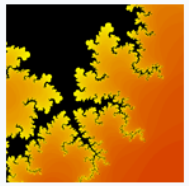
KERLABS

Communication system: current status and future developments

Kerrighed
Linux clusters made easy

Outlines



- Janitor
  - deprecate RPC mechanism
  - remove virtual-node mechanism
- Features
  - transparent switch between in-interrupt operations and work-queue-like operations
  - efficient pack/unpack
  - canceled transaction
  - communication failure support
  - time management support

in-interrupt / work-queue-like operations

- need efficient small request management
- need to support work-queue-like mechanism in order to:
  - handle burst of requests
  - support blocking handlers

- Add a TRANSACT_AM flag
  - declare a callback “active-message” aware
- By default work-queue-like operations

RPC mechanism

- keep the current API
- port the engine on top of transaction

- Benefits:
  - remove duplicated code
  - one simple interface in order to define
    - active message (non blocking handler)
    - work-queue like action (might blocking handler)
  - Remove the bunch of service_manager threads.

Virtual-node mechanism

- node to processes communication
- Pb: how to broadcast a request to migrating processes ?
- Need a reliable multi-cast support
  - new *comm* protocol dedicated to this

Pack/Unpack

- Current implementation:
  - one pack == one message (regardless the size)
- Need to aggregate fragments of transaction in order
  - to save the bandwidth
  - to reduce the pressure on the receive side

Canceled transaction

- with transaction, request are handled packets per packets
- after the first packet, we may decide to discard the current transaction
  - out of memory
  - “useless” request
- Need to be able to stop the receive operation:
  - krg_transact_cancel(desc)
    - advertise the other side of this decision
  - krg_transact_discard(desc)
    - silently discard all the future packet in this transaction

Communication failure support

- Network layer supposed to be reliable
  - all sent messages are supposed delivered
  - transaction are supposed to complete (ie: execute the callback)
- What's happen in a node failure case ?
  - transaction will be blocked:
    - sender: waiting for an ack
    - receiver: waiting for some packets
  - Need to advertise to upper layers
- Current transaction take care of successful transaction
  - need to add one callback dedicated to the failure case
  - keep the same kind of interface
    - provide a set of two callbacks (cb and failure_cb)

Time management support

- How to avoid *time travel* ?
  - process might migrate in the past
- Can't have the same time on all the nodes
  - Need to ensure that time is always increasing
- Need to embed time informations in:
  - migration operations
  - all or periodic messages (heartbeat?)

Conclusions

- Communication layer quite stable
- New features are about:
  - efficiency
  - cluster changes support
- Features list is not closed
- No particular schedule decided

- I'm not forgetting dynamic streams:
  - full rewrite is needed