# ZHT:

## A Light-weight Reliable Persistent Dynamic Scalable Zero-hop Distributed Hash Table

**Development tutorial**

Tonglin Li, Xiaobing Zhou

Illinois Institute of Technology, Chicago, U.S.A

# idea overview

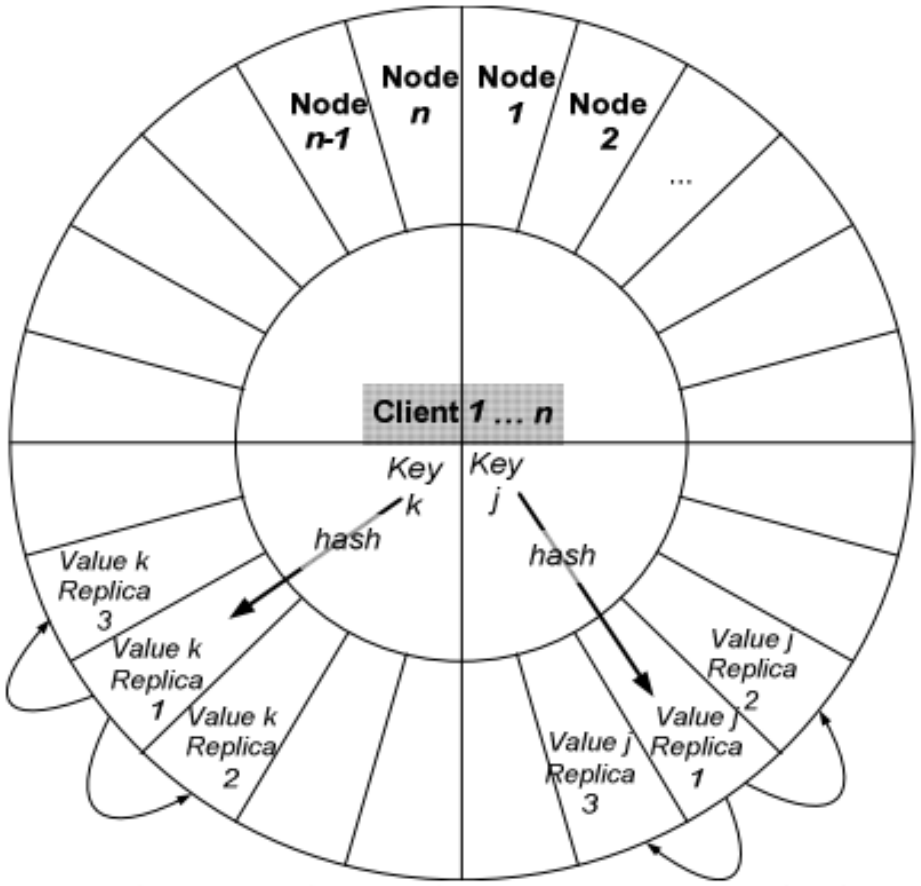



Figure 2: ZHT architecture design showing namespace, hash function, and replication

# architecture overview

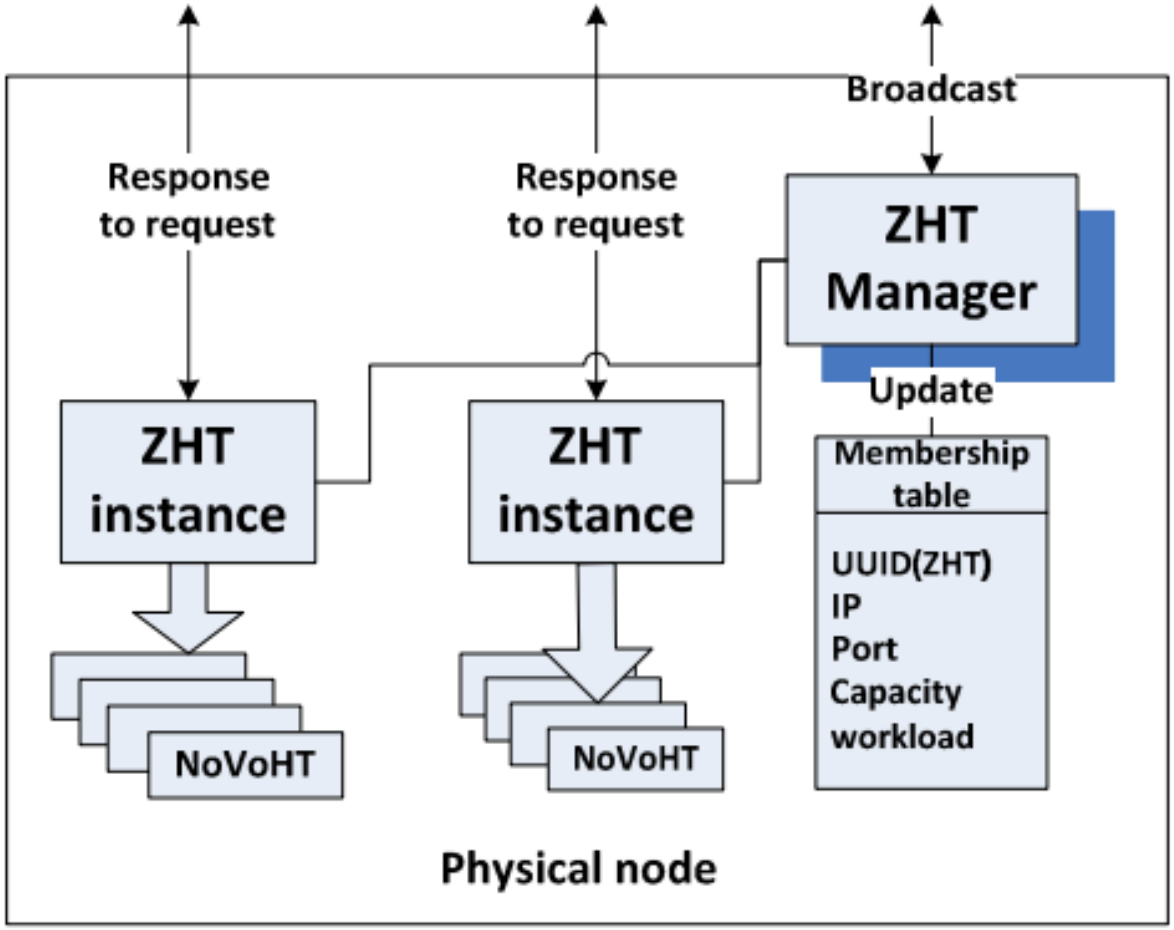
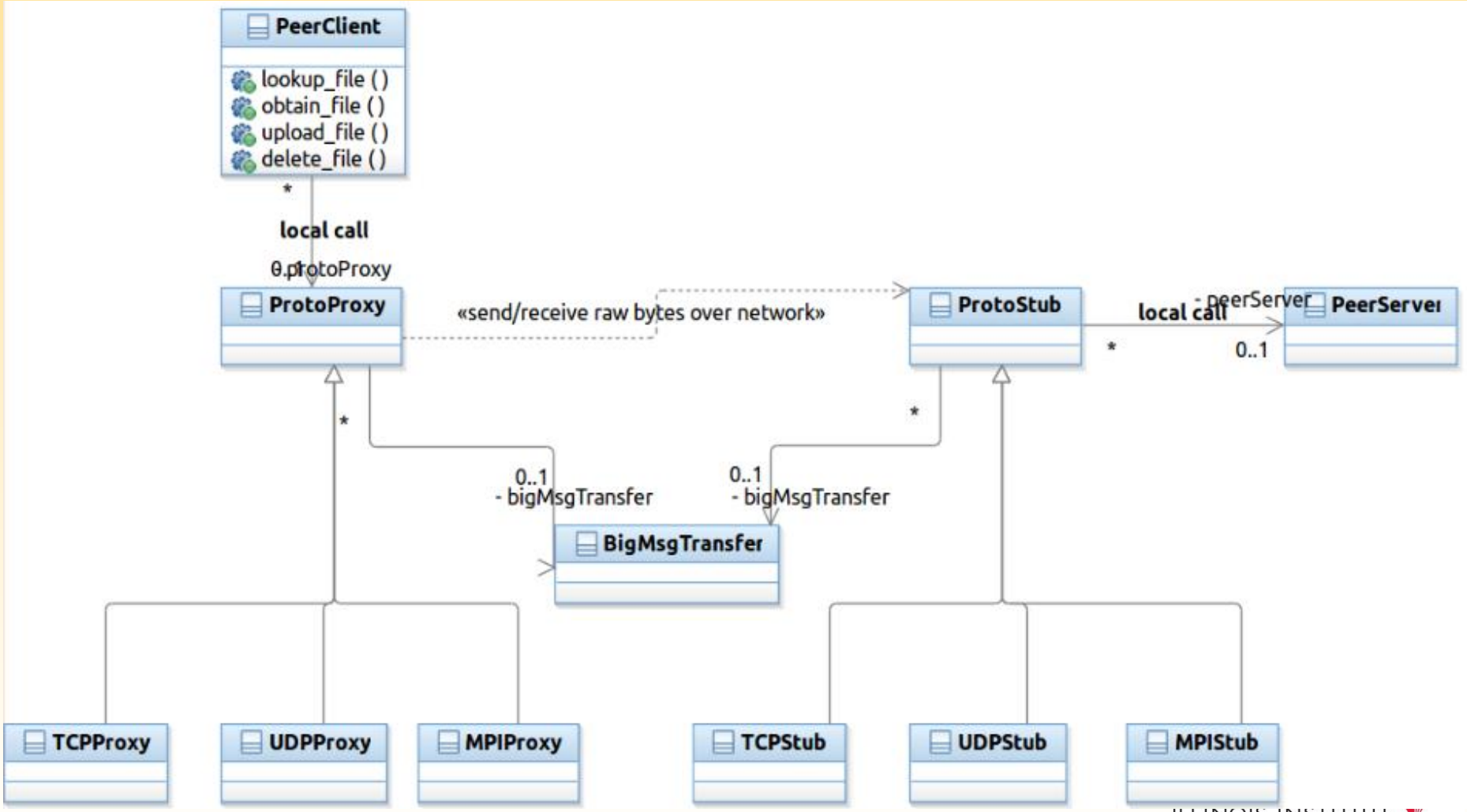



Figure 3: ZHT architecture per node

# file structure and readme

- See also <FILE_STRUTURE>
- See also <README>
- https://bitbucket.org/xiaobingo/iit.datasys.zht-mpi

# Protocol abstraction



# How to choose Protocol

- zht.conf
  - PROTOCOL TCP
  - PROTOCOL UDP
  - PROTOCOL MPI

# How to build ZHT

- Make executables for IP protocol family
  - make
  - Executables:
    - zht_cpptest/zht_ctest/zhtserver/c_zhtclient_lanl_threaded/c_zhtclient_threaded_test/cpp_zhtclient_threaded_test
- Make executables for MPI protocol family
  - make mpi
  - Executables:
    - zht-mpibroker
    - zht-mpiserver

# How to launch ZHT servers

- For IP protocol family:
  - ./zhtserver -z zht.conf -n neighbor.conf
- For MPI protocl family:
  - mpiexec -np 4 ./zht-mpiserver -z zht.conf -n neighbor.mpi.conf : ./zht-mpibroker -z zht.conf -n neighbor.mpi.conf
- The ZHT client and YOUR_OWN_APP are not aware of the protocols

# ZHT Language bindings

- C
- C++
- Recommendations
  - Always try the C++ binding since IT'S MORE CONVENIENT to pass user-defined composite data structure using OFFICIAL Google protocol buffer C++ binding
  - ZHT C binding depends on NON- OFFICIAL Google protocol buffer C binding, BUGS potentailly
  - ZHT C binding is built on top of C++ binding

# How to dev your ZHT apps

- Find C examples to call ZHT-client-API
  - See <c_zhtclient_test.c> for C example on how to call ZHT-client-API
  - See <c_zhtclient_threaded_test.cpp> and <c_zhtclient_lanl_threaded.c> for C example on how to call ZHT-client-API in multi-threaded context

# How to dev your ZHT apps

- Find C++ examples to call ZHT-client-API
  - See <cpp_zhtclient_test.cpp> for C++ example on how to call ZHT-client-API
  - See <cpp_zhtclient_threaded_test.cpp> for C++ example on how to call ZHT-client-API in multi-threaded context

# How to dev your ZHT apps - walkthrough

- Assume the directory:
  - iit.datasys.zht-mpi
  - iit.datasys.zht-mpi/src
  - iit.datasys.zht-mpi/tutorial
  - iit.datasys.zht-mpi/tutorial/zhtsample.cpp
- See iit.datasys.zht-mpi/src/README to install < Google protocol buffers c binding, VERSION 0.15 > and < Google protocol buffers c++ binding, VERSION 2.4.1 >
- for example, you got iit.datasys.zht-mpi/tutorial/zhtsample.cpp as your app
- cd to iit.datasys.zht-mpi/src
- make or make mpi
- cd ../tutorial/

# How to dev your ZHT apps - walkthrough

- mkdir include #create dir to hold ZHT header files your app may need
- mkdir lib #create dir to hold ZHT lib file your app needs to link to
- cp ../src/*.h include/ #copy ZHT header files
- cp ../src/libzht.a lib/ #copy ZHT lib file
- vim comp.sh and enter
  - gcc –g –o zhtsample zhtsample.cpp -Iinclude/ -Llib/ -lzht -lstdc++ -lpthread -lprotobuf -lprotobuf-c

# How to dev your ZHT apps - walkthrough

- bash comp.sh #this will generate executable zhtsample
- cd to ../src, and start ZHT server as
  - ./zhtserver –z zht.conf –n neighbor.conf
- cd ../tutorial
- Run ZHT sample as
  - ./zhtsample –z ../src/zht.conf –n ../src/neighbor.conf

# How to dev your ZHT apps – passing composite datastructure

- cd to iit.datasys.zht-mpi/tutorial/
- Define your Google protocol buffer specification file, vim student.proto and enter
  - message Student {
    - required int32 id = 1;
    - required bool gender = 2;
    - required bytes firstname = 3;
    - required bytes lastname = 4;
    - required bytes address = 5;
    - required bytes phone = 6;
    - optional bytes hobbies = 7;
    - repeated bytes course = 8;
  - }
- protoc –cpp_out=. student.proto #this will generate student.pb.h and student.pb.cc

# How to dev your ZHT apps – passing composite datastructure

- For example, you got iit.datasys.zht-mpi/tutorial/udtsample.cpp as app
- Other steps same as others in previous case
- vim comp.sh and append line as
  - gcc –g –o udtsample udtsample.cpp student.pb.cc -Iinclude/ -Llib/ -lzht -lstdc++ -lpthread -lprotobuf -lprotobuf-c
- launch zhtserver as mentioned before
- Run udtsample as
  - ./udtsample –z ../src/zht.con –n ../src/neighbor.conf

# How to dev your ZHT apps – passing composite datastructure

- To define relationship between student and enrollment, see https://developers.google.com/protocol-buffers/ for details

# How to dev your ZHT apps – define your persistent storage

- When you launch zhtserver, it prompts:
  - Usage:
  - ./zhtserver -z zht.conf -n neighbor.conf [-p port] [-f novoht_db_file] [-h(help)]
- Using -f option, you specify the file to persist items you opeated, e.g. novoht_db_file

# How to dev your ZHT apps – run ZHT over MPI protocol(standalone mode)

- vim iit.datasys.zht-mpi/src/zht.conf, set
  - PROTOCOL MPI
- make mpi
- Launch 4 ZHT servers as
  - mpiexec -np 4 ./zht-mpiserver -z zht.conf -n neighbor.mpi.conf : ./zht-mpibroker -z zht.conf -n neighbor.mpi.conf
- Run your ZHT apps

# How to dev your ZHT apps – run ZHT(cluster mode)

- see iit.datasys.zht-mpi/README

# How to customize ZHT-augment client API

- Declare and define new C++ binding API in < cpp_zhtclient.h > and < cpp_zhtclient.cpp >,
  - Declare and define your operation code in < Const.h > and < Const.cpp >, e.g. Const::*ZSC_OPC_YOURS*, like Const::*ZSC_OPC_LOOKUP*
  - Learn from the existing API, e.g. ZHTClient::lookup, the stack looks like:
    - int ZHTClient::lookup(const char *key, char *result)
    - int ZHTClient::lookup(const string &key, string &result)
    - int ZHTClient::commonOp(const string &opcode, const string &key, const string &val, const string &val2, string &result, int lease)
    - string ZHTClient::commonOpInternal(const string &opcode, const string &key, const string &val, const string &val2, string &result, int lease)
- Declare and define delegation for new API in < c_zhtclientStd.h > and < c_zhtclientStd.cpp >

# How to customize ZHT-augment client API

- Declare and define C binding for new API in < c_zhtclient.h > and < c_zhtclient.cpp >

# How to customize ZHT-augment server implementation

- Declare and define server implementation for new client API in < HTWorker.h> and < HTWorker.cpp >
- Learn from existing impl, e.g. lookup, the stack looks like:
  - string HTWorker::run(const char *buf)
    - Be sure to add the operation dispatch code like:
      - if (zpack.opcode() == Const::*ZSC_OPC_YOURS*)
  - string HTWorker::lookup_shared(const ZPack &zpack)

# How to dev your ZHT apps- thread safe

- ZHT client thread safe
  - ZHT client API is thread-safe at operation level
  - ZHT client API is thread-safe at socket level
  - We will explore making it thread safe at MPI rank level
- ZHT server is thread safe

# How to dev your ZHT apps- configration

- zht.conf
  - PROTOCOL TCP #communication protocol for ZHT client and server
  - PORT 50000 #zhtserver port to listen on, overrides by -p option
  - MSG_MAXSIZE 1000000 #max size of message for a single trip
  - SCCB_POLL_INTERVAL 100 #the interval in milliseconds to resume polling a status of a item
  - INSTANT_SWAP 1 #set if instantly swap in-memory data to disk

# How to dev your ZHT apps-configration

- neighbor.conf, for non-MPI standalone deployment
  - localhost 50000
  - localhost 50001
  - You MUST actually launch two ZHT servers at port 50000 and 50001, otherwise errors prompt.
- neighbor.conf, for non-MPI cluster deployment
  - 192.168.1.100 50000
  - 192.168.1.101 50000

# How to dev your ZHT apps-configration

- neighbor.mpi.conf, for MPI standalone deployment
  - localhost
  - Only localhost for multiple ZHT servers launched
- neighbor.mpi.conf, for MPI cluster deployment, no port needed
  - 192.168.1.100
  - 192.168.1.101
  - 192.168.1.102

# How to dev your ZHT apps-configration

- launch your applications always with zht.conf and neighbor.conf(or neighbor.mpi.conf) as startup arguments
- Be careful the path for configuration files

# How to dev your ZHT apps- TWO special API(s)

- int c_state_change_callback(const char *key, const char *expeded_val, int lease), in C
- int state_change_callback(const string &key, const string &expected_val,int lease), in C++
  - monitor the value change of the key, block or unblock ZHT client
  - EXPECDED_VAL: the value expected to be equal to what is lookuped by the key, if equal, return 0(zero), or keep polling in server-side and block ZHT client
  - LEASE: the lease in milliseconds after which ZHT client will be unblocked.
- See <c_zhtclient_threaded_test.cpp> for C example and <cpp_zhtclient_threaded_test.cpp> for C++ example

# How to dev your ZHT apps- TWO special API(s)

- int c_zht_compare_swap(const char *key, const char *seen_value, const char *new_value, char *value_queried), in C
- int compare_swap(const string &key, const string &seen_val, const string &new_val, string &result)
  - Return o(zero), if SEEN_VALUE equals to value lookuped by the key, and set the value to NEW_VALUE returned
  - Return non-zero, if the above doesn't meet, and VALUE_QUERIED
  - SEEN_VALUE: value expected to be equal to that lookuped by the key
  - NEW_VALUE: if equal, set value to NEW_VALUE
  - VALUE_QUERIED: if equal or not equal, get new value queried
- See <c_zhtclient_lanl_threaded.c> for example

# How to dev your ZHT apps- Prototype

- Prototype by customizing ZHT
  - iit.cs550.pa1
    - https://bitbucket.org/xiaobingo/iit.cs550.pa1

# How to dev your ZHT apps- max size of item

- Max size of item can be transferred by ZHT
  - 6oMB, see the configuration in zht.conf
  - MSG_MAXSIZE 1000000 #max size of message for a single trip

# How to dev your ZHT apps

- Demo

# Questions?

**Tonglin Li, Xiaobing Zhou**
tli13@hawk.iit.edu, xzhou40@hawk.iit.edu

http://datasys.cs.iit.edu/projects/ZHT/
https://bitbucket.org/xiaobingo/iit.datasys.zht-mpi