****
   85483 Fri Jul 26 21:08:44 2013
new/usr/src/lib/libzfs/common/libzfs_sendrecv.c
3909 Fix hang when sending dedup stream
****
______*unchanged_portion_omitted_*

```
/*
 * Generate a send stream for the dataset identified by the argument zhp.
 *
 * The content of the send stream is the snapshot identified by
 * 'tosnap'.  Incremental streams are requested in two ways:
 *     - from the snapshot identified by "fromsnap" (if non-null) or
 *     - from the origin of the dataset identified by zhp, which must
 *       be a clone.  In this case, "fromsnap" is null and "fromorigin"
 *       is TRUE.
 *
 * The send stream is recursive (i.e. dumps a hierarchy of snapshots) and
 * uses a special header (with a hdrtype field of DMU_COMPOUNDSTREAM)
 * if "replicate" is set.  If "doall" is set, dump all the intermediate
 * snapshots. The DMU_COMPOUNDSTREAM header is used in the "doall"
 * case too. If "props" is set, send properties.
 */
int
zfs_send(zfs_handle_t *zhp, const char *fromsnap, const char *tosnap,
    sendflags_t *flags, int outfd, snapfilter_cb_t filter_func,
    void *cb_arg, nvlist_t **debugnvp)
{
        char errbuf[1024];
        send_dump_data_t sdd = { 0 };
        int err = 0;
        nvlist_t *fss = NULL;
        avl_tree_t *fsavl = NULL;
        static uint64_t holdseq;
        int spa_version;
        pthread_t tid = 0;
        int pipefd[2];
        dedup_arg_t dda = { 0 };
        int featureflags = 0;

        (void) snprintf(errbuf, sizeof (errbuf), dgettext(TEXT_DOMAIN,
            "cannot send '%s'"), zhp->zfs_name);

        if (fromsnap && fromsnap[0] == '\0') {
                zfs_error_aux(zhp->zfs_hdl, dgettext(TEXT_DOMAIN,
                    "zero-length incremental source"));
                return (zfs_error(zhp->zfs_hdl, EZFS_NOENT, errbuf));
        }

        if (zhp->zfs_type == ZFS_TYPE_FILESYSTEM) {
                uint64_t version;
                version = zfs_prop_get_int(zhp, ZFS_PROP_VERSION);
                if (version >= ZPL_VERSION_SA) {
                        featureflags |= DMU_BACKUP_FEATURE_SA_SPILL;
                }
        }

        if (flags->dedup && !flags->dryrun) {
                featureflags |= (DMU_BACKUP_FEATURE_DEDUP |
                    DMU_BACKUP_FEATURE_DEDUPPROPS);
                if (err = pipe(pipefd)) {
                        zfs_error_aux(zhp->zfs_hdl, strerror(errno));
                        return (zfs_error(zhp->zfs_hdl, EZFS_PIPEFAILED,
                            errbuf));
                }
                dda.outputfd = outfd;
```

```
                dda.inputfd = pipefd[1];
                dda.dedup_hdl = zhp->zfs_hdl;
                if (err = pthread_create(&tid, NULL, cksummer, &dda)) {
                        (void) close(pipefd[0]);
                        (void) close(pipefd[1]);
                        zfs_error_aux(zhp->zfs_hdl, strerror(errno));
                        return (zfs_error(zhp->zfs_hdl,
                            EZFS_THREADCREATEFAILED, errbuf));
                }
        }

        if (flags->replicate || flags->doall || flags->props) {
                dmu_replay_record_t drr = { 0 };
                char *packbuf = NULL;
                size_t buflen = 0;
                zio_cksum_t zc = { 0 };

                if (flags->replicate || flags->props) {
                        nvlist_t *hdrnv;

                        VERIFY(0 == nvlist_alloc(&hdrnv, NV_UNIQUE_NAME, 0));
                        if (fromsnap) {
                                VERIFY(0 == nvlist_add_string(hdrnv,
                                    "fromsnap", fromsnap));
                        }
                        VERIFY(0 == nvlist_add_string(hdrnv, "tosnap", tosnap));
                        if (!flags->replicate) {
                                VERIFY(0 == nvlist_add_boolean(hdrnv,
                                    "not_recursive"));
                        }

                        err = gather_nvlist(zhp->zfs_hdl, zhp->zfs_name,
                            fromsnap, tosnap, flags->replicate, &fss, &fsavl);
                        if (err)
                                goto err_out;
                        VERIFY(0 == nvlist_add_nvlist(hdrnv, "fss", fss));
                        err = nvlist_pack(hdrnv, &packbuf, &buflen,
                            NV_ENCODE_XDR, 0);
                        if (debugnvp)
                                *debugnvp = hdrnv;
                        else
                                nvlist_free(hdrnv);
                        if (err)
                                goto stderr_out;
                }

                if (!flags->dryrun) {
                        /* write first begin record */
                        drr.drr_type = DRR_BEGIN;
                        drr.drr_u.drr_begin.drr_magic = DMU_BACKUP_MAGIC;
                        DMU_SET_STREAM_HDRTYPE(drr.drr_u.drr_begin.
                            drr_versioninfo, DMU_COMPOUNDSTREAM);
                        DMU_SET_FEATUREFLAGS(drr.drr_u.drr_begin.
                            drr_versioninfo, featureflags);
                        (void) snprintf(drr.drr_u.drr_begin.drr_toname,
                            sizeof (drr.drr_u.drr_begin.drr_toname),
                            "%s@%s", zhp->zfs_name, tosnap);
                        drr.drr_payloadlen = buflen;
                        err = cksum_and_write(&drr, sizeof (drr), &zc, outfd);

                        /* write header nvlist */
                        if (err != -1 && packbuf != NULL) {
                                err = cksum_and_write(packbuf, buflen, &zc,
                                    outfd);
                        }
                        free(packbuf);
                        if (err == -1) {
                                err = errno;
                                goto stderr_out;
                        }

                        /* write end record */
                        bzero(&drr, sizeof (drr));
                        drr.drr_type = DRR_END;
                        drr.drr_u.drr_end.drr_checksum = zc;
                        err = write(outfd, &drr, sizeof (drr));
                        if (err == -1) {
                                err = errno;
                                goto stderr_out;
                        }

                        err = 0;
                }
        }

        /* dump each stream */
        sdd.fromsnap = fromsnap;
        sdd.tosnap = tosnap;
        if (tid != 0)
                sdd.outfd = pipefd[0];
        else
                sdd.outfd = outfd;
        sdd.replicate = flags->replicate;
        sdd.doall = flags->doall;
        sdd.fromorigin = flags->fromorigin;
        sdd.fss = fss;
        sdd.fsavl = fsavl;
        sdd.verbose = flags->verbose;
        sdd.parsable = flags->parsable;
        sdd.progress = flags->progress;
        sdd.dryrun = flags->dryrun;
        sdd.filter_cb = filter_func;
        sdd.filter_cb_arg = cb_arg;
        if (debugnvp)
                sdd.debugnv = *debugnvp;

        /*
         * Some flags require that we place user holds on the datasets that are
         * being sent so they don’t get destroyed during the send. We can skip
         * this step if the pool is imported read-only since the datasets cannot
         * be destroyed.
         */
        if (!flags->dryrun && !zpool_get_prop_int(zfs_get_pool_handle(zhp),
            ZPOOL_PROP_READONLY, NULL) &&
            zfs_spa_version(zhp, &spa_version) == 0 &&
            spa_version >= SPA_VERSION_USERREFS &&
            (flags->doall || flags->replicate)) {
                ++holdseq;
                (void) snprintf(sdd.holdtag, sizeof (sdd.holdtag),
                    ".send-%d-%llu", getpid(), (u_longlong_t)holdseq);
                sdd.cleanup_fd = open(ZFS_DEV, O_RDWR|O_EXCL);
                if (sdd.cleanup_fd < 0) {
                        err = errno;
                        goto stderr_out;
                }
                sdd.snapholds = fnvlist_alloc();
        } else {
                sdd.cleanup_fd = -1;
                sdd.snapholds = NULL;
        }
        if (flags->verbose || sdd.snapholds != NULL) {
                /*
```

```
                 * Do a verbose no-op dry run to get all the verbose output
                 * or to gather snapshot hold’s before generating any data,
                 * then do a non-verbose real run to generate the streams.
                 */
                sdd.dryrun = B_TRUE;
                err = dump_filesystems(zhp, &sdd);

                if (err != 0)
                        goto stderr_out;

                if (flags->verbose) {
                        if (flags->parsable) {
                                (void) fprintf(stderr, "size\t%llu\n",
                                    (longlong_t)sdd.size);
                        } else {
                                char buf[16];
                                zfs_nicenum(sdd.size, buf, sizeof (buf));
                                (void) fprintf(stderr, dgettext(TEXT_DOMAIN,
                                    "total estimated size is %s\n"), buf);
                        }
                }

                /* Ensure no snaps found is treated as an error. */
                if (!sdd.seento) {
                        err = ENOENT;
                        goto err_out;
                }

                /* Skip the second run if dryrun was requested. */
                if (flags->dryrun)
                        goto err_out;

                if (sdd.snapholds != NULL) {
                        err = zfs_hold_nvl(zhp, sdd.cleanup_fd, sdd.snapholds);
                        if (err != 0)
                                goto stderr_out;

                        fnvlist_free(sdd.snapholds);
                        sdd.snapholds = NULL;
                }

                sdd.dryrun = B_FALSE;
                sdd.verbose = B_FALSE;
        }

        err = dump_filesystems(zhp, &sdd);
        fsavl_destroy(fsavl);
        nvlist_free(fss);

        /* Ensure no snaps found is treated as an error. */
        if (err == 0 && !sdd.seento)
                err = ENOENT;

        if (tid != 0) {
                if (err != 0)
                        (void) pthread_cancel(tid);
                (void) close(pipefd[0]);
#endif /* ! codereview */
                (void) pthread_join(tid, NULL);
                (void) close(pipefd[0]);
        }

        if (sdd.cleanup_fd != -1) {
                VERIFY(0 == close(sdd.cleanup_fd));
                sdd.cleanup_fd = -1;
        }
```

```
	if (!flags->dryrun && (flags->replicate || flags->doall ||
	    flags->props)) {
		/*
		 * write final end record.  NB: want to do this even if
		 * there was some error, because it might not be totally
		 * failed.
		 */
		dmu_replay_record_t drr = { 0 };
		drr.drr_type = DRR_END;
		if (write(outfd, &drr, sizeof (drr)) == -1) {
			return (zfs_standard_error(zhp->zfs_hdl,
			    errno, errbuf));
		}
	}

	return (err || sdd.err);

stderr_out:
	err = zfs_standard_error(zhp->zfs_hdl, err, errbuf);
err_out:
	fsavl_destroy(fsavl);
	nvlist_free(fss);
	fnvlist_free(sdd.snapholds);

	if (sdd.cleanup_fd != -1)
		VERIFY(0 == close(sdd.cleanup_fd));
	if (tid != 0) {
		(void) pthread_cancel(tid);
		(void) close(pipefd[0]);
#endif /* ! codereview */
		(void) pthread_join(tid, NULL);
		(void) close(pipefd[0]);
	}
	return (err);
}
```

______unchanged_portion_omitted_