ERl_DHT version 1.0

User Guide

Saket kunwar

Aug 26,2009

Erly_ring

Intro

---

erly_ring is an implementation of erl_dht(gitclone) for use with tcp/ip socket based communication.All modules and functions remains unchanged.i.e it is used as in the simulation version.This is largely due to the message passing programming feature faciliated by the use of erlang.The only difference is that erly_ring is adapted to use the communication interface provided by tcp_node_client.erl,tcp_node_server.erl.The module dispatcher.erl does all internal routing which is especiall suitable if a host intends to use multiple erly_ring node on one host and same port.

Running erly_ring

---

edit the config file("config.txt") first provided in ./ebin directory
to point to boot host and other nodes
config example
{{localhostname,"localhost"}, %%address of the host...change the name to that of the host comp
{startport,5000}, %%the port used by the node
{maxapp,10}, %%this is currently not used...but leave the default
{boothost,"localhost"}, %%address of boot host ....change the name to that of the host comp
{bootport,5009}}. %% port of boot host

1.
From erlang shell

cd to ../ebin
it's better to spawn the shell with a name i.e erl -sname node1,
especiall when spawning multile nodes with same node and host.

2.erly_ring:start_boot().

and then to start other nodes
3.erly_ring:start_node(node()).
4.for nodes that wants to utilize the same port as other node
do erly_ring:start_node(masternode@localhost) where masternode@localhost is the name of a node that has already been initialized to erly_ring node.This is where the tcp_node_server.erl and dispatcher.erl work together to route the messages to the particular node concerned.

To_do
There are a lot of timing related issues that needs to be fixed
such as stabilization periods and querry periods.
som tcp error handling also needs to be provided.
and extensive testing in a large network environment

Erl_dht

Erl_dht is a simulation framework for evaluating and deploying routing schemes for distributed hash tables.Currently chord has been implemented but the modular nature of the core api makes it extensible making it possible to implemented other protocols.
An important feature of erl_dht is the ease of use for live deployment by implementing a costum communication interface such as tcp/ip.
Erl_dht is coded in erlang otp ,a highly concurrent ,distributed and fault tolerant functional programming language.The message passing feature of erlang is especially suited for performing simulation.The basic architecture of the api is as in fig 1.

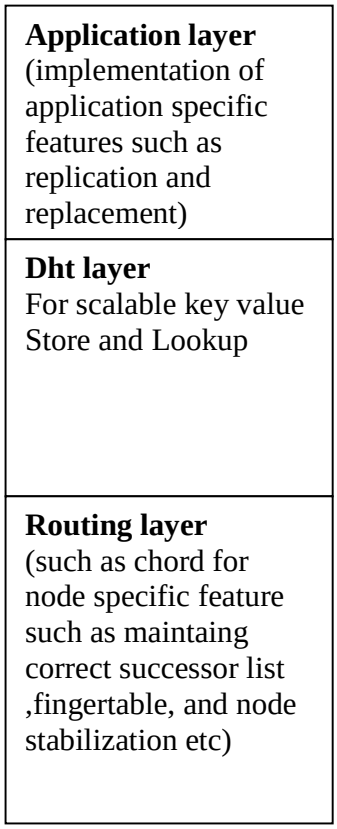

Fig 1

Building

cd /src
  make

the binary will be stored at ebin directory /ebin

Running

There is an example of events file i.e "events.txt" that contains the events to be executed..
run the script erl_dht_test or erl_dht_test.bat (windows)
or
  erl -pa ./ebin
and at the erl promt erl_dht:eventtest("events.txt").
or erl_dht:start(Num). where Num is the number of nodes

When performing simulation the terminal that runs the simulation often gets swamped with simulation result outputs and is difficult to send other live commands.To overcome this i usually spawn a terminal with erl –sname somename and execute the simulation at the erl terminal then I spawn another terminal with erl –sname someothername and send node join,node leave ,store or lookup commands through the someothername terminal with rpc calls as :
rpc:call(somename@host,simul,stop,[]). etc
This way I can monitor simulation events as well as send commands or querries to the simulation.Have a look at simul.erl or it's doc to perform other simulation specific tests.

```
add node:
 To add node during simulation
 Boot=rpc:call(somename@host,simul,get_boot,[]).
  rpc:call(somename@host,simul,add_node,[Boot]).
kill Node:
 To kill node during simulation
rpc:call(somename@host,boot,nodelist,[]).
Select node Node from the nodelist then do
 rpc:call(somename@host,simul,kill_node,[Node]).


Follow the same procedures for store,lookup and other test.
```

Event Prototype

Most Simulation specific use can be accomplished through a script i.e
simul:eventtest("eventfile.txt"). Where eventfile.txt is the event script

{init,{numNode,2,0},{resultfile,"result.txt"}}.
{{event,Sn},{function,join,random},{at,T}}.
{{event,Sn},{function,leave,Num},{at,T}}. %%the num node in masterlist of nodes
{{event,Sn},{function,{store,"kunwar",10},3}{at,T}}.
{{event,Sn},{function,{lookup,"kunwar"},3}{at,T}}.
{{event,Sn},{function,all_test,{"saket",Val}},{at,T}}.
{{event,Sn},{function,analyse,{"data.dat",succlist}},{at,T}}.

where T is the time interval to wait till next execution

The eventmanager module loads events to be executed
sequentially from the event file and evalutes a lazy function for each events.
The stored lazy function are executed in a discrete event driven simulation.
The parameter {at,T} determines T ,which is the time the next event waits to be executed.

The analyserZ module does the required analysis of the correctness of the routing
List i.e the successor list and the finger table entries.The lists are compared against
a brute computation. Due to the algorithm of chord all predecessor nodes are correct as
compared to brute evaluation however it is not necessary that all nodes should have
correct successor or finger list since nodes are only aware of the departure of immediate
successor and are thus updated or when the analysis is performed the nodes haven't had
enough stabilization period to update it's successor list.

Directory Structure

ebin	Directory where all the beam files are stored after compilation
src	Contains all the source files
doc	Contains all the edoc generated doc files
example	Few examples of simulation usage

example	Contains some example of event script