Work Queueing with Redis.pm

B. Estrade

http://houston.pm.org/

August 8th, 2013

What is Redis?

- a keyed (shared) data structures server
- supports its own protocol
- supports: scalar, hash, list, set
- “in memory” + optional bin logging
- “single threaded, except when it’s not”
- publish/subscribe “channels”

Demo requires set up time if you wish to run it yourself.

Now's a good time to start the Vagrant process if you have not yet done so.

https://github.com/estrabd/houston-pm-redis-talk

Redis.pm

- “official” Redis client for Perl
- wrapper around Redis protocol, methods

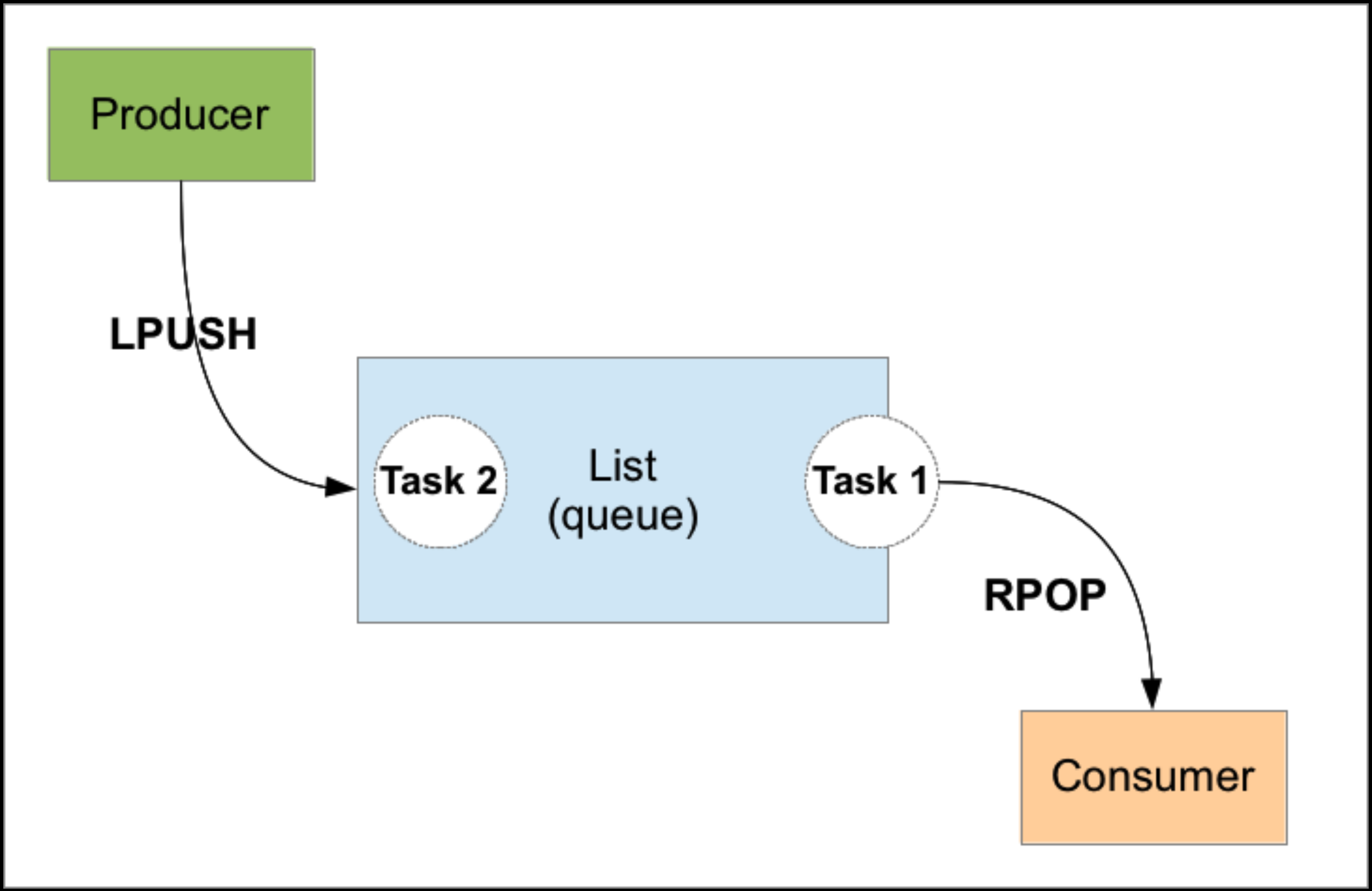
Perl Script

```
use Redis;
my $redis = Redis->new(server =>
'192.168.2.3');
$redis->ping;
```

Telnet Session

```
vagrant@precise64:/vagrant$  telnet 192.168.3.2 6379
Trying 192.168.3.2...
Connected to 192.168.3.2.
Escape character is '^]'.
ping
+PONG
```

A Work Queue (FIFO)



What is Work Queueing? Why?

- a method of distributing tasks to a pool of worker processes
- useful for massively scaling web applications
- decouples requests from resource intensive interactions (e.g., with a DB)
- more secure, workers can be in a private net
- # of workers can be tuned based on load

Redis as a Queue?

- use the “list” data structure

- non-blocking:
  - lpush, rpush, lpop, rpop, rpoplpush*

- blocking
  - blpop, brpop, brpoplpush*

- necessarily implements atomic pop’ing

- other structures can be used for meta data

* provides for “reliable” queues

Why Not MySQL as a Queue

- list operations must be emulated
- inefficient table locking req’d for atomic pops

Why Not Memcached as Queue

- federation would be a nice feature of a queue
- but, memcached supports only scalar key/val
- back to implementing atomic pops (idk how?)
- MemcachedQ, based on MemcachedBD exists, but languishing

Other options

- beanstalkd - not mature, not stable enough
- RabbitMQ - overkill (but not for HA messaging?)
- NoSQL option? Not sure.

Simple Queue Client using Redis.pm

- `submit_task`
- `get_task`
- `bget_task`

Supporting hooks for serialization/deserialization:

- `_encode_task`
- `_decode_task`

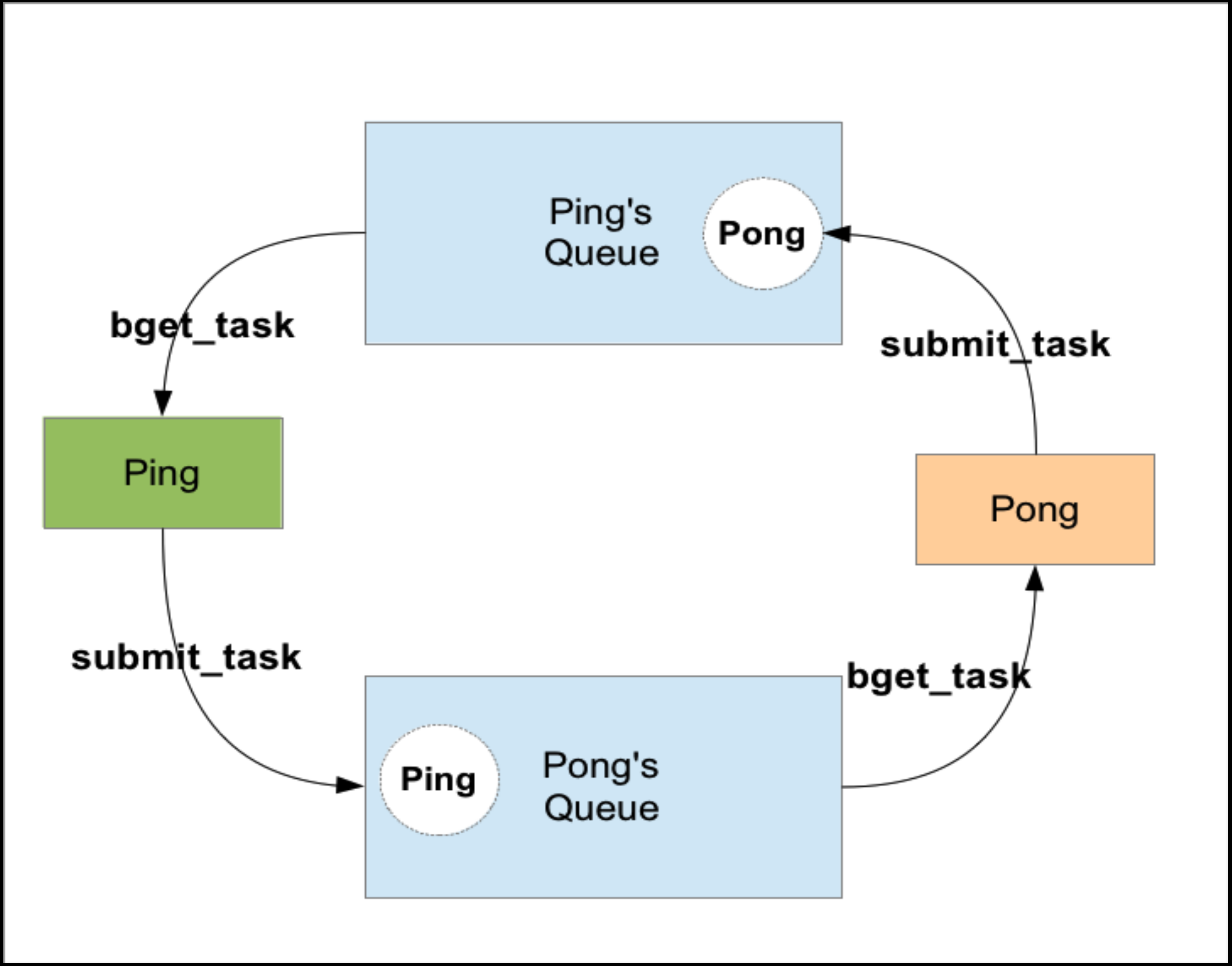
Task.pm

- send/receive blessed Task references
- fields: `type` ($pkg), `id`, `payload` (‘HASH’)
- Sending:
  - serialize blessed ref (encode as JSON)
  - lpush string onto Redis list
- Receiving
  - pop off of list, parse decode with JSON::XS
  - re-bless with $task->{type}

Ping Pong

- Synchronize
- Ponger waits for Ping
- Pinger sends Ping, waits for ACK via Pong
- Repeat in turn until $rounds complete

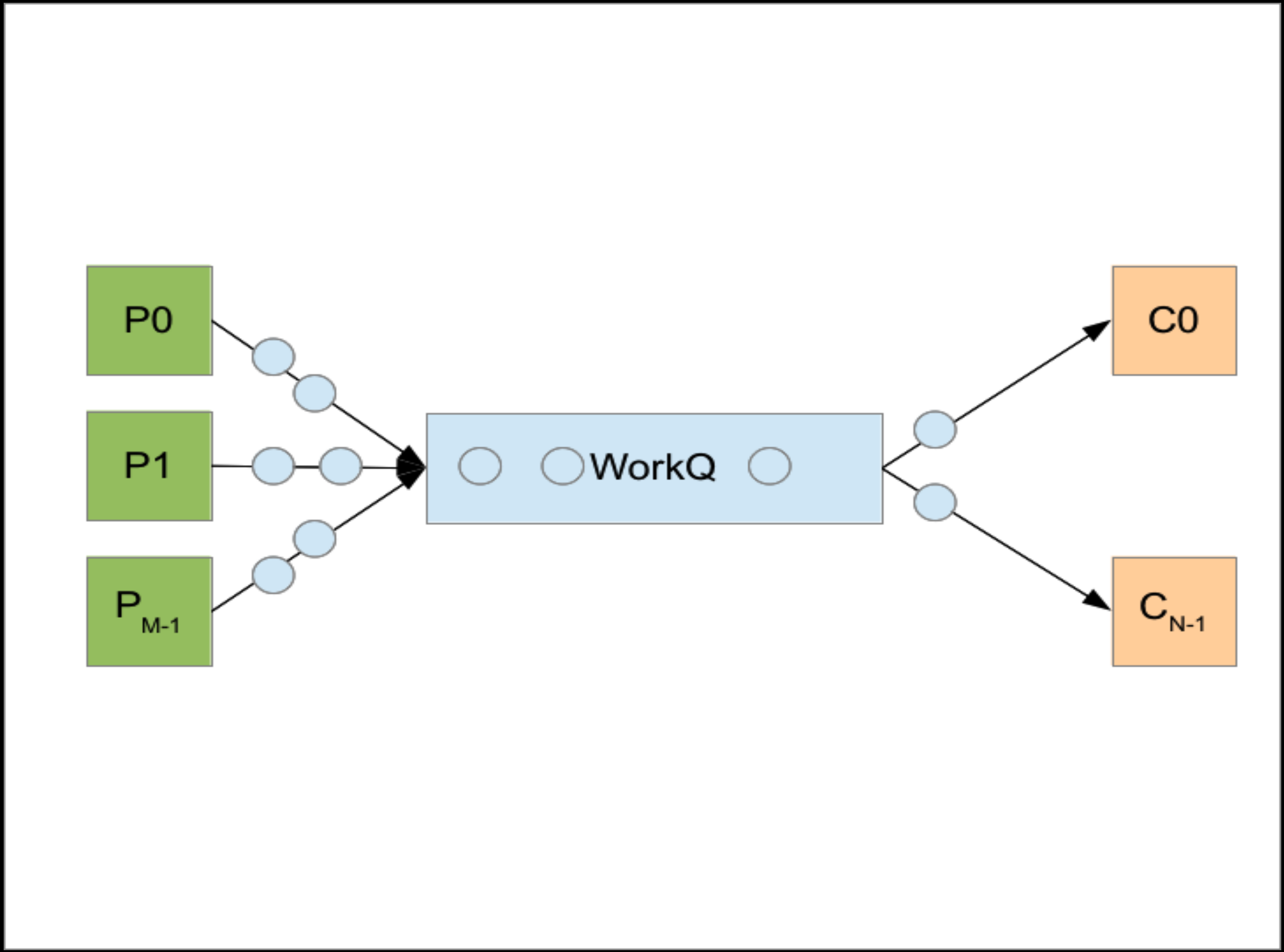
Ping Pong


Ping's Queue
Pong
bget_task
submit_task
Ping
Pong
submit_task
bget_task
Ping
Pong's Queue

MxN Producer-Consumer, 1 Queue

- M Producers
- N Consumers
- Producers “fire and forget” - asynchronous task submit
- Consumers pull from Queue in first come first serve order

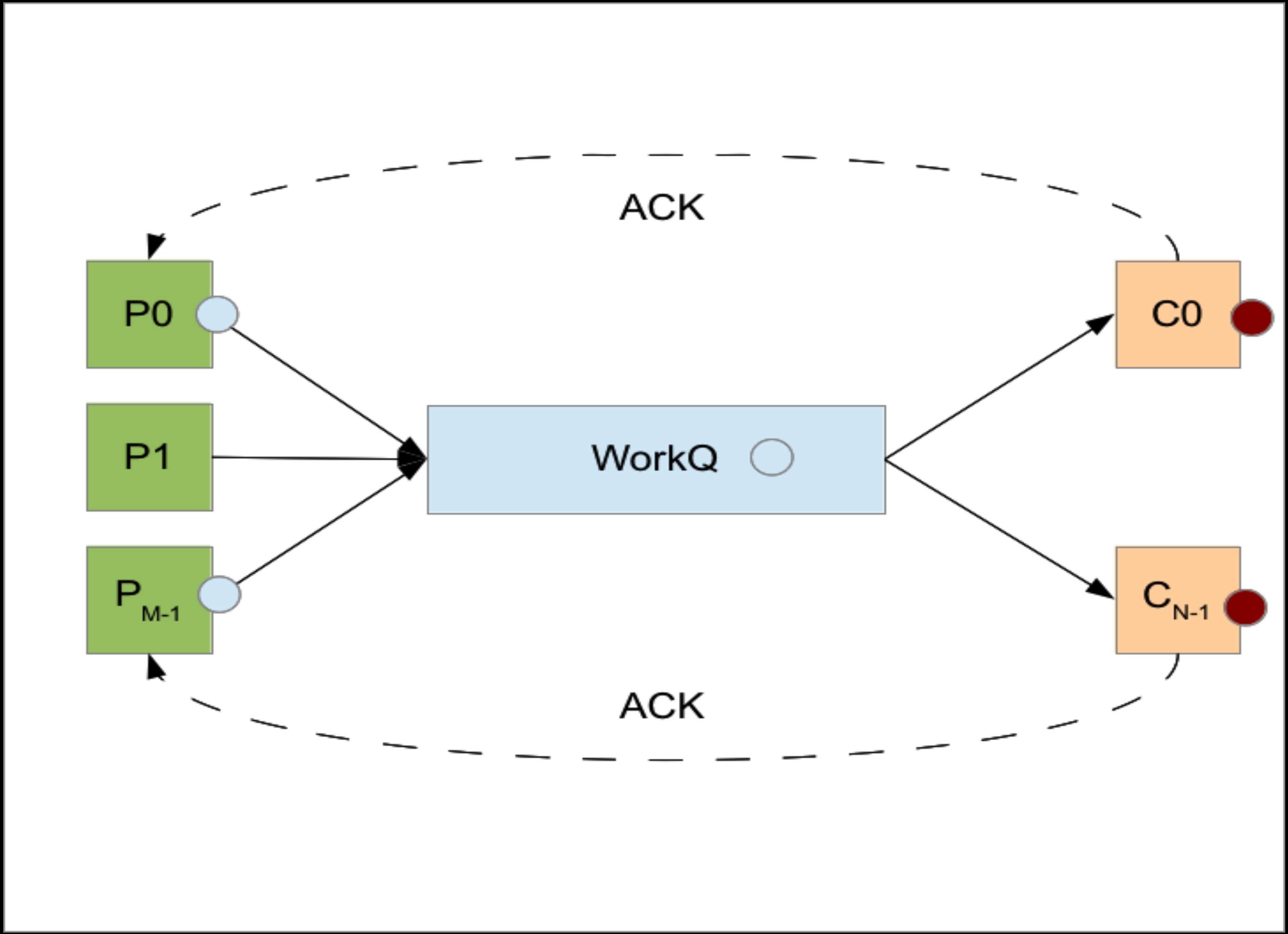
MxN Producer-Consumer, 1 Queue



Sync’d MxN Produce-Consume

- M Producers
- N Consumers
- Producer blocks on submit until it gets a response from whichever Consumer got it
- Requires use of “private” queues for ACKs

Sync'd MxN Produce-Consume



Other Patterns

- *Scaling* out synchronous produce/consume
  - M producers, N consumers, P queues
  - best implemented with forking consumers,
  - with each child watching a different queue
- Circuitous messaging and routing
  - tasks beget other tasks to other consumers
  - chain reaction like
  - heavy use of private queus
  - useful for something?

Redis Failover Options?

- Master/Slave replication via binary log
- Redis HA Cluster in development
- Craigslist uses sharding & “federated” Redis, which is not supported natively (here & here)
- Could use a pool of Redis instances/queues
    - Sharing/Federation is often overkill for *just* queuing
    - Producers will try to submit until successful
    - Available queue assumed to have at least one consumer
    - Also implement a “reliable” protocol (using ACKs, etc)

Note on General Messaging

- Redis is not the best foundation for “reliable” 2-way messaging
- Redis “cluster”, sharding/federating is best here for reliability
- RabbitMQ seems to a fine, if heavy solution for this
- ...which segues nicely into Failover

Tips

- treat Redis instances as ephemeral
- turn off binary logging for high throughput
- not convinced it's a good durable data store
- Redis seems highly stable/reliable
- 1 machine can support many Redis daemons
- it's smart to wrap blocking calls with `alarm`

Demo

- Reproducible using Vagrant manifest (KMA, Murphy! ;)
- Ping Pong
- Asynchronous M Producer x N Consumer
- Synchronous M Producer x N Consumer

Conclusion

- Redis shines for work queueing
- Lots of potential to make w-q patterns scale
- Similarly, it can be highly available/reliable
- Open Questions -
    - leveraging other data structures for meta data
    - e.g., implement “queue” state -
        - accepting
        - draining
        - offline

Resources

- https://github.com/estrabd/houston-pm-redis-talk
- https://github.com/melo/perl-redis
- http://blog.zawodny.com/2011/02/26/redis-sharding-at-craigslist/
- Vagrant
- http://houston.pm.org/
- http://www.cpanel.net