KERLABS

Future developments in the container sub-system

Renaud Lottiaux

Kerrighed
Linux clusters made easy

3rd Level Object Naming

- Today an object is identified by
  - A container id (32 bits identifier)
  - An object id (up to 32 bits identifier)
- In some cases, this naming is not enough, we would like more “bits” to name the object.
- For instance, containers hosting pages from a file
  - One file is identified by a inode number (32 bits)
  - In the file, the page is identified by a page id (32 bits)
  - We need all the current address space !
    - Even more if several file systems are mounted !
- Solution: introduce a third level of naming
  - A “class” identifier

What is a class ?

- A class is a “family” of containers
  - Inodes from a given file system
  - Address spaces of processes
  - System containers
  - Etc...
- It's a 32 bits identifier

```
void * ctnr_get_object  ( classid_t classid, /* Id of the container class */
                          ctnrid_t ctnrid,   /* Id of the container */
                          objid_t objid      /* Id of the object */
                        )
```

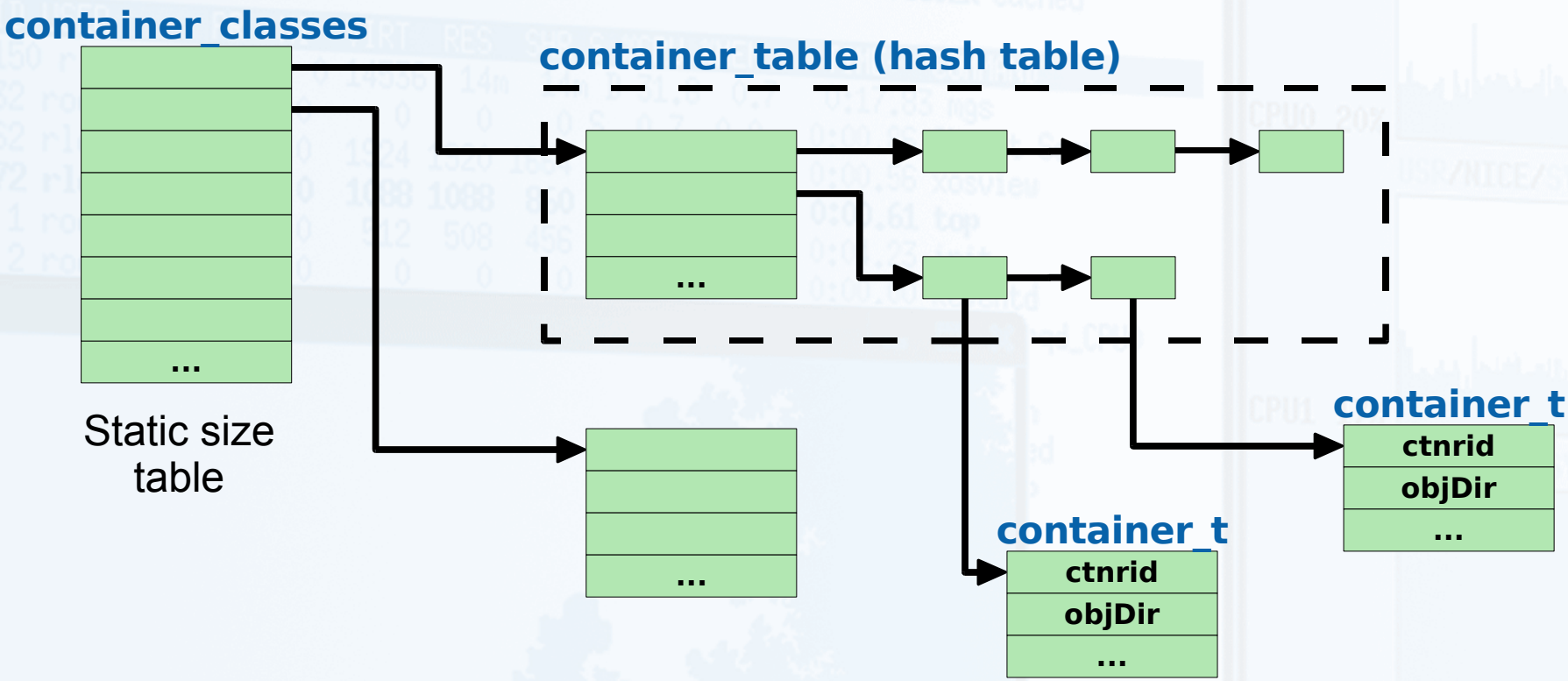
A new interface function

- Add a new set of functions to use classes only when needed
  - Most containers are “system” containers, belonging to the same class: CTNR_SYSTEM_CLASS
  - Avoid modifying all the existing access to container

```
void * ctnr_get_object  ( ctnrid_t ctnrid,   /* Id of the container */
                          objid_t objid      /* Id of the object */
                        )
```

```
void * ectnr_get_object ( classid_t classid, /* Id of the container class */
                          ctnrid_t ctnrid,   /* Id of the container */
                          objid_t objid      /* Id of the object */
                        )
```

Data Structure



Object Lookup Optimization

- Currently, each call to a ctnr_*_object leads to
  - A look-up in the container table
  - A look-up in the object table
- In many cases, these look-ups can be avoided
  - The same container is often used many time in the same function.
    - One look-up is enough for the whole function
  - At least 2 functions are called to manipulate an object
    - ctnr_{get,grab}_object followed by a ctnr_put_object
      - An object look-up for the ctnr_put_object is not usefull
- The idea: extend the interface to allow optimized manipulation of object

Extended Interface

- Regular interface is kept

```
void * ctnr_*_object ( ctnrid_t ctnrid,   /* Id of the container */
                       objid_t objid      /* Id of the object */
                     )
```

- New interface

```
void * _ctnr_*_object   ( ctnrid_t ctnrid,       /* Id of the container */
                          objid_t objid,         /* Id of the object */
                          ctnrObj_t **obj        /* Pointer to object entry */
                        )
```

Example

```
void * obj;

obj = ctnr_get_object ( ctnrid, objid );

ctnr_put_object ( ctnrid, objid );
```

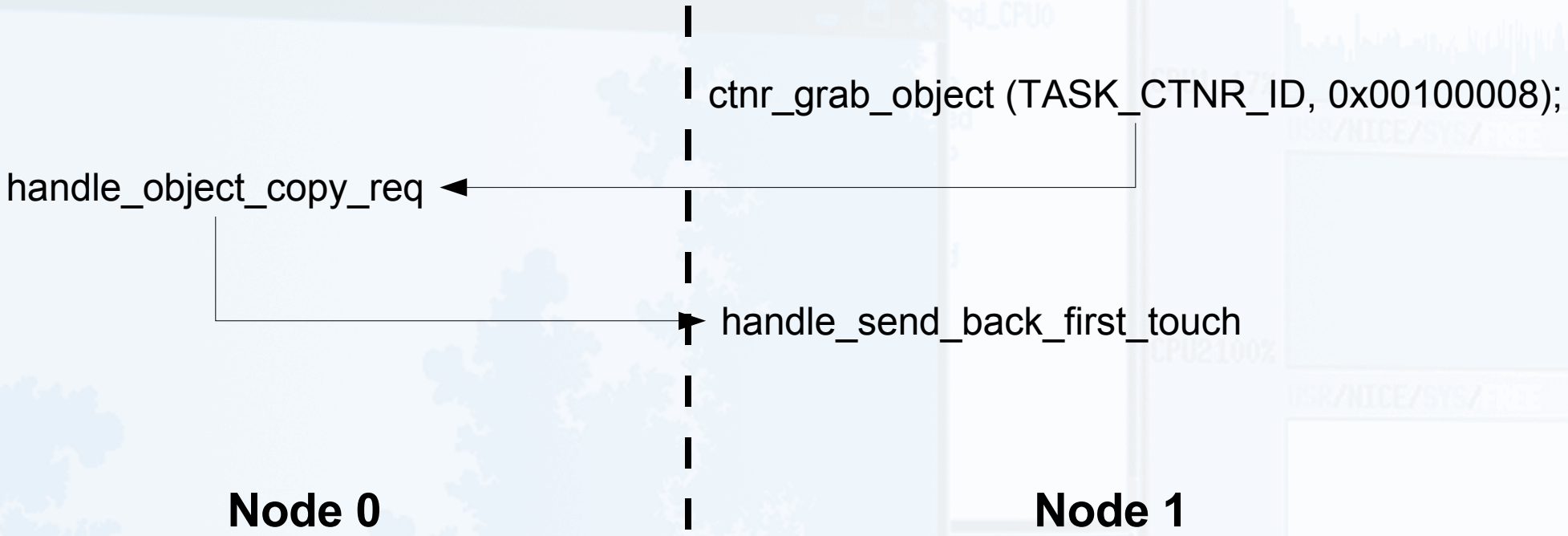
```
void * obj;
ctnrObj_t * obj_entry = NULL;

obj = _ctnr_get_object ( ctnrid, objid, &obj_entry );

_ctnr_put_object ( obj_entry );
```

Default Owner Optimization

- Currently, default owner is defined using a round-robin policy
  - default_owner = objid % kerrighed_nb_nodes
- In many cases, this induces useless network traffic
- Let's take the PID example
  - PID is generated using creation node id + local pid



Default Owner Optimization

- What is the best default owner strategy ?
  - It really depends on the object type
    - The developer of a given container know exactly what is the best policy for its container
- Add a new way to define the default owner of an object
  - Function pointer passed during the creation of a container

Open Issues

- Object entry is too big
  - Currently : 74 bytes
  - Can easily reduced to
    - 42 bytes for non master copies
    - 74 bytes for master copies
  - But, still to big...
  - Some other fields can also be removed
  - The big issue : the copyset...
    - Can grows dramatically with number of nodes