Q in Every Core

"First step in exploration of protocol-oriented architecture design space"

Jay Han <jay.han@gmail.com>

2015.05.14 KX SF Meetup

Q SWMR

- Single writer
- Multiple reader
- Single writer means concurrency problems are precluded – no deadlocks, no starvation, etc.
- See Apter, Shasha, Whitney paper
- http://cs.nyu.edu/cs/faculty/shasha/papers/hpts.pdf

Q IPC

- Q has IPC built-in.
- Sync call to other q processes on other cores
- Async call possible
- Send data or...
- Query data in another process
- Shared memory?
  - Not included in q
  - Can be done + (precluded concurrency problems + OS integration, portability, tuning, etc. issues)
  - Future reference: SHM by Apter

N-1 cores

- What do N-1 cores do when not reading?
- N is growing larger.
  - In 2015, N≥4 on consumer devices or phones!
  - N≥16, 32, 64, 128 or more for servers.
    - Octo xeon with 18 cores x 2 hyperthreads.
  - N will “double" every few years.
- SWMR → MWMR?

Standard

- pthreads
- Concurrency hard with shared data
- Mix and confusion over control/data
  - Lock over data to effect control
  - Memory-based programming

A Few Caveats

- Do NOT try to circumvent kdb+ I/O (write is still locked).
- Do NOT write to globals from non-main threads.
- Run kdb+ in multithreaded mode or with -p -NEGPORT.

Simple Programming Model

```
/ ping/pong across two threads
pong:{[pipe;args]
   signal[pipe; 0x0]; / signal readiness
   while[not `ping~recv[pipe];/ recv command
      do[1000*1000; md5 string args]];    / “work"
   send[pipe; `pong]}   / report to caller
pong0:new[pong; args] / spawn new thread
send[pong0; `work] / tell pong to work
```

- More pongs can run in parallel, using N cores.
- N.B. pong is just a simple function.

Simple Programming Model

```
func:{[pipe;args]
    signal[pipe; 0x0];
    while[not `stop~recv[pipe]; code[args]];
    send[pipe; `done]}
ctrl:new[func; args]
```

- send[ctrl; `dontstop]
-
- N.B. *func* is a simple function
- N.B. args is a fixed initial condition for func.

Simple Programming Model + 1

```
func:{[pipe;args]
    signal[pipe; 0x0];
    while[not `stop~recv[pipe]; code[args]];
    send[pipe; `done]}
ctrl:new[func; args]
send[ctrl; `dontstop]
```

- N.B. args is a fixed initial condition for func.
- Data flow can happen via Pub/Sub message queue
- Other choices possible

Simpler or Advanced Programming Model

```
compute:{[pipe;thunk]
   signal[pipe; 0x0];
   while[not `quit~recv[pipe];
       // do something with thunk
   send[pipe; `ack]}
ctrl:new[compute; thunk]
send[ctrl; `sync]
```

- N.B. *thunk* can be domain specific protocol:

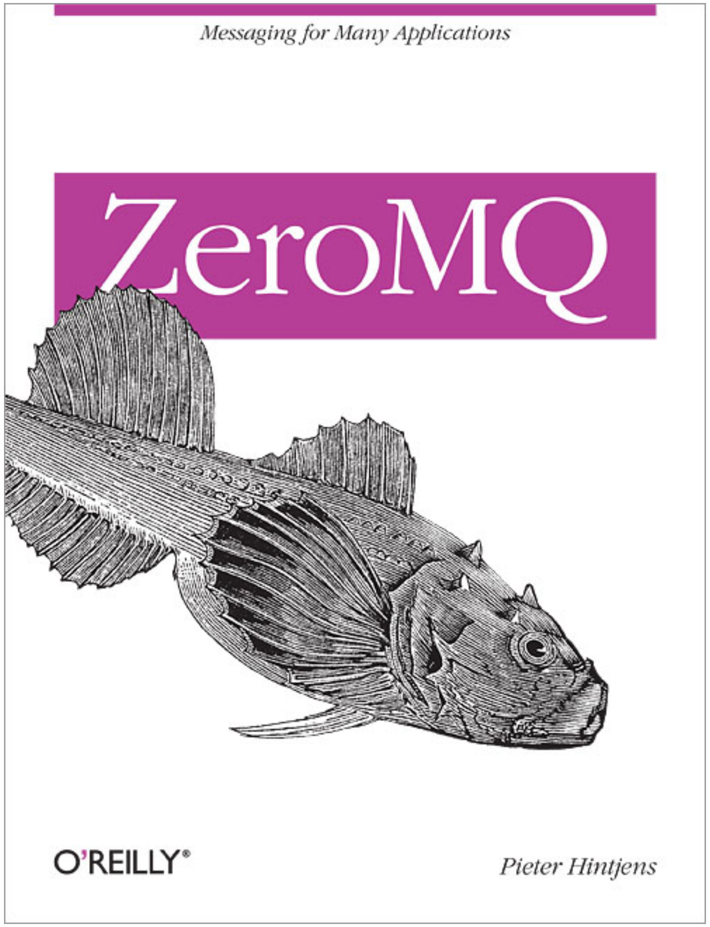
```
   `sync`ack`nack!({sync code};{ack code};{nack code})
```

Communications Across Networks

- Separate control and data flows in design
- Integrate control and data with protocols
- Distribute protocol definition and implementations across threads, processes, hosts over many transports (aka networks)
- Design freedom: both data AND code can move around.

  (Data, lists, tables, dictionaries, and functions or "protocols" are all first-class values in q.)

Distribution



qzmq

- ZeroMQ – "an intelligent transport layer for your distributed apps" http://zeromq.org
- CZMQ – high-level ZeroMQ library in C, including threading http://czmq.zeromq.org
- qzmq – q bindings of CZMQ https://github.com/jaeheum/qzmq
- Open Source under Mozilla Public License v2 (same as CZMQ)
- One more option in your system design space