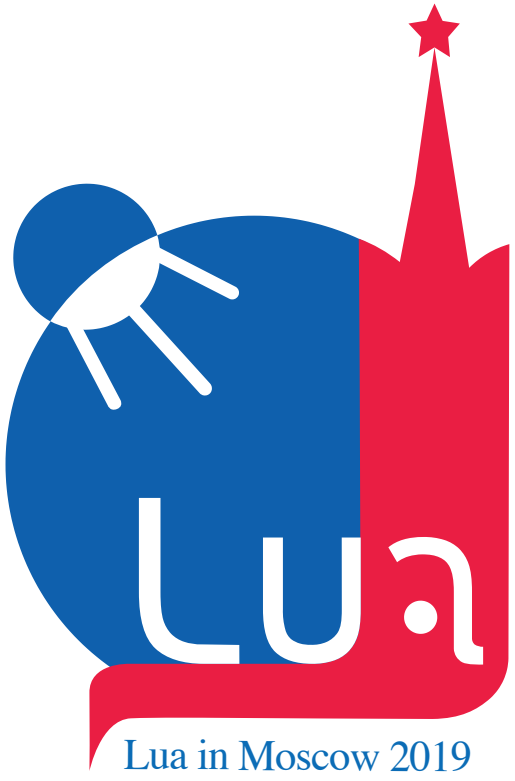


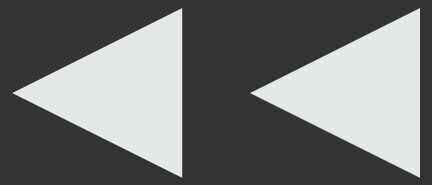
Tarantool usecases for rich applications

Mons Anderson

Mail.ru Cloud Solutions

IPONWEB

TAMASHI

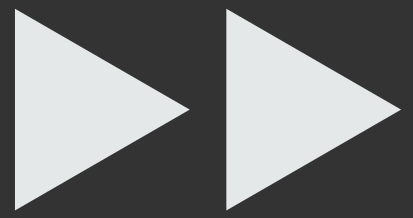


kostja: Implement beanstalkd-like queues.
`f879dfb` on *Oct 3, 2012*

```lua
box.queue = {}
local box_queue_id = 0

local c_id = 0
local c_fid = 1
local c_state = 2
local c_count = 3

local function lshift(number, bits)
   return number * 2^32
end

local function rshift(number, bits)
   return number / 2^32
end

function box.queue.id(delay)
   box_queue_id = box_queue_id + 2^12
   return tonumber64(lshift(os.time() + delay) + bit.tobit(box_queue_id))
end

function box.queue.put(sno, delay, ...)
   sno, delay = tonumber(sno), tonumber(delay)
   local id = box.queue.id(delay)
   return box.insert(sno, id, 0, 'R', 0, ...)
end

function box.queue.push(sno, ...)
   sno = tonumber(sno)
   local minid = box.unpack('l', box.space[sno].index[0]:min()[0])
   local id = lshift(rshift(minid) - 1)
   return box.insert(sno, id, 0, 'R', 0, ...)
end

function box.queue.delete(sno, id)
   sno, id = tonumber(sno), tonumber64(id)
   return box.space[sno]:delete(id)
end

function box.queue.take(sno, timeout)
   sno, timeout = tonumber(sno), tonumber(timeout)
   if timeout == nil then
      timeout = 60*60*24*365
   end
   local task = nil
   while true do
      task = box.select(sno, 1, 0)
      if task ~= nil then break end
      if timeout > 0 then
         box.fiber.sleep(1)
         timeout = timeout - 1
      else
         return nil
      end
   end
   return box.update(sno, task[0], "=p=p+p",
                  c_fid, box.fiber.id(),
                  c_state, 'T', -- taken
                  c_count, 1)
end
```

```lua
local function consumer_find_task(sno, id)
   local task = box.select(sno, 0, id)
   if task == nil then
      error("task not found")
   end
   if task[c_fid] ~= box.fiber.id() then
      error("the task does not belong to the consumer")
   end
   return task
end

function box.queue.ack(sno, id)
   sno, id = tonumber(sno), tonumber64(id)
   local task = consumer_find_task(sno, id)
   box.space[sno]:delete(id)
end

function box.queue.release(sno, id, delay)
   sno, id, delay = tonumber(sno), tonumber64(id), tonumber(delay)
   local task = consumer_find_task(sno, id)
   local newid = box.queue.id(delay)
   return box.update(sno, id, "=p=p=p", c_id, newid, c_fid, 0, c_state, 'R')
end

local function queue_expire(sno)
   box.fiber.detach()
   box.fiber.name("box.queue "..sno)
   local idx = box.space[sno].index[1].idx
   while true do
      local i = 0
      for it, tuple in idx.next, idx, box.fiber.id() do
         fid  = tuple[c_fid]
         if box.fiber.find(fid) == nil then
            local id = tuple[0]
            box.update(sno, id, "=p=p", c_fid, 0, c_state, 'R')
            break
         end
         i = i + 1
         if i == 100 then
            box.fiber.sleep(1)
            i = 0
         end
      end
      box.fiber.sleep(1)
   end
end

function box.queue.start(sno)
   sno = tonumber(sno)
   box.fiber.resume(box.fiber.create(queue_expire), sno)
end
```

kostja: Implement beanstalkd-like queues.
`f879dfb` on *Oct 3, 2012*



init.lua

```
box.cfg{}
```

init.lua

```
box.cfg{}
```

```
box.schema.create_space('queue',{})
```

init.lua

```
box.cfg{}
```

```
box.schema.create_space('queue',{
   if_not_exists = true;
})
```

init.lua

```
box.cfg{}
```

```
box.schema.create_space('queue',{
   format = {
      { name = 'id';     type = 'number' },
      { name = 'status'; type = 'string' },
      { name = 'data';   type = '*'      },
   };
   if_not_exists = true;
})
```

init.lua

```
box.cfg{}
```

```
box.schema.create_space('queue',{
   format = {
      { name = 'id';     type = 'number' },
      { name = 'status'; type = 'string' },
      { name = 'data';   type = '*'      },
   };
   if_not_exists = true;
})
```

```
box.space.queue:create_index('primary', {
   parts = {1,'number'};
   if_not_exists = true;
})
```

init.lua

```
box.cfg{}
```

```
box.schema.create_space('queue',{
   format = {
      { name = 'id';     type = 'number' },
      { name = 'status'; type = 'string' },
      { name = 'data';   type = '*'      },
   };
   if_not_exists = true;
})
```

```
box.space.queue:create_index('primary', {
   parts = {1,'number'};
   if_not_exists = true;
})
```

init.lua

```
box.cfg{}
```

```
box.schema.create_space('queue',{
   format = {
      { name = 'id';     type = 'number' },
      { name = 'status'; type = 'string' },
      { name = 'data';   type = '*'      },
   };
   if_not_exists = true;
})
```

```
box.space.queue:create_index('primary', {
   parts = {1,'number'};
   if_not_exists = true;
})
```

```
require'console'.start()
os.exit()
```

init.lua

```
$ tarantool init.lua
```

```
main/101/init.lua C> Tarantool 1.9.0-83-g4158922
main/101/init.lua C> log level 5
main/101/init.lua I> mapping 268435456 bytes for memtx tuple arena...
main/101/init.lua I> mapping 134217728 bytes for vinyl tuple arena...
main/101/init.lua I> initializing an empty data directory
snapshot/101/main I> saving snapshot ./00000000000000000000.snap.inprogress
snapshot/101/main I> done
main/101/init.lua I> ready to accept requests
main/104/checkpoint_daemon I> started
tarantool>
```

Demo

queue.lua

```
require'strict'.on()
```

queue.lua

```
require'strict'.on()
```

```
queue = {}

function queue.put(...)

end

function queue.take(...)

end
```

queue.lua

```
require'strict'.on()
```

```
STATUS = {}
STATUS.READY = 'R'
STATUS.TAKEN = 'T'
```

```
queue = {}

function queue.put(...)

end

function queue.take(...)

end
```

queue.put

```
function queue.put(...)
    local id = gen_id()
    return box.space.queue:insert{ id, STATUS.READY, ... }
end
```

queue.put

```
local clock = require 'clock'
local function gen_id()
   local new_id = clock.realtime64()/1e3
   while box.space.queue:get(new_id) do
      new_id = new_id + 1
   end
   return new_id
end
```

```
function queue.put(...)
   local id = gen_id()
   return box.space.queue:insert{ id, STATUS.READY, ... }
end
```

queue.put

```
tarantool> queue.put("aaa")
---
- [1529262394933356, 'R', 'aaa']
...

tarantool> queue.put("my","data")
---
- [1529262398900772, 'R', 'my', 'data']
...
tarantool> queue.put({complex = {struct = 1}})
---
- [1529262413142216, 'R', {'complex': {'struct': 1}}]
...
```

queue.put

```
tarantool> box.space.queue:select()
---
- - [1529262394933356, 'R', 'aaa']
  - [1529262398900772, 'R', 'my', 'data']
  - [1529262413142216, 'R', {'complex': {'struct': 1}}]
...
```

queue.take

```
box.space.queue:create_index('status', {
   parts = {2, 'string', 1, 'number'};
   if_not_exists = true;
})
```

queue.take

```
box.space.queue:create_index('status', {
   parts = {2, 'string', 1, 'number'};
   if_not_exists = true;
})
```

```
function queue.take(...)
   for _,t in
      box.space.queue.index.status
      :pairs({ STATUS.READY }, { iterator=box.index.EQ })
   do
      return box.space.queue:update({t.id},{
         {'=', 2, STATUS.TAKEN }
      })
   end
   return
end
```

queue.take

```
box.space.queue:create_index('status', {
   parts = {2, 'string', 1, 'number'};
   if_not_exists = true;
})
```

```
function queue.take(...)
   for _,t in
      box.space.queue.index.status
      :pairs({ STATUS.READY }, { iterator=box.index.EQ })
   do
      return box.space.queue:update({t.id},{
         {'=', 2, STATUS.TAKEN }
      })
   end
   return
end
```

queue.take

```
local F = {
   id     = 1;
   status = 2;
   data   = 3;
}
```

```
function queue.take(...)
   for _,t in
      box.space.queue.index.status
      :pairs({ STATUS.READY }, { iterator=box.index.EQ })
   do
      return box.space.queue:update({t.id},{
         {'=', F.status, STATUS.TAKEN }
      })
   end
   return
end
```

put + take

```
tarantool> queue.put("task 1")
---
- [1529262869440082, 'R', 'task 1']
...
```

put + take

```
tarantool> queue.put("task 1")
---
- [1529262869440082, 'R', 'task 1']
...
```

```
tarantool> queue.take()
---
- [1529262869440082, 'T', 'task 1']
...
```

put + take

```
tarantool> queue.put("task 1")
---
- [1529262869440082, 'R', 'task 1']
...
```

```
tarantool> queue.take()
---
- [1529262869440082, 'T', 'task 1']
...
```

```
tarantool> queue.take()
---
...
```

Demo

Task return: release + ack

```
function queue.ack(id)
    local t = assert(box.space.queue:get{id},"Task not exists")
    if t and t.status == STATUS.TAKEN then
        return box.space.queue:delete{t.id}
    else
        error("Task not taken")
    end
end

function queue.release(id)
    local t = assert(box.space.queue:get{id},"Task not exists")
    if t and t.status == STATUS.TAKEN then
        return box.space.queue:update({t.id},{{'=', F.status, STATUS.READY }})
    else
        error("Task not taken")
    end
end
```

```
tarantool> queue.put("task 1")
- [1529263926863733, 'R', 'task 1']

tarantool> queue.put("task 2")
- [1529263928104079, 'R', 'task 2']
```

```
tarantool> queue.put("task 1")
- [1529263926863733, 'R', 'task 1']

tarantool> queue.put("task 2")
- [1529263928104079, 'R', 'task 2']
```

```
tarantool> queue.take()
- [1529263926863733, 'T', 'task 1']
```

```
tarantool> queue.put("task 1")
- [1529263926863733, 'R', 'task 1']

tarantool> queue.put("task 2")
- [1529263928104079, 'R', 'task 2']
```

```
tarantool> queue.take()
- [1529263926863733, 'T', 'task 1']
```

```
tarantool> queue.release(1529263926863733)
- [1529263926863733, 'R', 'task 1']
```

```
tarantool> queue.put("task 1")
- [1529263926863733, 'R', 'task 1']

tarantool> queue.put("task 2")
- [1529263928104079, 'R', 'task 2']
```

```
tarantool> queue.take()
- [1529263926863733, 'T', 'task 1']
```

```
tarantool> queue.release(1529263926863733)
- [1529263926863733, 'R', 'task 1']
```

```
tarantool> queue.take()
- [1529263926863733, 'T', 'task 1']

tarantool> queue.ack(1529263926863733)
- [1529263926863733, 'T', 'task 1']
```

```
tarantool> queue.put("task 1")
- [1529263926863733, 'R', 'task 1']

tarantool> queue.put("task 2")
- [1529263928104079, 'R', 'task 2']
```

```
tarantool> queue.take()
- [1529263926863733, 'T', 'task 1']
```

```
tarantool> queue.release(1529263926863733)
- [1529263926863733, 'R', 'task 1']
```

```
tarantool> queue.take()
- [1529263926863733, 'T', 'task 1']

tarantool> queue.ack(1529263926863733)
- [1529263926863733, 'T', 'task 1']
```

```
tarantool> queue.take()
- [1529263928104079, 'T', 'task 2']
```

Consumer

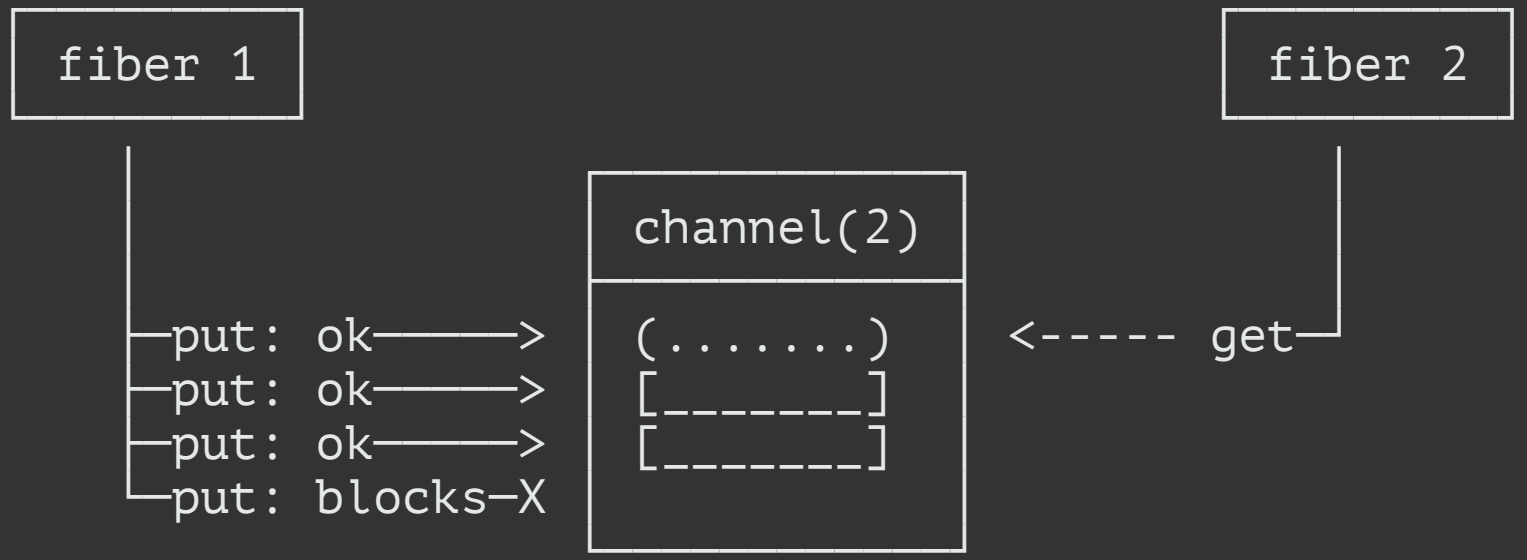
```
while true do
   local task = queue.take()
   if task then
      -- ...
   end
end
```

Consumer

```
while true do
   local task = queue.take()
   if task then
      -- ...
   end
end
```

10K RPS of CPU heating

Channel



Demo: channels

Channel

```
local fiber = require 'fiber'


function queue.take(timeout)
    if not timeout then timeout = 0 end
    local now = fiber.time()
    local found
    while not found do
        found = box.space.queue.index.status
            :pairs({STATUS.READY},{ iterator = box.index.EQ }):nth(1)
        if not found then
            local left = (now + timeout) - fiber.time()
            if left <= 0 then return end

        end
    end
    return box.space.queue:update({found.id},
        {{'=', F.status, STATUS.TAKEN }})
end
```

Channel

```
local fiber = require 'fiber'
local wait = fiber.channel(0)

function queue.take(timeout)
    if not timeout then timeout = 0 end
    local now = fiber.time()
    local found
    while not found do
        found = box.space.queue.index.status
            :pairs({STATUS.READY},{ iterator = box.index.EQ }):nth(1)
        if not found then
            local left = (now + timeout) - fiber.time()
            if left <= 0 then return end
            wait:get(left)
        end
    end
    return box.space.queue:update({found.id},
        {{'=', F.status, STATUS.TAKEN }})
end
```

Channel

```
tarantool> do
   fiber.create(function()
      fiber.sleep(0.1)
      queue.put("task 1")
   end)

   local start = fiber.time()
   return queue.take(3), {wait = fiber.time()-start}
end
```

Channel

```
tarantool> do
   fiber.create(function()
      fiber.sleep(0.1)
      queue.put("task 1")
   end)

   local start = fiber.time()
   return queue.take(3), {wait = fiber.time()-start}
end
```

```
---
- [1529264646170985, 'T', 'task 1']
- wait: 3.0040209293365
...
```

Channel

```
function queue.put(...)
    local id = gen_id()



    return box.space.queue:insert{ id, STATUS.READY, ... }
end
```

Channel

```
function queue.put(...)
    local id = gen_id()

    wait:put(true)

    return box.space.queue:insert{ id, STATUS.READY, ... }
end
```

Channel

```
function queue.put(...)
    local id = gen_id()

    wait:put(true, 0)

    return box.space.queue:insert{ id, STATUS.READY, ... }
end
```

Channel

```
function queue.put(...)
    local id = gen_id()

    if wait:has_readers() then
        wait:put(true,0)
    end

    return box.space.queue:insert{ id, STATUS.READY, ... }
end
```

Channel

```
tarantool> do
   fiber.create(function() fiber.sleep(0.1) queue.put("task 1") end)
   local start = fiber.time()
   return queue.take(3), {wait = fiber.time()-start}
end
```

```
---
- [1529264977777259, 'T', 'task 1']
- wait: 0.10470604896545
...
```

Channel

```
tarantool> do
   fiber.create(function() fiber.sleep(0.1) queue.put("task 1") end)
   local start = fiber.time()
   return queue.take(3), {wait = fiber.time()-start}
end
```

```
---
- [1529264977777259, 'T', 'task 1']
- wait: 0.10470604896545
...
```

```
tarantool> do local start = fiber.time() return
   queue.take(3), {wait = fiber.time()-start} end
```

```
---
- null
- wait: 3.0040738582611
...
```

Networking

server.lua

```
box.cfg{
   listen = 'localhost:3311'
}
```

server.lua

```
box.cfg{
   listen = 'localhost:3311'
}
```

```
box.schema.user.grant('guest', 'super')
```

server.lua

```
box.cfg{
   listen = 'localhost:3311'
}
```

```
box.once('access:v1', function()
   box.schema.user.grant('guest', 'super')
end)
```

producer.lua

```
local netbox = require 'net.box'
local conn = netbox.connect('localhost:3311')
local yaml = require 'yaml'

local res = conn:call('queue.put',{unpack(arg)})
print(yaml.encode(res))

conn:close()
```

producer.lua

```
local netbox = require 'net.box'
local conn = netbox.connect('localhost:3311')
local yaml = require 'yaml'

local res = conn:call('queue.put',{unpack(arg)})
print(yaml.encode(res))

conn:close()
```

```
$ tarantool producer.lua "test task"
--- [1529277044524945, 'R', 'test task']
...
```

consumer.lua

```lua
local netbox = require 'net.box'
local conn = netbox.connect('localhost:3311')
local yaml = require 'yaml'

while true do
   local task = conn:call('queue.take',{1})
   print(yaml.encode(task))
   if task then
      conn:call('queue.release',{task.id})
   end
end
```

consumer.lua

```
local netbox = require 'net.box'
local conn = netbox.connect('localhost:3311')
local yaml = require 'yaml'

while true do
   local task = conn:call('queue.take',{1})
   print(yaml.encode(task))
   if task then
      conn:call('queue.release',{task.id})
   end
end
```

```
tarantool consumer.lua
--- [1529277044524945, 'T', 'test task']
...
```

consumer.lua

```
local netbox = require 'net.box'
local conn = netbox.connect('localhost:3311')
local yaml = require 'yaml'

while true do
   local task = conn:call('queue.take',{1})
   print(yaml.encode(task))
   if task then
      conn:call('queue.release',{task.id})
   end
end
```

```
tarantool consumer.lua
--- [1529277044524945, 'T', 'test task']
...
```

```
Invalid key part count in an exact match (expected 1, got 0)
```

"Frozen" tasks

```
$ tarantool consumer.lua
--- null
...

--- null
...
```

"Frozen" tasks

```
$ tarantool consumer.lua
--- null
...

--- null
...
```

```
tarantool> box.space.queue:select{}
---
- - [1529277323023563, 'T', 'test task']
...
```

Triggers

```
local log = require 'log'
box.session.on_connect(function()
   log.info("connected %s:%s from %s", box.session.id(),
      box.session.user(), box.session.peer())
end)

box.session.on_disconnect(function()
   log.info("disconnected %s:%s from %s", box.session.id(),
      box.session.user(), box.session.peer())
end)
```

Triggers

```
local log = require 'log'
box.session.on_connect(function()
   log.info("connected %s:%s from %s", box.session.id(),
      box.session.user(), box.session.peer())
end)

box.session.on_disconnect(function()
   log.info("disconnected %s:%s from %s", box.session.id(),
      box.session.user(), box.session.peer())
end)
```

```
main/103/main I> connected 2:guest from 127.0.0.1:62223
main/103/main I> disconnected 2:guest from nil
```

Triggers

```
local log = require 'log'
box.session.on_connect(function()
   log.info("connected %s:%s from %s", box.session.id(),
      box.session.user(), box.session.peer())
end)

box.session.on_disconnect(function()
   log.info("disconnected %s:%s from %s", box.session.id(),
      box.session.user(), box.session.peer())
end)
```

```
main/103/main I> connected 2:guest from 127.0.0.1:62223
main/103/main I> disconnected 2:guest from nil
```

WTF `nil` instead of peer?

Hack peer info

```
local log = require 'log'
box.session.on_connect(function()
   log.info("connected %s:%s from %s", box.session.id(),
      box.session.user(), box.session.peer())
   box.session.storage.peer = box.session.peer()
end)

box.session.on_disconnect(function()
   log.info("disconnected %s:%s from %s", box.session.id(),
      box.session.user(), box.session.storage.peer)
end)
```

```
main/103/main I> connected 2:guest from 127.0.0.1:49665
main/103/main I> disconnected 2:guest from 127.0.0.1:49665
```

Task ownership

```
queue.taken = {};
queue.bysid = {};

function queue.take(timeout)
    -- ...
    local sid = box.session.id()
    log.info("Register %s by %s",found.id, sid)

    queue.bysid[ sid ] = queue.bysid[ sid ] or {}
    queue.taken[ found.id ] = sid
    queue.bysid[ sid ][ found.id ] = true

    return box.space.queue:update({found.id},{{'=', F.status, STATUS.TAKEN }}
end
```

Task ownership

```
function queue.get_task(key)
    if not key then error("Task id required",2) end
    local t = box.space.queue:get{key}
    if not t then
        error(string.format( "Task {%s} was not found", key ),2)
    end
    if not queue.taken[key] then
        error(string.format( "Task %s not taken by any", key ),2)
    end
    if queue.taken[key] ~= box.session.id() then
        error(string.format( "Task %s taken by %d. Not you (%d)",
            key, queue.taken[key], box.session.id() ),2)
    end
    return t
end
```

Task ownership

```
function queue.ack(id)
    local t = queue.get_task(id)
    return box.space.queue:delete{t.id}
end

function queue.release(id)
    local t = queue.get_task(id)
    if queue.wait:has_readers() then queue.wait:put(true,0) end
    return box.space.queue:update({t.id},{{'=', F.status, STATUS.READY }})
end
```

Consumer

```
tarantool consumer.lua
--- [1529278401159819, 'T', 'test task']
...

server.lua:122: Task id required
```

Consumer

```
tarantool consumer.lua
--- [1529278401159819, 'T', 'test task']
...

server.lua:122: Task id required
```

```
--- consumer.lua
   if task then
-        conn:call('queue.release',{ task.id })
+        conn:call('queue.release',{ task[1] })
   end
```

Consumer

```
tarantool consumer.lua
--- [1529278401159819, 'T', 'test task']
...

server.lua:122: Task id required
```

```
--- consumer.lua
   if task then
-        conn:call('queue.release',{ task.id })
+        conn:call('queue.release',{ task[1] })
   end
```

```
--- [1529278570209869, 'T', 'test task']
...

server.lua:123: Task 1529278570209869ULL not taken by any
```

WTF, Lua?

```
tarantool> t = {}
tarantool> t[1] = 1
tarantool> t[1LL] = 2
tarantool> t[1ULL] = 3
tarantool> t['1'] = 4
tarantool> t
```

WTF, Lua?

```
tarantool> t = {}
tarantool> t[1] = 1
tarantool> t[1LL] = 2
tarantool> t[1ULL] = 3
tarantool> t['1'] = 4
tarantool> t
```

```
---
- 1: 1
  1: 3
  1: 2
  '1': 4
...
```

WTF, Lua?

```
tarantool> t = {}
tarantool> t[1] = 1
tarantool> t[1LL] = 2
tarantool> t[1ULL] = 3
tarantool> t['1'] = 4
tarantool> t
```

```
---
- 1: 1
  1: 3
  1: 2
  '1': 4
...
```

```
tarantool> return t[1],
                  t['1'],
                  t[1LL],
                  t[1ULL]
```

WTF, Lua?

```
tarantool> t = {}
tarantool> t[1] = 1
tarantool> t[1LL] = 2
tarantool> t[1ULL] = 3
tarantool> t['1'] = 4
tarantool> t
```

```
---
- 1: 1
  1: 3
  1: 2
  '1': 4
...
```

```
tarantool> return t[1],
                  t['1'],
                  t[1LL],
                  t[1ULL]
```

```
---
- 1
- 4
- null
- null
...
```

Pack key

```lua
local msgpack = require 'msgpack'

local function keypack( key )
   return msgpack.encode(key)
end

local function keyunpack( data )
   return msgpack.decode(data)
end
```

Pack key

```
function queue.take(timeout)
    -- ...
    local sid = box.session.id()
    local packid = keypack(found.id)
    log.info("Register %s by %s",found.id, sid)
    queue.bysid[ sid ] = queue.bysid[ sid ] or {}
    queue.taken[ packid ] = sid
    queue.bysid[ sid ][ packid ] = true
```

Pack key

```
function queue.get_task(id)
    if not id then error("Task id required",2) end
    local key = tonumber64(id)
    if not key then error(string.format("Task id '%s' is wrong",id),2) end
    local t = box.space.queue:get{key}
    if not t then
        error(string.format( "Task {%s} was not found", key ),2)
    end
    local packid = keypack(t.id)
    if not queue.taken[packid] then
        error(string.format( "Task %s not taken by any", key ),2)
    end
    if queue.taken[packid] ~= box.session.id() then
        error(string.format( "Task %s taken by %d. Not you (%d)",
            key, queue.taken[packid], box.session.id() ),2)
    end
    return t, packid
end
```

Pack key

```
function queue.release(id)
    local t, packid = queue.get_task(id)
    queue.taken[ packid ] = nil
    queue.bysid[ box.session.id() ][ packid ] = nil
    ...
end

function queue.ack(id)
    local t, packid = queue.get_task(id)
    queue.taken[ packid ] = nil
    queue.bysid[ box.session.id() ][ packid ] = nil
    ...
end
```

Autorelease by disconnect

```
box.session.on_disconnect(function()
   log.info("disconnected %s:%s from %s", box.session.id(),
      box.session.user(), box.session.storage.peer)

   local sid = box.session.id()
   local bysid = queue.bysid[ sid ]
   if bysid then
      for packid in pairs(bysid) do
         local key = keyunpack(packid)
         log.info("Autorelease %s by disconnect", key);
         queue.release( key )
      end
      queue.bysid[ sid ] = nil
   end
end)
```

Autorelease at start

```
while true do
   local t = box.space.queue.index.status:pairs({STATUS.TAKEN}):nth(1)
   if not t then break end
   box.space.queue:update({t.id},{{'=', F.status, STATUS.READY }})
   log.info("Autireleased %s at start", t.id)
end
```

Improve

Delayed processing

```
box.schema.create_space('queue',{
   format = {
      { name = 'id';     type = 'number' },
      { name = 'status'; type = 'string' },
      { name = 'runat';  type = 'number' },
      { name = 'data';   type = '*'      },
   };
   if_not_exists = true;
})
```

Delayed processing

```
box.schema.create_space('queue',{
   format = {
      { name = 'id';     type = 'number' },
      { name = 'status'; type = 'string' },
      { name = 'runat';  type = 'number' },
      { name = 'data';   type = '*'      },
   };
   if_not_exists = true;
})
```

```
box.space.queue:create_index('runat', {
   parts = {3, 'number', 1, 'number'};
   if_not_exists = true;
})
```

Delayed processing

```
box.schema.create_space('queue',{
   format = {
      { name = 'id';     type = 'number' },
      { name = 'status'; type = 'string' },
      { name = 'runat';  type = 'number' },
      { name = 'data';   type = '*'      },
   };
   if_not_exists = true;
})
```

```
box.space.queue:create_index('runat', {
   parts = {3, 'number', 1, 'number'};
   if_not_exists = true;
})
```

```
STATUS.WAITING = 'W'
```

Delayed put

```
function queue.put(data,opts)
    local id = gen_id()
    local runat = 0
    local status = STATUS.READY
    if opts and opts.delay then
        runat = clock.realtime() + tonumber(opts.delay)
        status = STATUS.WAITING
    else
        if queue.wait:has_readers() then queue.wait:put(true,0) end
    end
    return box.space.queue:insert{ id, status, runat, unpack(data) }
end
```

Background actions: fibers

Demo

Background actions: fibers

```
queue.runat = fiber.create(function()
   fiber.name('queue.runat')
   while true do
      local remaining
      -- ...
      if not remaining or remaining > 1 then remaining = 1 end
      fiber.sleep(remaining)
   end
end)
```

Background actions: fibers

```
local remaining
local now = clock.realtime()
for _,t in box.space.queue.index.runat:pairs({0},
   { iterator = box.index.GT }) do

   if t.runat > now then
      remaining = t.runat - now
      break
   else
      ---
   end
end
if not remaining or remaining > 1 then remaining = 1 end
```

Background actions: fibers

```
        if t.runat > now then
            remaining = t.runat - now
            break
        else
            if t.status == STATUS.WAITING then
                log.info("Runat: W->R %s",t.id)
                if queue.wait:has_readers() then queue.wait:put(true,0) end
                box.space.queue:update({t.id},{
                    {'=', F.status, STATUS.READY },
                    {'=', F.runat, 0 },
                })
            else
                log.error("Runat: bad status %s for %s", t.status, t.id)
                box.space.queue:update({t.id},{{'=', F.runat, 0 }})
            end
        end
```

Background actions: fibers

```
queue.runat = fiber.create(function()
   fiber.name('queue.runat')
   while true do
      local remaining
      local now = clock.realtime()
      for _,t in box.space.queue.index.runat:pairs({0},
         { iterator = box.index.GT }) do

         if t.runat > now then
            remaining = t.runat - now
            break
         else
            if t.status == STATUS.WAITING then
               log.info("Runat: W->R %s",t.id)
               if queue.wait:has_readers() then queue.wait:put(true,0) end
               box.space.queue:update({t.id},{
                  {'=', F.status, STATUS.READY },
                  {'=', F.runat, 0 },
               })
            else
               log.error("Runat: bad status %s for %s", t.status, t.id)
               box.space.queue:update({t.id},{{'=', F.runat, 0 }})
            end
         end
      end
      if not remaining or remaining > 1 then remaining = 1 end
      fiber.sleep(remaining)
   end
end)
```

Background actions: fibers

```
tarantool> queue.put({1,2,3},{delay=5})
---
- [1529452967197172, 'W', 1529452972.1973, 1, 2, 3]
...
```

Background actions: fibers

```
tarantool> queue.put({1,2,3},{delay=5})
---
- [1529452967197172, 'W', 1529452972.1973, 1, 2, 3]
...
```

```
tarantool> queue.take(0)
---
...
```

Background actions: fibers

```
tarantool> queue.put({1,2,3},{delay=5})
---
- [1529452967197172, 'W', 1529452972.1973, 1, 2, 3]
...
```

```
tarantool> queue.take(0)
---
...
```

```
tarantool> queue.take(5)
main/105/queue.runat I> Runat: W->R 1529452967197172ULL
main/101/background.lua I> Register 1529452967197172ULL by 1
---
- [1529452967197172, 'T', 0, 1, 2, 3]
...
```

Statistics & monitoring

queue.stats

```
function queue.stats()
    return {
        total   = box.space.queue:len(),
        ready   = box.space.queue.index.status:count({STATUS.READY}),
        waiting = box.space.queue.index.status:count({STATUS.WAITING}),
        taken   = box.space.queue.index.status:count({STATUS.TAKEN}),
    }
end
```

queue.stats

```
function queue.stats()
    return {
        total   = box.space.queue:len(),
        ready   = box.space.queue.index.status:count({STATUS.READY}),
        waiting = box.space.queue.index.status:count({STATUS.WAITING}),
        taken   = box.space.queue.index.status:count({STATUS.TAKEN}),
    }
end
```

```
tarantool> queue.stats()
---
- ready: 10
  taken: 2
  waiting: 5
  total: 17
...
```

Small queue

```
tarantool> local s = clock.time() queue.stats() return clock.time()-s
---
- 0.00019598007202148
...
```

```
tarantool> queue.stats()
---
- ready: 10
  taken: 2
  waiting: 5
  total: 17
...
```

Big queue

```
tarantool> for k = 1,1e6 do queue.put({1}) end
```

Big queue

```
tarantool> for k = 1,1e6 do queue.put({1}) end
```

```
tarantool> queue.stats()
---
- ready: 1000000
  taken: 0
  waiting: 0
  total: 1000000
...
```

Big queue

```
tarantool> for k = 1,1e6 do queue.put({1}) end
```

```
tarantool> queue.stats()
---
- ready: 1000000
  taken: 0
  waiting: 0
  total: 1000000
...
```

```
tarantool> local s = clock.time() queue.stats() return clock.time()-s
---
- 0.30088090896606
...
```

Cached counters

```
queue._stats = {
   ready   = 0LL,
   taken   = 0LL,
   waiting = 0LL,
}

for _,t in box.space.queue:pairs() do
   queue._stats[ t[F.status] ]
      = (queue._stats[ t[F.status] ] or 0LL)+1
end

function queue.stats()
   return {
      total   = box.space.queue:len(),
      ready   = queue._stats[STATUS.READY] or 0,
      waiting = queue._stats[STATUS.WAITING] or 0,
      taken   = queue._stats[STATUS.TAKEN] or 0,
   }
end
```

Cached counters

```
tarantool> queue.stats()
---
- ready: 1000000
  taken: 0
  waiting: 0
  total: 1000000
...

tarantool> local s = clock.time() queue.stats() return clock.time()-s
---
- 0.000020027160
...
```

Cached counters

```
-- function queue.take(timeout)
   queue._stats[ STATUS.READY ] = queue._stats[ STATUS.READY ] - 1
   queue._stats[ STATUS.TAKEN ] = queue._stats[ STATUS.TAKEN ] + 1
```

```
-- function queue.ack(id)
   queue._stats[ t[F.status] ] = queue._stats[ t[F.status] ] - 1
```

```
-- function queue.release(id)
   queue._stats[ t[F.status] ] = queue._stats[ t[F.status] ] - 1
   queue._stats[ STATUS.READY ] = queue._stats[ STATUS.READY ] + 1
```

...

Cached counters

```
tarantool> queue.stats()
---
- ready: 999998
  taken: 2
  waiting: 1
  total: 1000001
...
```

Cached counters

```
tarantool> queue.stats()
---
- ready: 999998
  taken: 2
  waiting: 1
  total: 1000001
...
```

...

Cached counters

```
tarantool> queue.stats()
---
- ready: 999998
  taken: 2
  waiting: 1
  total: 1000001
...
```

...

```
tarantool> queue.stats()
---
- ready: 4
  taken: -2
  waiting: 7
  total: 3
...
```

Data modification trigger

```
box.space.queue:on_replace(function(old,new)
   if old then
      queue._stats[ old[ F.status ] ] =
         queue._stats[ old[ F.status ] ] - 1
   end
   if new then
      queue._stats[ new[ F.status ] ] =
         queue._stats[ new[ F.status ] ] + 1
   end
end)
```

Data modification trigger

```
box.space.queue:on_replace(function(old,new)
   if old then
      queue._stats[ old[ F.status ] ] =
         (queue._stats[ old[ F.status ] ] or 0LL) - 1
   end
   if new then
      queue._stats[ new[ F.status ] ] =
         (queue._stats[ new[ F.status ] ] or 0LL) + 1
   end
end)
```

Demo

Network monitoring

Sockets

Sockets inside database!

Sockets inside database + appserver!

```
local socket = require 'socket'
```

Sockets inside database + appserver!

```
local socket = require 'socket'
```

```
local graphite_host = 'localhost'
local graphite_port = 2003
```

Sockets inside database + appserver!

```
local socket = require 'socket'
```

```
local graphite_host = 'localhost'
local graphite_port = 2003
```

```
local ai = socket.getaddrinfo(graphite_host, graphite_port, 1,
      { type = 'SOCK_DGRAM' })
local addr,port
for _,info in pairs(ai) do
   addr,port = info.host,info.port
   break
end
if not addr then error("Failed to resolve host") end
```

Sockets inside database + appserver!

```
local socket = require 'socket'
```

```
local graphite_host = 'localhost'
local graphite_port = 2003
```

```
local ai = socket.getaddrinfo(graphite_host, graphite_port, 1,
      { type = 'SOCK_DGRAM' })
local addr,port
for _,info in pairs(ai) do
   addr,port = info.host,info.port
   break
end
if not addr then error("Failed to resolve host") end
```

```
local remote = socket('AF_INET', 'SOCK_DGRAM', 'udp')
remote:sendto(addr, port, msg)
```

Monitoring fiber with socket

```
queue.monitor = fiber.create(function()
   fiber.name('queue.monitor')
   local remote = socket('AF_INET', 'SOCK_DGRAM', 'udp')

   while true do
      for k,v in pairs(queue.stats()) do
         local msg = string.format("queue.stats.%s %s %s\n",k,
            tonumber(v),math.floor(fiber.time()))
         remote:sendto(addr, port, msg)
      end
      fiber.sleep(1)
   end
end)
```

Also TCP

```
while true do
   local remote =  socket.tcp_connect(graphite_host, graphite_port)
   if not remote then
      log.error("Failed to connect %s",errno.strerror()) fiber.sleep(1)
   else
      while true do
         local data = {}
         for k,v in pairs(queue.stats()) do
            table.insert(data,string.format("queue.stats.%s %s %s\n",k,
               tonumber(v),math.floor(fiber.time())))
         end
         data = table.concat(data,'')
         if not remote:send(data) then
            log.error("%s",errno.strerror()) break
         end
         fiber.sleep(1)
      end
   end
end
```

Get your data in *RAM*

Reload code *without restart*

Code load

Code load

```
require 'my.module'
```

Code load

```
require 'my.module'
```

```
if not package.loaded['my.module'] then
   package.loaded['my.module'] = dofile('my/module.lua')
end
```

Code load

```
require 'my.module'
```

```
if not package.loaded['my.module'] then
   package.loaded['my.module'] = dofile('my/module.lua')
end
```

Reload

1. Remember `package.loaded`
2. Check first run
3. Clean loaded (`package.loaded[X] = nil`)
4. Proceed

Reload

```
local fiber = require'fiber'
local clock = require'clock'
local log   = require'log'

local not_first_run = rawget(_G,'_NOT_FIRST_RUN')
_NOT_FIRST_RUN = true
if not_first_run then
   for k,v in pairs(package.loaded) do
      if not preloaded[k] then
         package.loaded[k] = nil
      end
   end
else
   preloaded = {}
   for k,v in pairs(package.loaded) do
      preloaded[k] = true
   end
end
```

Reload

```
local fiber = require'fiber'
local clock = require'clock'
local log   = require'log'

require 'package.reload'
```

Reload

```
local fiber = require'fiber'
local clock = require'clock'
local log   = require'log'

require 'package.reload'
```

reload by

```
tarantool> dofile("/path/to/init.lua")
```

Reload

```
local fiber = require'fiber'
local clock = require'clock'
local log   = require'log'

require 'package.reload'
```

reload by

```
tarantool> dofile("/path/to/init.lua")
```

or just

```
tarantool> package.reload()
```

Reloadable code

```
-queue = {}
+if not queue then
+   queue = {}
+   ...
+   -- Autorelease only at start
+   ...
+   -- Initial stats
+   ...
+else
+   log.info("Queue reload %s", package.reload.count)
+end
```

Reloadable trigger

```
queue.on_replace_trigger = box.space.queue:on_replace(function(old,new)
  if old then
     queue._stats[ old[ F.status ] ] = queue._stats[ old[ F.status ] ] - 1
  end
  if new then
     queue._stats[ new[ F.status ] ] = queue._stats[ new[ F.status ] ] + 1
  end
end, queue.on_replace_trigger)
```

Reloadable trigger

```
queue.on_replace_trigger = box.space.queue:on_replace(function(old,new)
   if old then
      queue._stats[ old[ F.status ] ] = queue._stats[ old[ F.status ] ] - 1
   end
   if new then
      queue._stats[ new[ F.status ] ] = queue._stats[ new[ F.status ] ] + 1
   end
end, queue.on_replace_trigger)
```

Reloadable trigger

```
queue.on_replace_trigger = box.space.queue:on_replace(function(old,new)
   if old then
      queue._stats[ old[ F.status ] ] = queue._stats[ old[ F.status ] ] - 1
   end
   if new then
      queue._stats[ new[ F.status ] ] = queue._stats[ new[ F.status ] ] + 1
   end
end, queue.on_replace_trigger)
```

```
queue.on_disconnect_trigger = box.session.on_disconnect(function()
   ...
end,queue.on_disconnect_trigger)
```

Reloadable fiber

```
queue.runat = fiber.create(function()
    fiber.name('queue.runat')
    while true do
        local now = clock.realtime()
        local remaining
        -- ...
        if not remaining or remaining > 1 then remaining = 1 end
        fiber.sleep(remaining)
    end
end)
```

Reloadable fiber

```
queue.runchan = fiber.channel()
queue.runat = fiber.create(function()
   fiber.name('queue.runat')
   while true do
      local now = clock.realtime()
      local remaining
      -- ...
      if not remaining or remaining > 1 then remaining = 1 end
      queue.runchan:get(remaining)
   end
end)
```

Reloadable fiber

```
queue.runchan = queue.runchan or fiber.channel()
queue.runat = fiber.create(function()
    local gen = package.reload.count
    fiber.name('queue.runat.'..gen)
    while true do
        local now = clock.realtime()
        local remaining
        -- ...
        if not remaining or remaining > 1 then remaining = 1 end
        queue.runchan:get(remaining)
    end
    log.info("I'm done")
end)
```

Reloadable fiber

```
queue.runchan = queue.runchan or fiber.channel()
queue.runat = fiber.create(function()
   local gen = package.reload.count
   fiber.name('queue.runat.'..gen)
   while gen == package.reload.count do
      local now = clock.realtime()
      local remaining
      -- ...
      if not remaining or remaining > 1 then remaining = 1 end
      queue.runchan:get(remaining)
   end
   log.info("I'm done")
end)
```

Reloadable fiber

```
queue.runchan = queue.runchan or fiber.channel()
queue.runat = fiber.create(function()
   local gen = package.reload.count
   fiber.name('queue.runat.'..gen)
   while gen == package.reload.count do
      local now = clock.realtime()
      local remaining
      -- ...
      if not remaining or remaining > 1 then remaining = 1 end
      queue.runchan:get(remaining)
   end
   log.info("I'm done")
end)
```

```
while queue.runchan:has_readers() do
   queue.runchan:put(0,0)
end
```

Do not start console

```
if not fiber.self().storage.console then
   require'console'.start()
   os.exit()
end
```

Do not start console

```
if not fiber.self().storage.console then
   require'console'.start()
   os.exit()
end
```

```
if package.reload.count == 1 then
   require'console'.start()
   os.exit()
end
```

Summary

- Spaces & indices
- Console & strict
- Clock, id generation
- Channels
- Networking (listen + net.box)
- Connection triggers + peer hack
- Lua table keys: number vs cdata LL
- MsgPack
- Background fibers & by-index processing
- Accounting & monitoring
- Data modification trigger
- Code reload

```
require'strict'.on()
fiber = require'fiber'
clock = require'clock'
log   = require'log'

require 'package.reload'

local msgpack = require 'msgpack'
local socket = require 'socket'


box.cfg{
   listen = 'localhost:3311'
}

box.once('access:v1', function()
    box.schema.user.grant('guest', 'super')
end)

box.schema.create_space('queue',{
   format = {
      { name = 'id';     type = 'number' },
      { name = 'status'; type = 'string' },
      { name = 'runat';  type = 'number' },
      { name = 'data';   type = '*'      },
   };
   if_not_exists = true;
})

local F = {
   id     = 1;
   status = 2;
   runat  = 3;
   data   = 4;
}

local STATUS = {}
STATUS.READY = 'R'
STATUS.TAKEN = 'T'
STATUS.WAITING = 'W'

box.space.queue:create_index('primary', {
   parts = {1,'number'};
   if_not_exists = true;
})

box.space.queue:create_index('status', {
   parts = {2, 'string', 1, 'number'};
   if_not_exists = true;
})

box.space.queue:create_index('runat', {
   parts = {3, 'number', 1, 'number'};
   if_not_exists = true;
})

if not rawget(_G,'queue') then
   log.info("First start")
   queue = {}
   -- Autorelease only at start
   local c = 0
   for _,t in box.space.queue.index.status:pairs({STATUS.TAKEN}) do
      box.space.queue:update({t.id},{{'=', F.status, STATUS.READY }})
      c = c + 1
   end
   log.info("Autorelease %d tasks at start",c)

   -- Initial stats
   queue._stats = {
      R = 0LL,
      T = 0LL,
      W = 0LL,
   }

   log.info("Precalculate stats")
   for _,t in box.space.queue:pairs() do
      queue._stats[ t[F.status] ]
         = (queue._stats[ t[F.status] ] or 0LL)+1
   end


   queue.taken = {};
   queue.bysid = {};

else
   log.info("Queue reload %s", package.reload.count)

end

queue.on_replace_trigger = box.space.queue:on_replace(function(old,new)
   if old then
      queue._stats[ old[ F.status ] ] = queue._stats[ old[ F.status ] ] - 1
   end
   if new then
      queue._stats[ new[ F.status ] ] = queue._stats[ new[ F.status ] ] + 1
   end
end, queue.on_replace_trigger)

local function keypack( key )
    return msgpack.encode(key)
end
local function keyunpack( data )
   return msgpack.decode(data)
end
```

```
box.session.on_connect(function()
   log.info("connected %s:%s from %s", box.session.id(), box.session.user(), box.session.peer())
   box.session.storage.peer = box.session.peer()
end)


queue.on_disconnect_trigger = box.session.on_disconnect(function()
   log.info("disconnected %s:%s from %s", box.session.id(), box.session.user(), box.session.storage.peer)
   local sid = box.session.id()
   local bysid = queue.bysid[ sid ]
   if bysid then
      for key in pairs(bysid) do
         log.info("Autorelease %s by disconnect",keyunpack(key));
         queue.release( keyunpack(key) )
      end
      queue.bysid[ sid ] = nil
   end
end,queue.on_disconnect_trigger)

local clock = require 'clock'
local function gen_id()
   local new_id = clock.realtime64()/1e3
   while box.space.queue:get(new_id) do
      new_id = new_id + 1
   end
   return new_id
end

queue.wait = queue.wait or fiber.channel(0)

function queue.put(data,opts)
   local id = gen_id()
   local runat = 0
   local status = STATUS.READY
   if opts and opts.delay then
      -- if queue.runchan:has_readers() then queue.runchan:put(true,0) end
      runat = clock.realtime() + tonumber(opts.delay)
      status = STATUS.WAITING
   else
      if queue.wait:has_readers() then queue.wait:put(true,0) end
   end
   return box.space.queue:insert{ id, status, runat, unpack(data) }
end

queue.runchan = queue.runchan or fiber.channel()
queue.runat = fiber.create(function()
   local gen = package.reload.count
   fiber.name('queue.runat '..gen)
   while gen == package.reload.count do
      local now = clock.realtime()
      local remaining
      for _,t in box.space.queue.index.runat:pairs({0},{iterator = box.index.GT}) do
         if t.runat > now then
            remaining = t.runat - now
            break
         else
            if t.status == STATUS.WAITING then
               log.info("Runat: W->R %s",t.id)
               if queue.wait:has_readers() then queue.wait:put(true,0) end
               box.space.queue:update({t.id},{
                  {'=', F.status, STATUS.READY },
                  {'=', F.runat, 0 },
               })
            else
               log.error("Runat: bad status %s for %s",t.status, tostring(t))
               box.space.queue:update({t.id},{{'=', F.runat, 0 }})
            end
         end
      end
      if not remaining or remaining > 1 then remaining = 1 end
      queue.runchan:get(remaining)
   end
   log.info("I'm done")
end)
while queue.runchan:has_readers() do
   queue.runchan:put(0,0)
end

function queue.take(timeout)
   if not timeout then timeout = 0 end
   local now = fiber.time()
   local found
   while not found do
      for _,t in box.space.queue.index.status:pairs({STATUS.READY},{ iterator = box.index.EQ }) do
         found = t
         break
      end
      if not found then
         local left = (now + timeout) - fiber.time()
         if left <= 0 then return end
         queue.wait:get(left)
      end
   end

   local sid = box.session.id()
   local packid = keypack(found.id)
   log.info("Register %s by %s",found.id, sid)
   queue.bysid[ sid ] = queue.bysid[ sid ] or {}
   queue.taken[ packid ] = sid
   queue.bysid[ sid ][ packid ] = true

   return box.space.queue:update({found.id},{{'=', F.status, STATUS.TAKEN }})
end
```

```
function queue.get_task(id)
   if not id then error("Task id required",2) end
   local key = tonumber64(id)
   if not key then error(string.format("Task id '%s' is wrong",id),2) end
    local t = box.space.queue:get{key}
    if not t then
        error(string.format( "Task {%s} was not found", key ),2)
    end
   local packid = keypack(t.id)
    if not queue.taken[packid] then
        error(string.format( "Task %s not taken by any", key ),2)
    end
    if queue.taken[packid] ~= box.session.id() then
        error(string.format( "Task %s taken by %d. Not you (%d)",
            key, queue.taken[packid], box.session.id() ),2)
    end
    return t, packid
end

function queue.ack(id)
   local t,packid = queue.get_task(id)
   queue.taken[ packid ] = nil
   queue.bysid[ box.session.id() ][ packid ] = nil
   return box.space.queue:delete{t.id}
end

function queue.release(id)
   local t,packid = queue.get_task(id)
   queue.taken[ packid ] = nil
   queue.bysid[ box.session.id() ][ packid ] = nil
   if queue.wait:has_readers() then queue.wait:put(true,0) end
   return box.space.queue:update({t.id},{{'=', F.status, STATUS.READY }})
end

function queue.stats()
   return {
      total   = box.space.queue:len(),
      ready   = queue._stats[STATUS.READY] or 0,
      waiting = queue._stats[STATUS.WAITING] or 0,
      taken   = queue._stats[STATUS.TAKEN] or 0,
   }
end

local graphite_host = 'localhost'
local graphite_port = 2003
queue.monitor = fiber.create(function()
   local gen = package.reload.count
   fiber.name('queue.mon.'..gen)
   while gen == package.reload.count do
      local remote = require 'socket'.tcp_connect(graphite_host, graphite_port)
      if not remote then
         log.error("Failed to connect %s",errno.strerror())
         fiber.sleep(1)
      else
         while gen == package.reload.count do
            local data = {}
            for k,v in pairs(queue.stats()) do
               table.insert(data,string.format("queue.stats.%s %s %s\n",k,tonumber(v),math.floor(fiber.time())))
            end
            data = table.concat(data,'')
            if not remote:send(data) then
               log.error("%s",errno.strerror())
               break
            end
            fiber.sleep(1)
         end
      end
   end
   log.info("I'm done")
end)

if package.reload.count == 1 then
   require'console'.start()
   os.exit()
end
```

```
require'strict'.on()
fiber = require'fiber'
clock = require'clock'
log   = require'log'

require 'package.reload'

local msgpack = require 'msgpack'
local socket = require 'socket'


box.cfg{
   listen = 'localhost:3311'
}

box.once('access:v1', function()
    box.schema.user.grant('guest', 'super')
end)

box.schema.create_space('queue',{
   format = {
      { name = 'id';     type = 'number' },
      { name = 'status'; type = 'string' },
      { name = 'runat';  type = 'number' },
      { name = 'data';   type = '*'      },
   };
   if_not_exists = true;
})

local F = {
   id     = 1;
   status = 2;
   runat  = 3;
   data   = 4;
}

local STATUS = {}
STATUS.READY = 'R'
STATUS.TAKEN = 'T'
STATUS.WAITING = 'W'

box.space.queue:create_index('primary', {
   parts = {1,'number'};
   if_not_exists = true;
})

box.space.queue:create_index('status', {
   parts = {2, 'string', 1, 'number'};
   if_not_exists = true;
})

box.space.queue:create_index('runat', {
   parts = {3, 'number', 1, 'number'};
   if_not_exists = true;
})

if not rawget(_G,'queue') then
   log.info("First start")
   queue = {}
   -- Autorelease only at start
   local c = 0
   for _,t in box.space.queue.index.status:pairs({STATUS.TAKEN}) do
      box.space.queue:update({t.id},{{'=', F.status, STATUS.READY }})
      c = c + 1
   end
   log.info("Autorelease %d tasks at start",c)

   -- Initial stats
   queue._stats = {
      R = 0LL,
      T = 0LL,
      W = 0LL,
   }

   log.info("Precalculate stats")
   for _,t in box.space.queue:pairs() do
      queue._stats[ t[F.status] ]
         = (queue._stats[ t[F.status] ] or 0LL)+1
   end


   queue.taken = {};
   queue.bysid = {};

else
   log.info("Queue reload %s", package.reload.count)

end

queue.on_replace_trigger = box.space.queue:on_replace(function(old,new)
   if old then
      queue._stats[ old[ F.status ] ] = queue._stats[ old[ F.status ] ] - 1
   end
   if new then
      queue._stats[ new[ F.status ] ] = queue._stats[ new[ F.status ] ] + 1
   end
end, queue.on_replace_trigger)

local function keypack( key )
    return msgpack.encode(key)
end
local function keyunpack( data )
   return msgpack.decode(data)
end
```

```
box.session.on_connect(function()
   log.info("connected %s:%s from %s", box.session.id(), box.session.user(), box.session.peer())
   box.session.storage.peer = box.session.peer()
end)


queue.on_disconnect_trigger = box.session.on_disconnect(function()
   log.info("disconnected %s:%s from %s", box.session.id(), box.session.user(), box.session.storage.peer)
   local sid = box.session.id()
   local bysid = queue.bysid[ sid ]
   if bysid then
      for key in pairs(bysid) do
         log.info("Autorelease %s by disconnect",keyunpack(key));
         queue.release( keyunpack(key) )
      end
      queue.bysid[ sid ] = nil
   end
end,queue.on_disconnect_trigger)

local clock = require 'clock'
local function gen_id()
   local new_id = clock.realtime64()/1e3
   while box.space.queue:get(new_id) do
      new_id = new_id + 1
   end
   return new_id
end

queue.wait = queue.wait or fiber.channel(0)

function queue.put(data,opts)
   local id = gen_id()
   local runat = 0
   local status = STATUS.READY
   if opts and opts.delay then
      -- if queue.runchan:has_readers() then queue.runchan:put(true,0) end
      runat = clock.realtime() + tonumber(opts.delay)
      status = STATUS.WAITING
   else
      if queue.wait:has_readers() then queue.wait:put(true,0) end
   end
   return box.space.queue:insert{ id, status, runat, unpack(data) }
end

queue.runchan = queue.runchan or fiber.channel()
queue.runat = fiber.create(function()
   local gen = package.reload.count
   fiber.name('queue.runat:'..gen)
   while gen == package.reload.count do
      local now = clock.realtime()
      local remaining
      for _,t in box.space.queue.index.runat:pairs({0},{iterator = box.index.GT}) do
         if t.runat > now then
            remaining = t.runat - now
            break
         else
            if t.status == STATUS.WAITING then
               log.info("Runat: W->R %s",t.id)
               if queue.wait:has_readers() then queue.wait:put(true,0) end
               box.space.queue:update({t.id},{
                  {'=', F.status, STATUS.READY },
                  {'=', F.runat, 0 },
               })
            else
               log.error("Runat: bad status %s for %s",t.status, tostring(t))
               box.space.queue:update({t.id},{{'=', F.runat, 0 }})
            end
         end
      end
      if not remaining or remaining > 1 then remaining = 1 end
      queue.runchan:get(remaining)
   end
   log.info("I'm done")
end)
while queue.runchan:has_readers() do
   queue.runchan:put(0,0)
end

function queue.take(timeout)
   if not timeout then timeout = 0 end
   local now = fiber.time()
   local found
   while not found do
      for _,t in box.space.queue.index.status:pairs({STATUS.READY},{ iterator = box.index.EQ }) do
         found = t
         break
      end
      if not found then
         local left = (now + timeout) - fiber.time()
         if left <= 0 then return end
         queue.wait:get(left)
      end
   end

   local sid = box.session.id()
   local packid = keypack(found.id)
   log.info("Register %s by %s",found.id, sid)
   queue.bysid[ sid ] = queue.bysid[ sid ] or {}
   queue.taken[ packid ] = sid
   queue.bysid[ sid ][ packid ] = true

   return box.space.queue:update({found.id},{{'=', F.status, STATUS.TAKEN }})
end
```

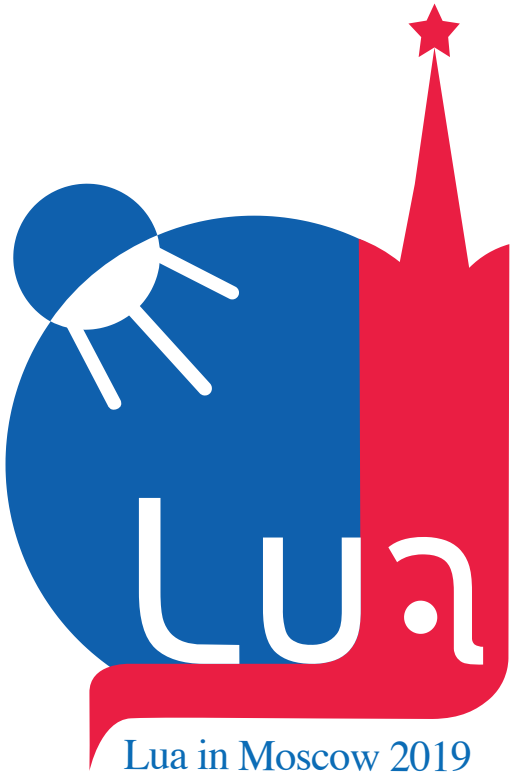
```
function queue.get_task(id)
   if not id then error("Task id required",2) end
   local key = tonumber64(id)
   if not key then error(string.format("Task id '%s' is wrong",id),2) end
    local t = box.space.queue:get{key}
    if not t then
        error(string.format( "Task {%s} was not found", key ),2)
    end
   local packid = keypack(t.id)
    if not queue.taken[packid] then
        error(string.format( "Task %s not taken by any", key ),2)
    end
    if queue.taken[packid] ~= box.session.id() then
        error(string.format( "Task %s taken by %d. Not you (%d)",
            key, queue.taken[packid], box.session.id() ),2)
    end
    return t, packid
end

function queue.ack(id)
   local t,packid = queue.get_task(id)
   queue.taken[ packid ] = nil
   queue.bysid[ box.session.id() ][ packid ] = nil
   return box.space.queue:delete{t.id}
end

function queue.release(id)
   local t,packid = queue.get_task(id)
   queue.taken[ packid ] = nil
   queue.bysid[ box.session.id() ][ packid ] = nil
   if queue.wait:has_readers() then queue.wait:put(true,0) end
   return box.space.queue:update({{t.id},{{'=', F.status, STATUS.READY }}})
end

function queue.stats()
   return {
      total   = box.space.queue:len(),
      ready   = queue._stats[STATUS.READY] or 0,
      waiting = queue._stats[STATUS.WAITING] or 0,
      taken   = queue._stats[STATUS.TAKEN] or 0,
   }
end

local graphite_host = 'localhost'
local graphite_port = 2003
queue.monitor = fiber.create(function()
   local gen = package.reload.count
   fiber.name('queue.mon:'..gen)
   while gen == package.reload.count do
      local remote =  require 'socket'.tcp_connect(graphite_host, graphite_port)
      if not remote then
         log.error("Failed to connect %s",errno.strerror())
         fiber.sleep(1)
      else
         while gen == package.reload.count do
            local data = {}
            for k,v in pairs(queue.stats()) do
               table.insert(data,string.format("queue.stats.%s %s %s\n",k,tonumber(v),math.floor(fiber.time())))
            end
            data = table.concat(data,'')
            if not remote:send(data) then
               log.error("%s",errno.strerror())
               break
            end
            fiber.sleep(1)
         end
      end
   end
   log.info("I'm done")
end)

if package.reload.count == 1 then
   require'console'.start()
   os.exit()
end
```

20K RPS queue

One day of coding

300 Lines of code

Bench?



Questions?

http://github.com/tarantool
http://github.com/moonlibs

Mons Anderson

Mail.ru Cloud Solutions

IPONWEB

TAMASHI