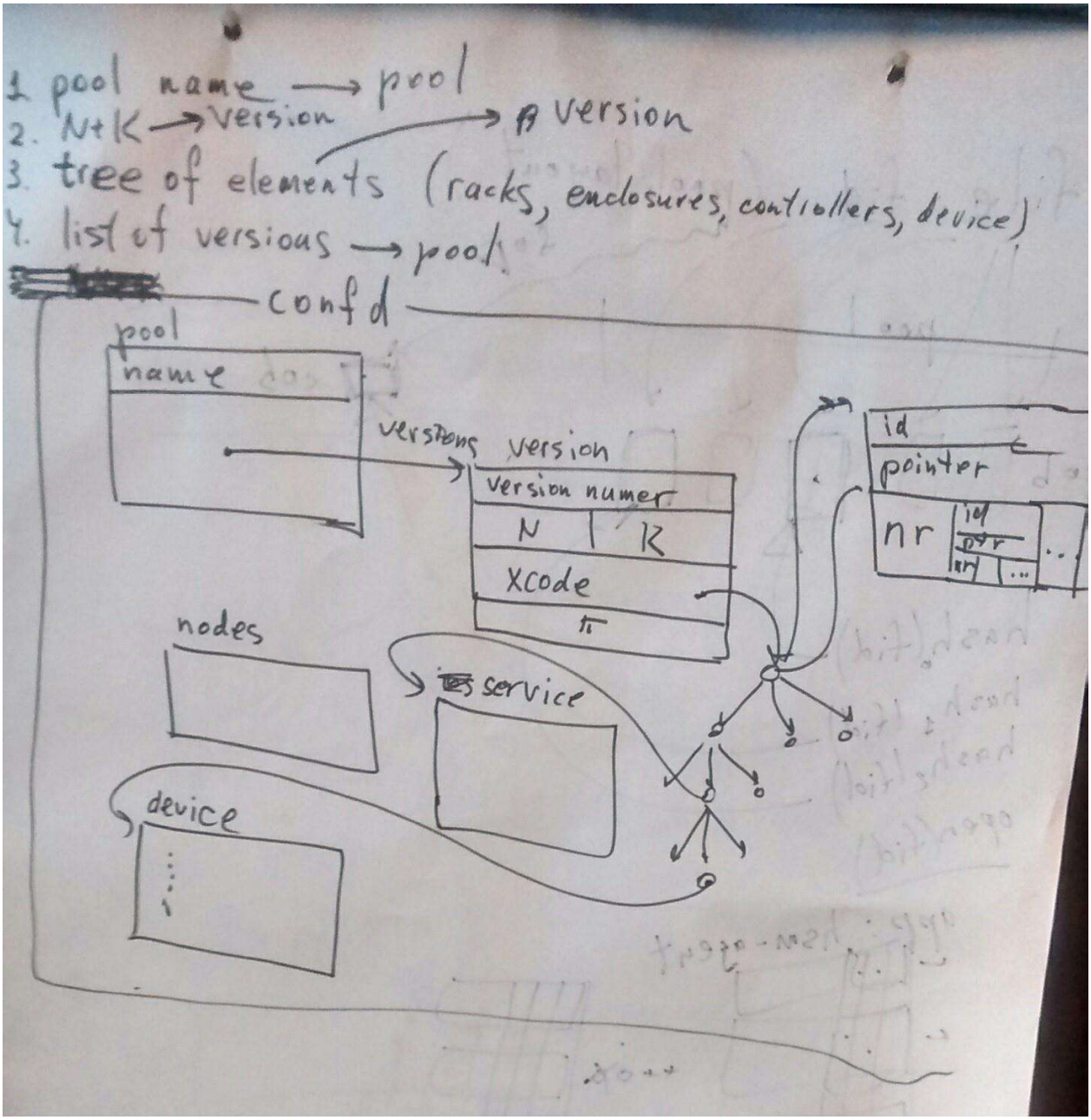
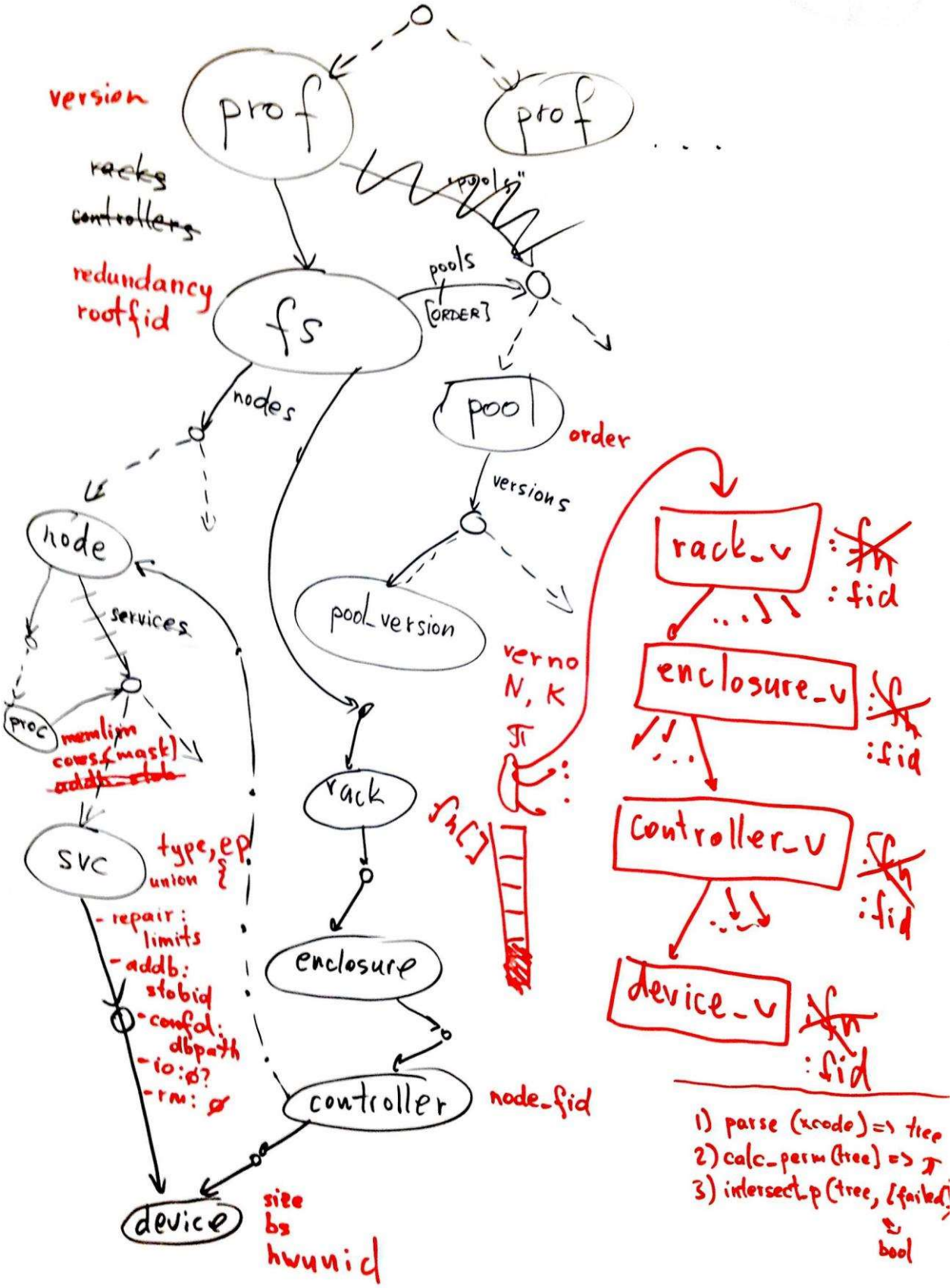
Description

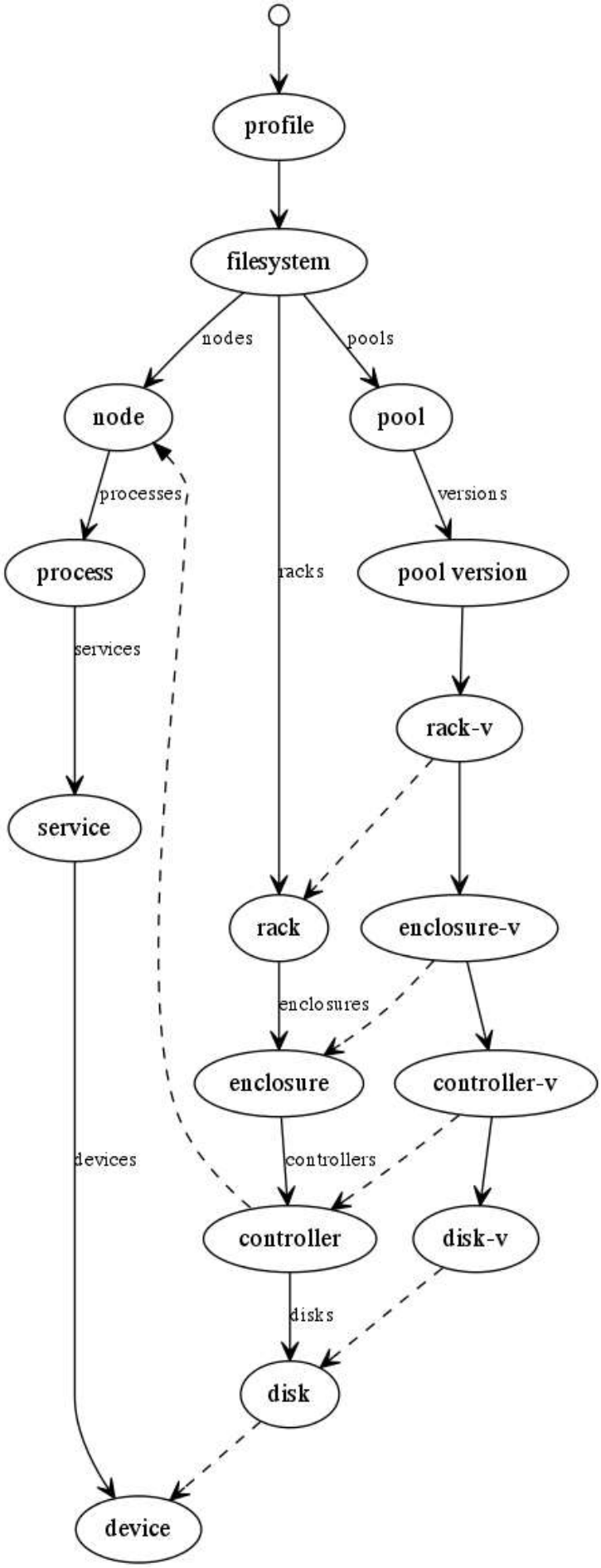
Support multiple pool configurations in order to support dynamic addition of devices, nodes, services, and racks.

Details

Pools are used to partition hardware resources (devices, servers). Pool versions are used to track changes in pool membership.


1. pool name → pool
2. N+K → version → version
3. tree of elements (racks, enclosures, controllers, device)
4. list of versions → pool
confd
pool
name
versions
version
version numer
N
K
Xcode
nodes
service
device
id
pointer
nr


version
prof
prof
racks
controllers
redundancy
rootfid
fs
pools
[ORDER]
nodes
pool
order
versions
node
services
pool_version
verno
N, K
proc
memlim
cows{mask}
addb stob
rack
svc
type, ep
union {
- repair: limits
- addb: stobid
- confd: dbpath
- io: ?
- rm:
enclosure
controller
node_fid
device
size
bs
hwuuid
rack_v
: fid
enclosure_v
: fid
controller_v
: fid
device_v
: fid
1) parse (xcode) => tree
2) calc_perm (tree) => π
3) intersect_p (tree, [failed]
bool


profile
filesystem
nodes
pools
racks
node
pool
processes
versions
process
pool version
services
rack-v
service
rack
enclosure-v
enclosures
enclosure
controller-v
devices
controllers
controller
disk-v
disks
disk
device

[Google Drive folder with DOT source and images]

Subtasks, estimations

1. Detailed design (DLD) - Mero-531 - (200 loc)
2. add pools to the configuration schema. - (100 loc) MERO-462
3. add pool versions to the configuration schema:
    a. ~~(not needed, user supplies pool versions) an algorithm to generate pool versions from a pool (if there are C controllers we have 2^C pool versions) (100) -~~
    b. modify m0t1fs to use multiple pools (100 LOC) - Mero-463
    c. store the tree associated with a pool version (50). - MERO-464
    d. modules to parse and manipulate the collection of pool-versions (stored as list) (100). MERO-465
4. Interfaces for m0t1fs to use pools and pool versions - MERO-466
    a. initialize/update client's failure set (CODE / 50 loc)
    b. find latest pool/pool version which does not intersect with failure set on configuration changes. (CODE / 50 loc)
5. modify ioservice to work with multiple pools, return version mismatch to clients as necessary, return pool version and pool in getattr reply (CODE / 50 loc) MERO - 467
6. associate layouts with pool versions (CODE / 50) - MERO-468
7. support assignment of pools to newly created objects (CODE / 50 loc) - MERO-469

Dependencies

Finding poolversion:

(Subtask 4b)

```
struct m0_confx_rack {
    struct m0_confx_header xr_header;
    /* Enclosures on this tack */
    struct arr_fid      xr_encls;
        /** List of associated pool version fids */
    struct arr_fid      xr_pvers;
} M0_XCA_RECORD;

struct m0_confx_encl {
    struct m0_confx_header xe_header;
    /* Controllers in this enclosure */
    struct arr_fid      xe_cntrls;
```

```
    /** List of associated pool version fids */
    struct arr_fid      xe_pvers;
} M0_XCA_RECORD;

struct m0_confx_cntrl {
    struct m0_confx_header xc_header;
    /* Storage devices attached with this controller */
    struct arr_fid      xc_devs;
    /** List of associated pool version fids */
    struct arr_fid      xc_pvers;
} M0_XCA_RECORD;
```

pool_version_add() populates
m0_confx_rack:: xr_pvers, m0_confx_encl:: xe_pvers & m0_confx_cntrl:: xc_pvers

Use cases can be modified as :

```
rackv_add(fid, pool_version, rack_fid) {
        rack_v = confc.alloc_add(fid);
        rack_v.rack_fid = rack_fid;
        pool_version.rackv_add(rack_v);

        rack = confc.lookup(rack_fid);
         rack.pvers_add(pool_version);
}

enclosurev_add(fid, pool_version, rack_v, encl_fid){
      encl_v = confc.alloc_add(fid);
      encl_v.encl_fid = encl_fid;
      rack_v.enclv_add(encl_v);

      encl = confc.lookup(encl_fid);
      encl.pvers_add(pool_version);
}

controllerv_add(fid, pool_version, encl_v, cntlr_fid) {
        cntlr_v = confc.alloc_add(fid);
        cntlr_v.cntlr_fid = cntlr_fid;
        encl_v.cntlrv_add(cntlr_v);

        cntl = confc.lookup(cntlr_fid);
        cntl.pvers_add(pool_version);
```

```
}



pool_version_get(confc, failure_set,**pver){
        available_pool_versions = pool_versions;
        for each f_dev in failure_devices{
                /* exclude failed device pool versions */
                available_pool_versions = available_pool_versions - fdev.pvers;
        }
        return available_pool_versions[0];
}

pool_version_get_nikita(confc, failure_set) : pool_version {
        for (pool in confc.pools) { /* iterate pools in order */
                pool_version = pool.last_version;
                for (e : failure_set) {
                        /* e can be device, or rack, or ... */
                        if (e.pvers_contains(pool_version)) {
                                /* pool_version intersects with the
                                failure set, ignore it. */
                                continue outer loop;
                        }
                }
                return pool_version;
}
```