```
**********************************************************
   12122 Mon Jun 17 22:40:14 2013
new/usr/src/lib/libzfs/common/libzfs_status.c
3818 zpool status -x should report pools with removed l2arc devices
Reviewed by: Saso Kiselkov <skiselkov.ml@gmail.com>
Reviewed by: George Wilson <gwilson@zfsmail.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) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2013 Steven Hartland. All rights reserved.
#endif /* ! codereview */
 */

/*
 * This file contains the functions which analyze the status of a pool.  This
 * include both the status of an active pool, as well as the status exported
 * pools.  Returns one of the ZPOOL_STATUS_* defines describing the status of
 * the pool.  This status is independent (to a certain degree) from the state of
 * the pool.  A pool's state describes only whether or not it is capable of
 * providing the necessary fault tolerance for data.  The status describes the
 * overall status of devices.  A pool that is online can still have a device
 * that is experiencing errors.
 *
 * Only a subset of the possible faults can be detected using 'zpool status',
 * and not all possible errors correspond to a FMA message ID.  The explanation
 * is left up to the caller, depending on whether it is a live pool or an
 * import.
 */

#include <libzfs.h>
#include <string.h>
#include <unistd.h>
#include "libzfs_impl.h"
#include "zfeature_common.h"

/*
 * Message ID table.  This must be kept in sync with the ZPOOL_STATUS_* defines
 * in libzfs.h.  Note that there are some status results which go past the end
 * of this table, and hence have no associated message ID.
 */
static char *zfs_msgid_table[] = {
        "ZFS-8000-14",
        "ZFS-8000-2Q",
        "ZFS-8000-3C",
```



```
        "ZFS-8000-4J",
        "ZFS-8000-5E",
        "ZFS-8000-6X",
        "ZFS-8000-72",
        "ZFS-8000-8A",
        "ZFS-8000-9P",
        "ZFS-8000-A5",
        "ZFS-8000-EY",
        "ZFS-8000-HC",
        "ZFS-8000-JQ",
        "ZFS-8000-K4",
};

#define NMSGID  (sizeof (zfs_msgid_table) / sizeof (zfs_msgid_table[0]))

/* ARGSUSED */
static int
vdev_missing(uint64_t state, uint64_t aux, uint64_t errs)
{
        return (state == VDEV_STATE_CANT_OPEN &&
            aux == VDEV_AUX_OPEN_FAILED);
}

/* ARGSUSED */
static int
vdev_faulted(uint64_t state, uint64_t aux, uint64_t errs)
{
        return (state == VDEV_STATE_FAULTED);
}

/* ARGSUSED */
static int
vdev_errors(uint64_t state, uint64_t aux, uint64_t errs)
{
        return (state == VDEV_STATE_DEGRADED || errs != 0);
}

/* ARGSUSED */
static int
vdev_broken(uint64_t state, uint64_t aux, uint64_t errs)
{
        return (state == VDEV_STATE_CANT_OPEN);
}

/* ARGSUSED */
static int
vdev_offlined(uint64_t state, uint64_t aux, uint64_t errs)
{
        return (state == VDEV_STATE_OFFLINE);
}

/* ARGSUSED */
static int
vdev_removed(uint64_t state, uint64_t aux, uint64_t errs)
{
        return (state == VDEV_STATE_REMOVED);
}

/*
 * Detect if any leaf devices that have seen errors or could not be opened.
 */
static boolean_t
find_vdev_problem(nvlist_t *vdev, int (*func)(uint64_t, uint64_t, uint64_t))
{
        nvlist_t **child;
        vdev_stat_t *vs;
```

```
126         uint_t c, children;
127         char *type;

129         /*
130          * Ignore problems within a 'replacing' vdev, since we're presumably in
131          * the process of repairing any such errors, and don't want to call them
132          * out again.  We'll pick up the fact that a resilver is happening
133          * later.
134          */
135         verify(nvlist_lookup_string(vdev, ZPOOL_CONFIG_TYPE, &type) == 0);
136         if (strcmp(type, VDEV_TYPE_REPLACING) == 0)
137                 return (B_FALSE);

139         if (nvlist_lookup_nvlist_array(vdev, ZPOOL_CONFIG_CHILDREN, &child,
140             &children) == 0) {
141                 for (c = 0; c < children; c++)
142                         if (find_vdev_problem(child[c], func))
143                                 return (B_TRUE);
144         } else {
145                 verify(nvlist_lookup_uint64_array(vdev, ZPOOL_CONFIG_VDEV_STATS,
146                     (uint64_t **)&vs, &c) == 0);

148                 if (func(vs->vs_state, vs->vs_aux,
149                     vs->vs_read_errors +
150                     vs->vs_write_errors +
151                     vs->vs_checksum_errors))
152                         return (B_TRUE);
153         }

155         /*
156          * Check any L2 cache devs
157          */
158         if (nvlist_lookup_nvlist_array(vdev, ZPOOL_CONFIG_L2CACHE, &child,
159             &children) == 0) {
160                 for (c = 0; c < children; c++)
161                         if (find_vdev_problem(child[c], func))
162                                 return (B_TRUE);
163         }

165 #endif /* ! codereview */
166         return (B_FALSE);
167 }

169 /*
170  * Active pool health status.
171  *
172  * To determine the status for a pool, we make several passes over the config,
173  * picking the most egregious error we find.  In order of importance, we do the
174  * following:
175  *
176  *      - Check for a complete and valid configuration
177  *      - Look for any faulted or missing devices in a non-replicated config
178  *      - Check for any data errors
179  *      - Check for any faulted or missing devices in a replicated config
180  *      - Look for any devices showing errors
181  *      - Check for any resilvering devices
182  *
183  * There can obviously be multiple errors within a single pool, so this routine
184  * only picks the most damaging of all the current errors to report.
185  */
186 static zpool_status_t
187 check_status(nvlist_t *config, boolean_t isimport)
188 {
189         nvlist_t *nvroot;
190         vdev_stat_t *vs;
191         pool_scan_stat_t *ps = NULL;
```



```
192         uint_t vsc, psc;
193         uint64_t nerr;
194         uint64_t version;
195         uint64_t stateval;
196         uint64_t suspended;
197         uint64_t hostid = 0;

199         verify(nvlist_lookup_uint64(config, ZPOOL_CONFIG_VERSION,
200             &version) == 0);
201         verify(nvlist_lookup_nvlist(config, ZPOOL_CONFIG_VDEV_TREE,
202             &nvroot) == 0);
203         verify(nvlist_lookup_uint64_array(nvroot, ZPOOL_CONFIG_VDEV_STATS,
204             (uint64_t **)&vs, &vsc) == 0);
205         verify(nvlist_lookup_uint64(config, ZPOOL_CONFIG_POOL_STATE,
206             &stateval) == 0);

208         /*
209          * Currently resilvering a vdev
210          */
211         (void) nvlist_lookup_uint64_array(nvroot, ZPOOL_CONFIG_SCAN_STATS,
212             (uint64_t **)&ps, &psc);
213         if (ps && ps->pss_func == POOL_SCAN_RESILVER &&
214             ps->pss_state == DSS_SCANNING)
215                 return (ZPOOL_STATUS_RESILVERING);

217         /*
218          * Pool last accessed by another system.
219          */
220         (void) nvlist_lookup_uint64(config, ZPOOL_CONFIG_HOSTID, &hostid);
221         if (hostid != 0 && (unsigned long)hostid != gethostid() &&
222             stateval == POOL_STATE_ACTIVE)
223                 return (ZPOOL_STATUS_HOSTID_MISMATCH);

225         /*
226          * Newer on-disk version.
227          */
228         if (vs->vs_state == VDEV_STATE_CANT_OPEN &&
229             vs->vs_aux == VDEV_AUX_VERSION_NEWER)
230                 return (ZPOOL_STATUS_VERSION_NEWER);

232         /*
233          * Unsupported feature(s).
234          */
235         if (vs->vs_state == VDEV_STATE_CANT_OPEN &&
236             vs->vs_aux == VDEV_AUX_UNSUP_FEAT) {
237                 nvlist_t *nvinfo;

239                 verify(nvlist_lookup_nvlist(config, ZPOOL_CONFIG_LOAD_INFO,
240                     &nvinfo) == 0);
241                 if (nvlist_exists(nvinfo, ZPOOL_CONFIG_CAN_RDONLY))
242                         return (ZPOOL_STATUS_UNSUP_FEAT_WRITE);
243                 return (ZPOOL_STATUS_UNSUP_FEAT_READ);
244         }

246         /*
247          * Check that the config is complete.
248          */
249         if (vs->vs_state == VDEV_STATE_CANT_OPEN &&
250             vs->vs_aux == VDEV_AUX_BAD_GUID_SUM)
251                 return (ZPOOL_STATUS_BAD_GUID_SUM);

253         /*
254          * Check whether the pool has suspended due to failed I/O.
255          */
256         if (nvlist_lookup_uint64(config, ZPOOL_CONFIG_SUSPENDED,
257             &suspended) == 0) {
```

```
258                 if (suspended == ZIO_FAILURE_MODE_CONTINUE)
259                         return (ZPOOL_STATUS_IO_FAILURE_CONTINUE);
260                 return (ZPOOL_STATUS_IO_FAILURE_WAIT);
261         }

263         /*
264          * Could not read a log.
265          */
266         if (vs->vs_state == VDEV_STATE_CANT_OPEN &&
267             vs->vs_aux == VDEV_AUX_BAD_LOG) {
268                 return (ZPOOL_STATUS_BAD_LOG);
269         }

271         /*
272          * Bad devices in non-replicated config.
273          */
274         if (vs->vs_state == VDEV_STATE_CANT_OPEN &&
275             find_vdev_problem(nvroot, vdev_faulted))
276                 return (ZPOOL_STATUS_FAULTED_DEV_NR);

278         if (vs->vs_state == VDEV_STATE_CANT_OPEN &&
279             find_vdev_problem(nvroot, vdev_missing))
280                 return (ZPOOL_STATUS_MISSING_DEV_NR);

282         if (vs->vs_state == VDEV_STATE_CANT_OPEN &&
283             find_vdev_problem(nvroot, vdev_broken))
284                 return (ZPOOL_STATUS_CORRUPT_LABEL_NR);

286         /*
287          * Corrupted pool metadata
288          */
289         if (vs->vs_state == VDEV_STATE_CANT_OPEN &&
290             vs->vs_aux == VDEV_AUX_CORRUPT_DATA)
291                 return (ZPOOL_STATUS_CORRUPT_POOL);

293         /*
294          * Persistent data errors.
295          */
296         if (!isimport) {
297                 if (nvlist_lookup_uint64(config, ZPOOL_CONFIG_ERRCOUNT,
298                     &nerr) == 0 && nerr != 0)
299                         return (ZPOOL_STATUS_CORRUPT_DATA);
300         }

302         /*
303          * Missing devices in a replicated config.
304          */
305         if (find_vdev_problem(nvroot, vdev_faulted))
306                 return (ZPOOL_STATUS_FAULTED_DEV_R);
307         if (find_vdev_problem(nvroot, vdev_missing))
308                 return (ZPOOL_STATUS_MISSING_DEV_R);
309         if (find_vdev_problem(nvroot, vdev_broken))
310                 return (ZPOOL_STATUS_CORRUPT_LABEL_R);

312         /*
313          * Devices with errors
314          */
315         if (!isimport && find_vdev_problem(nvroot, vdev_errors))
316                 return (ZPOOL_STATUS_FAILING_DEV);

318         /*
319          * Offlined devices
320          */
321         if (find_vdev_problem(nvroot, vdev_offlined))
322                 return (ZPOOL_STATUS_OFFLINE_DEV);
```



```
324         /*
325          * Removed device
326          */
327         if (find_vdev_problem(nvroot, vdev_removed))
328                 return (ZPOOL_STATUS_REMOVED_DEV);

330         /*
331          * Outdated, but usable, version
332          */
333         if (SPA_VERSION_IS_SUPPORTED(version) && version != SPA_VERSION)
334                 return (ZPOOL_STATUS_VERSION_OLDER);

336         /*
337          * Usable pool with disabled features
338          */
339         if (version >= SPA_VERSION_FEATURES) {
340                 int i;
341                 nvlist_t *feat;

343                 if (isimport) {
344                         feat = fnvlist_lookup_nvlist(config,
345                             ZPOOL_CONFIG_LOAD_INFO);
346                         feat = fnvlist_lookup_nvlist(feat,
347                             ZPOOL_CONFIG_ENABLED_FEAT);
348                 } else {
349                         feat = fnvlist_lookup_nvlist(config,
350                             ZPOOL_CONFIG_FEATURE_STATS);
351                 }

353                 for (i = 0; i < SPA_FEATURES; i++) {
354                         zfeature_info_t *fi = &spa_feature_table[i];
355                         if (!nvlist_exists(feat, fi->fi_guid))
356                                 return (ZPOOL_STATUS_FEAT_DISABLED);
357                 }
358         }

360         return (ZPOOL_STATUS_OK);
361 }

363 zpool_status_t
364 zpool_get_status(zpool_handle_t *zhp, char **msgid)
365 {
366         zpool_status_t ret = check_status(zhp->zpool_config, B_FALSE);

368         if (ret >= NMSGID)
369                 *msgid = NULL;
370         else
371                 *msgid = zfs_msgid_table[ret];

373         return (ret);
374 }

376 zpool_status_t
377 zpool_import_status(nvlist_t *config, char **msgid)
378 {
379         zpool_status_t ret = check_status(config, B_TRUE);

381         if (ret >= NMSGID)
382                 *msgid = NULL;
383         else
384                 *msgid = zfs_msgid_table[ret];

386         return (ret);
387 }

389 static void
```

```
390 dump_ddt_stat(const ddt_stat_t *dds, int h)
391 {
392         char refcnt[6];
393         char blocks[6], lsize[6], psize[6], dsize[6];
394         char ref_blocks[6], ref_lsize[6], ref_psize[6], ref_dsize[6];

396         if (dds == NULL || dds->dds_blocks == 0)
397                 return;

399         if (h == -1)
400                 (void) strcpy(refcnt, "Total");
401         else
402                 zfs_nicenum(1ULL << h, refcnt, sizeof (refcnt));

404         zfs_nicenum(dds->dds_blocks, blocks, sizeof (blocks));
405         zfs_nicenum(dds->dds_lsize, lsize, sizeof (lsize));
406         zfs_nicenum(dds->dds_psize, psize, sizeof (psize));
407         zfs_nicenum(dds->dds_dsize, dsize, sizeof (dsize));
408         zfs_nicenum(dds->dds_ref_blocks, ref_blocks, sizeof (ref_blocks));
409         zfs_nicenum(dds->dds_ref_lsize, ref_lsize, sizeof (ref_lsize));
410         zfs_nicenum(dds->dds_ref_psize, ref_psize, sizeof (ref_psize));
411         zfs_nicenum(dds->dds_ref_dsize, ref_dsize, sizeof (ref_dsize));

413         (void) printf("%6s   %6s   %5s   %5s   %5s   %6s   %5s   %5s   %5s\n",
414             refcnt,
415             blocks, lsize, psize, dsize,
416             ref_blocks, ref_lsize, ref_psize, ref_dsize);
417 }

419 /*
420  * Print the DDT histogram and the column totals.
421  */
422 void
423 zpool_dump_ddt(const ddt_stat_t *dds_total, const ddt_histogram_t *ddh)
424 {
425         int h;

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

429         (void) printf("bucket   "
430             "           allocated             "
431             "          referenced          \n");
432         (void) printf("______   "
433             "______________________________   "
434             "______________________________\n");

436         (void) printf("%6s   %6s   %5s   %5s   %5s   %6s   %5s   %5s   %5s\n",
437             "refcnt",
438             "blocks", "LSIZE", "PSIZE", "DSIZE",
439             "blocks", "LSIZE", "PSIZE", "DSIZE");

441         (void) printf("%6s   %6s   %5s   %5s   %5s   %6s   %5s   %5s   %5s\n",
442             "------",
443             "------", "-----", "-----", "-----",
444             "------", "-----", "-----", "-----");

446         for (h = 0; h < 64; h++)
447                 dump_ddt_stat(&ddh->ddh_stat[h], h);

449         dump_ddt_stat(dds_total, -1);

451         (void) printf("\n");
452 }
```