Painless Parallelization with Gearman

LCA 2011

Tim Uckun

http://tim.uckun.com/

Enspiral LTD New Zealand

Linux + Postgresql + Ruby + Rails + Gearman

What is Gearman?


MANAGER
GEARMAN

“A massively distributed, massively fault tolerant fork mechanism.”

- Joe Stump, SimpleGeo

A protocol with multiple implementations.

Job Server

- Perl Server (Gearman::Server in CPAN)
  - The original implementation
  - Actively maintained by folks at SixApart
- C Server (https://launchpad.net/gearmand)
  - Rewrite for performance and threading
  - Added new features like persistent queues
  - Different port (IANA assigned 4730)
  - Now moving to C++

Client API

- Available for most common languages
- Command line tool
- User defined functions in SQL databases
  - MySQL
  - PostgreSQL
  - Drizzle

Worker API

- Available for most common languages
  - Usually in the same packages as the client API
- Command line tool

Why Gearman?

Features

- Open Source
- Simple, Fast
- Multi-language
- Flexible application design
- No single point of failure

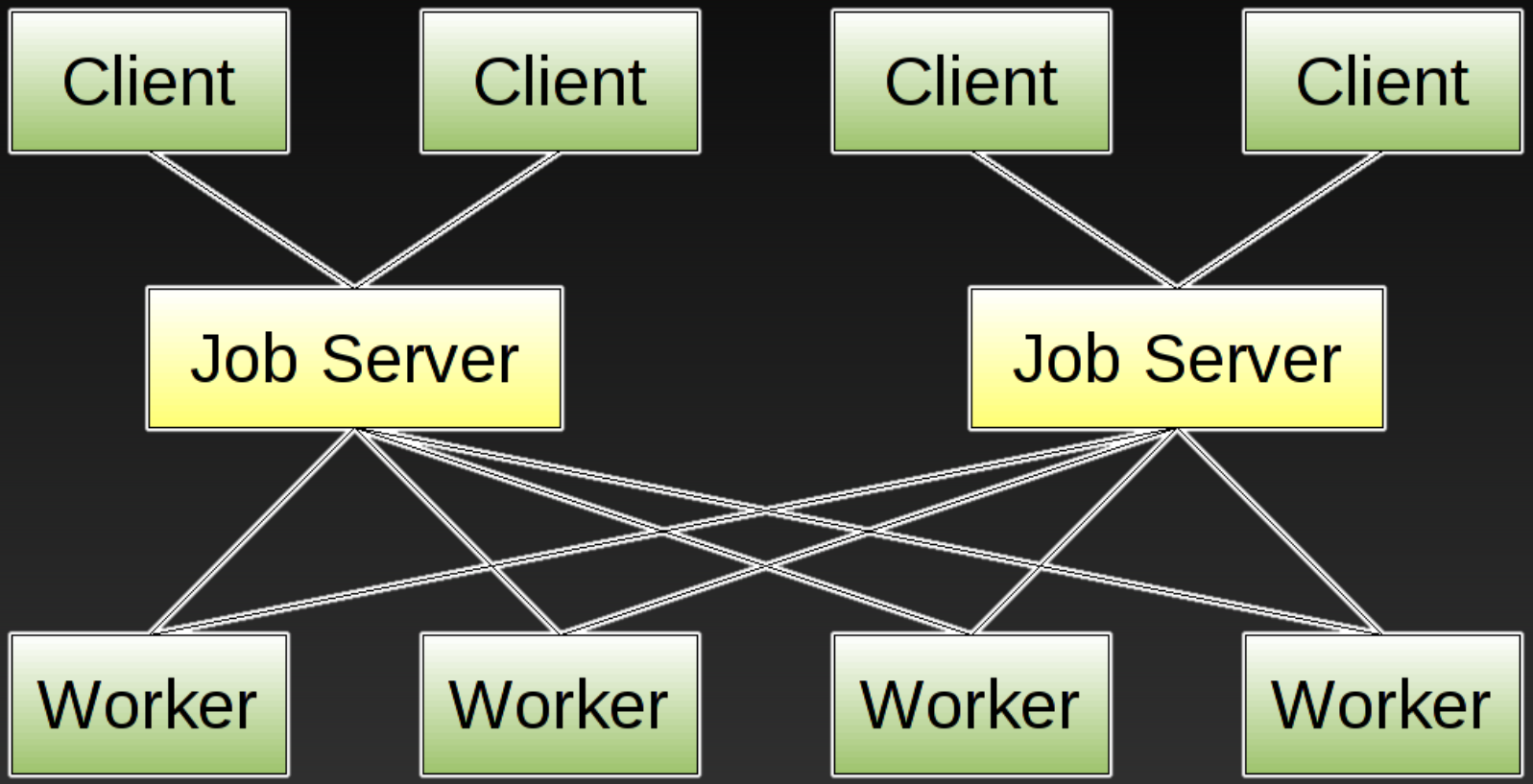
Persistent Queues (or Not)

- By default, jobs are only stored in memory
- Various persistence options
  - MySQL/Drizzle
  - PostgreSQL
  - SQLite
  - Tokyo Cabinet
  - memcached (not always “persistent”)

Foreground
(synchronous)
or
Background
(asynchronous)

While large-scale architectures work well, you can start off simple.

How does Gearman work?


Client
Client
Client
Client
Job Server
Job Server
Worker
Worker
Worker
Worker

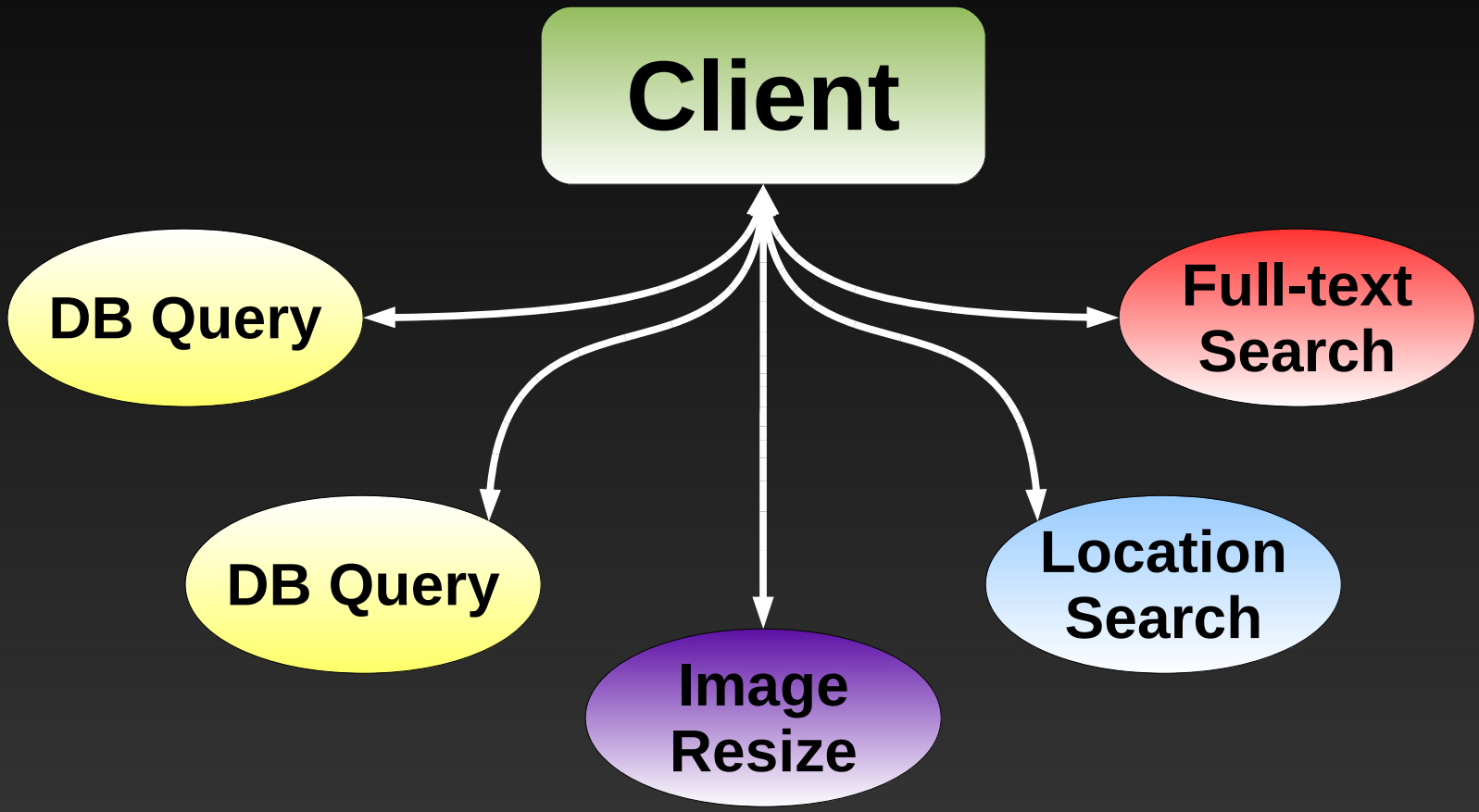
Use Cases

- Scatter/Gather
- Map/Reduce
- Asynchronous Queues
- Pipeline Processing

Scatter/Gather

- Perform a number of tasks concurrently
- Great way to speed up web applications
- Tasks don't need to be related
- Allocate dedicated resources for different tasks
- Push logic down to where data exists

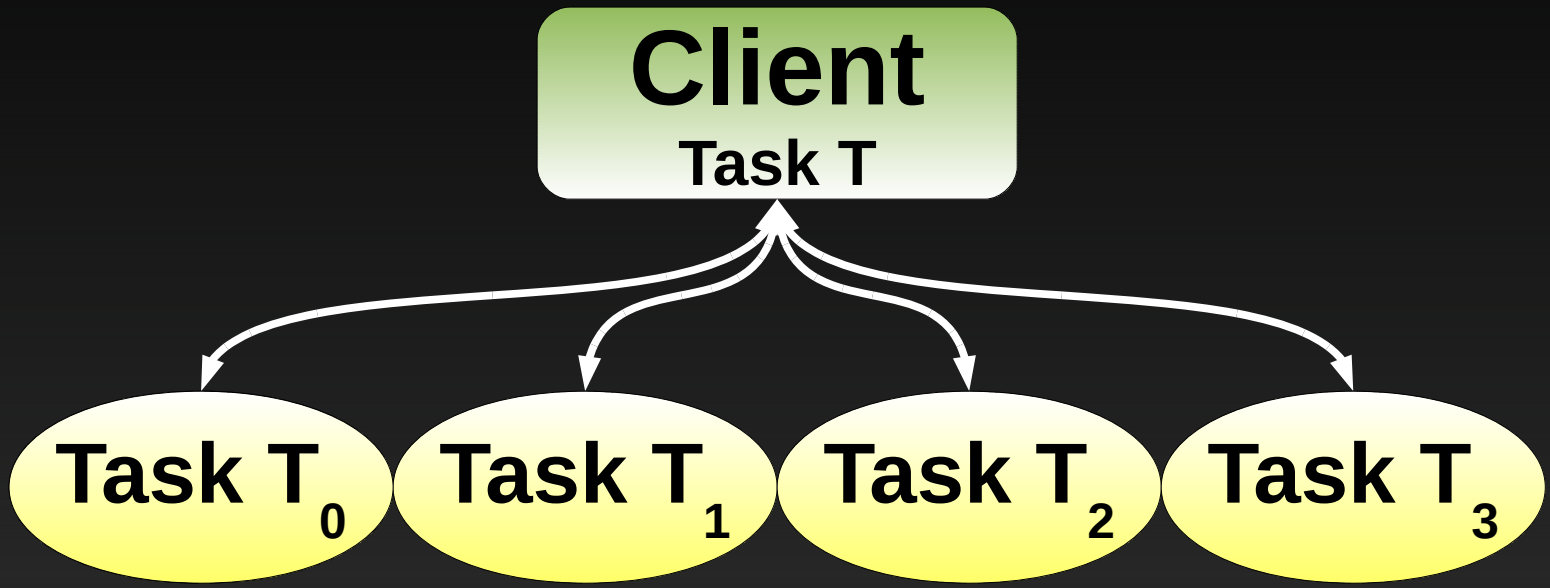
Scatter/Gather



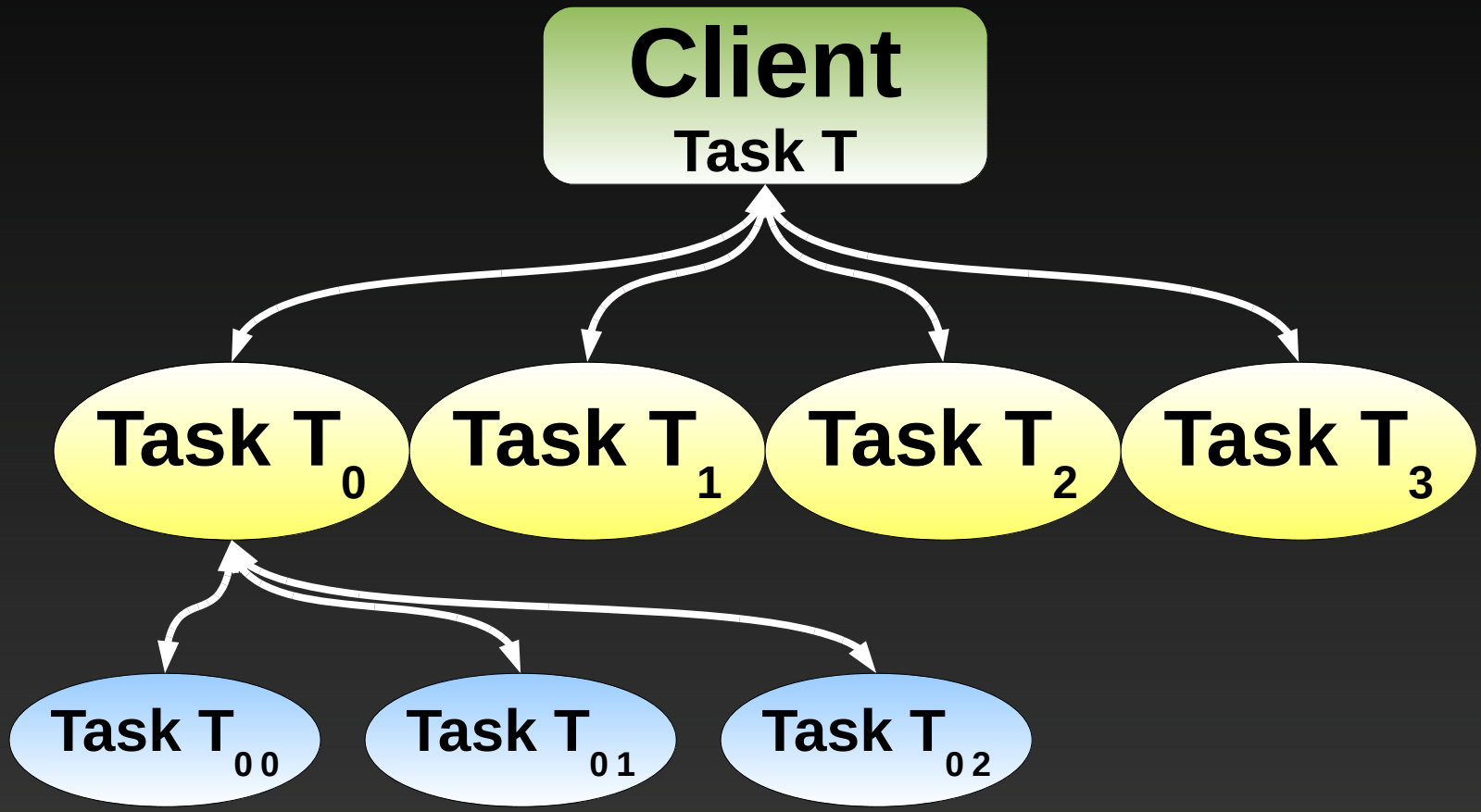
Map/Reduce

- Similar to scatter/gather, but split up one task
- Push logic to where data exists (map)
- Report aggregates or other summary (reduce)
- Can be multi-tier
- Can be asynchronous

Map/Reduce



Map/Reduce

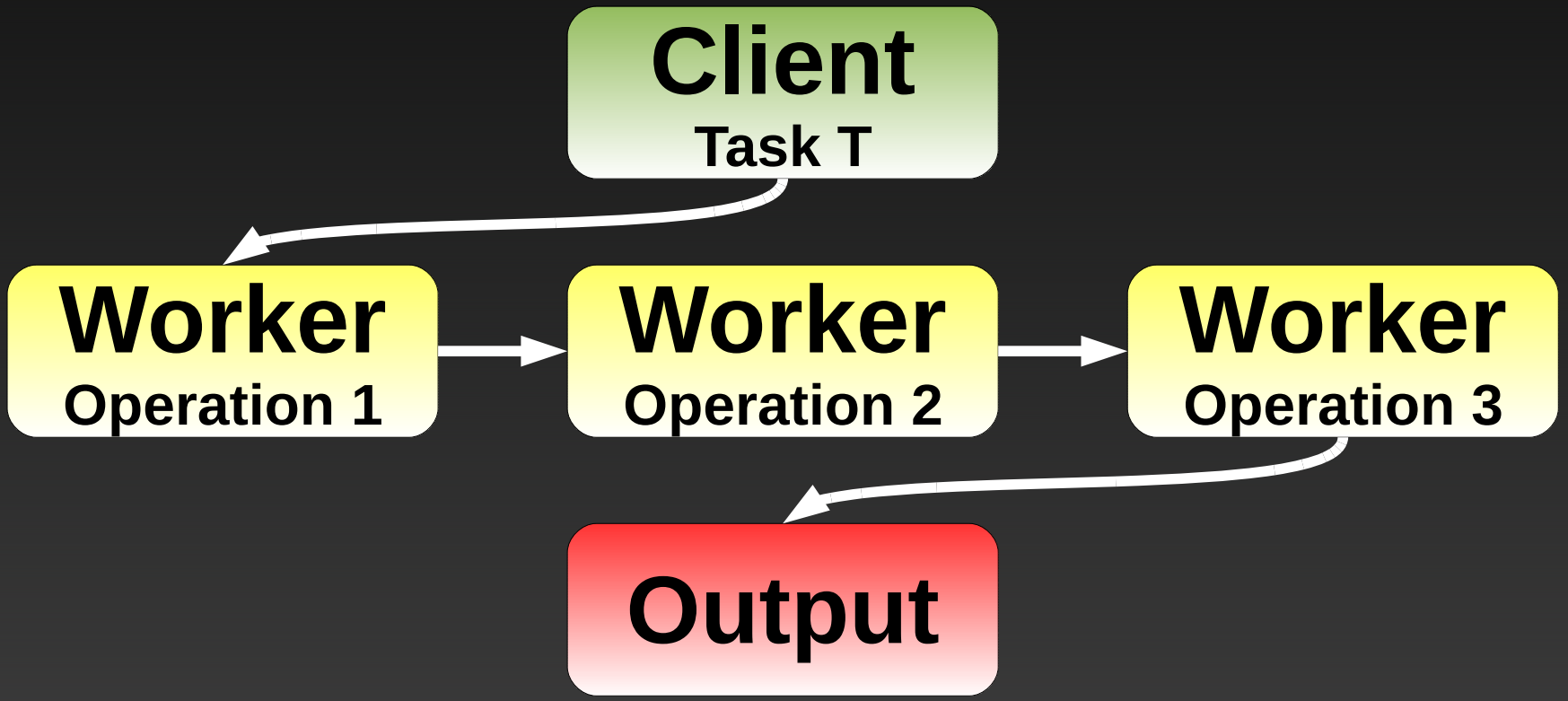

Client
Task T
Task T0
Task T1
Task T2
Task T3
Task T00
Task T01
Task T02

Asynchronous Queues

- They help you scale
- Not everything needs immediate processing
  - Sending e-mail, tweets, …
  - Log entries and other notifications
  - Data insertion and indexing
- Allows for batch operations

Pipeline Processing

- Some tasks need a series of transformations
- Chain workers to send data for the next step



Examples

Examples

Asynchronous Clients.

```
# Submit Job

# gearman_servers is an array
# data is a string (Marshall::dump, JSON, YAML, XML etc)

gearman = Gearman::Worker.new(gearman_servers)

gearman.do_task('queue_name', data, {:background => true})

#One Liner
Gearman::Client.new(gearman_servers).do_task(
      'recalculate_index', data, :background => true })
```

Examples

Workers.

```
worker = Gearman::Worker.new(CONFIG.gearman.servers)
worker.reconnect_sec = 2

worker.add_ability('recalculate_index') do |data,job|
  item = Marshal::load(data)
  # … Do the work here
End

loop{ worker.work }
```

Examples

More Complex Synchronous Client.

```
servers = ['localhost:4730', 'other.host.com:1234']

client = Gearman::Client.new(servers)
taskset = Gearman::TaskSet.new(client)

task = Gearman::Task.new('some_task', some_data)
task.retry_count = 2
task.on_complete {|d| puts d }
task.on_exception {|ex| puts "This should never be called" }
task.on_warning {|warning| puts "WARNING: #{warning}" }
task.on_retry { puts "PRE-RETRY HOOK: retry no. #{task.retries_done}" }
task.on_fail { puts "TASK FAILED, GIVING UP" }

taskset.add_task(task)
taskset.wait(100)
```

Examples

Returning data to synchronous clients

```
worker.add_ability('multiplication') do |data,job|
  values = Marshal.load(data)

  # Ruby returns the result of the last expression
  # The gearman client turns it into a string by calling to_s
  # You can also serialize the result on the worker
  # Marshall.dump, JSON, YAML, XML, etc..

  values.first * values.last
end
```

Examples

Worker Raises an exception.

```
#raised exceptions are passed back to clients automatically

servers = ['localhost:4730']
w = Gearman::Worker.new(servers)

w.add_ability('some_task') do |data,job|
  raise Exception.new("Exception in worker (args: #{data.inspect})")
end
loop { w.work }
```

Examples

Chunked Data Client

```
client = Gearman::Client.new(servers)
taskset = Gearman::TaskSet.new(client)

task = Gearman::Task.new('chunked_transfer')
task.on_data {|d| puts d }
task.on_complete {|d| puts d }

taskset.add_task(task)
taskset.wait(100)
```

Examples

Chunked Data Worker

```
worker = Gearman::Worker.new(servers)

worker.add_ability('chunked_transfer') do |data, job|
  5.times do |i|
    sleep 1
    job.send_data("CHUNK #{i}")
    job.report_status(i,5)
  end
  "EOD"
end
loop { worker.work }
```

Examples

Get the state of the queue

```
hash  = Gearman::Server.new(gearman_servers).status


{"server1.mycompany.com:4730"=>
  {"fast_track"=>{:queue=>"0", :running=>"0", :workers=>"2"},
   "consolidated_url"=>{:queue=>"1", :running=>"1", :workers=>"2"},
   "url_expire"=>{:queue=>"2545", :running=>"4", :workers=>"4"},
   "digg"=>{:queue=>"180", :running=>"2", :workers=>"2"},
   "trending"=>{:queue=>"0", :running=>"0", :workers=>"3"},
   "facebook"=>{:queue=>"181", :running=>"3", :workers=>"3"},
   "reddit"=>{:queue=>"20", :running=>"3", :workers=>"3"},
   "screenshot"=>{:queue=>"0", :running=>"0", :workers=>"2"},
   "search_result"=>{:queue=>"2023", :running=>"7", :workers=>"7"},
   "twitter"=>{:queue=>"126", :running=>"3", :workers=>"4"}}}
```

Database UDFs

- Also useful as a database trigger
- Start background jobs on database changes
- Postgres, Mysql, Drizzle

```
SELECT gman_servers_set('localhost');
SELECT gman_do('reverse', 'Hello World!');
SELECT gman_do_high('reverse', 'Hello World!');
SELECT gman_do_low('reverse', 'Hello World!');
SELECT gman_do_background('reverse', 'Hello World!');
SELECT gman_do_high_background('reverse', 'Hello World!');
SELECT gman_do_low_background('reverse', 'Hello World!');
```

Optional Ingredients

- Databases (SQL or otherwise)
- Shared or distributed file systems
- Other network protocols
  - HTTP
  - E-Mail
- Domain specific libraries
  - Image manipulation
  - Full-text indexing

Timeouts

- By default, operations block forever
- Clients may want a timeout on foreground jobs
- Workers may need to periodically run other code besides job callback

Source: 09

Shortcomings

- Clients must connect to all servers.
- No replication between servers.
- Slower than pure messaging servers.
- Logging not all that great.
- Steps must be taken to assure recovery of queued messages if a server is completely destroyed.
- Small community, development has slowed.

Get involved!

- http://gearman.org/
  - Mailing list, documentation, related projects
- #gearman on irc.freenode.net

Questions?