done, io_private, type, priority,
889         data, size, done, private, type, priority,
890         flags | ZIO_FLAG_CANFAIL | ZIO_FLAG_DONT_RETRY,
891         vd, offset, NULL,
892         ZIO_STAGE_VDEV_IO_START >> 1, ZIO_VDEV_CHILD_PIPELINE);

893     return (zio);
894 }

```

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_