An Englishman’s Guide To Queueing
paulhammond.org/2012/queueing
“An Englishman, even if he is alone, forms an orderly queue of one”

– George Mikes, How to be an Alien, 1946
<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
<th>DevOps</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVOPS BORAT</td>
<td>-</td>
<td>230,561</td>
</tr>
<tr>
<td>Patrick Debois</td>
<td>Jedi BVBA</td>
<td>230</td>
</tr>
<tr>
<td>Artur Bergman</td>
<td>Fastly</td>
<td>229</td>
</tr>
<tr>
<td>Mike Krieger</td>
<td>Instagram</td>
<td>123</td>
</tr>
<tr>
<td>Mike Rembetsy</td>
<td>Etsy</td>
<td>122</td>
</tr>
<tr>
<td>Theo Schlossnagle</td>
<td>OmniTI</td>
<td>116</td>
</tr>
<tr>
<td>John Willis</td>
<td>enStratus</td>
<td>112</td>
</tr>
<tr>
<td>Paul Hammond</td>
<td>Typekit</td>
<td>0</td>
</tr>
<tr>
<td>John Allspaw</td>
<td>Etsy</td>
<td>-32</td>
</tr>
</tbody>
</table>
What should happen when this button gets pushed? It can’t be that hard, right?
# on update

UPDATE users SET devups = devups + 1;

# to display the leaderboard

SELECT * FROM users ORDER BY devups LIMIT 10;
“We need an audit trail”
# on update

```sql
INSERT INTO devups (user_id, voter_id, created)
VALUES (10, 100, "2012-10-04 14:07:23");
```

# to display the leaderboard

```sql
SELECT user_id, count(*) AS c FROM devups
GROUP BY user_id ORDER BY c DESC LIMIT 10;
```
“Great Job!”
“Can we get a page showing the people you’ve voted for?”
# to display the people you’ve voted for

```
SELECT user_id, count(*) AS c FROM devups WHERE user_id IN (SELECT user_id FROM devups AS v WHERE v.voter_id = 3) GROUP BY user_id ORDER BY c DESC LIMIT 10;
```
“Can we change the algorithm?”
# generate leaderboard with weighted averages

```sql
SELECT user_id, SUM((SELECT count(*) FROM devups AS v WHERE v.voter_id = u.voter_id AND v.user_id = u.user_id) / (SELECT count(*) FROM devups AS v WHERE v.voter_id = u.voter_id)) FROM devups AS u GROUP BY u.user_id;
```
“Can we show your score at the top of every page?”
“Can we get a leaderboard of people that work at Etsy?”
“Can we get a leaderboard of companies?”
“Can we get a leaderboard of people called Mike?”
“Can we get a leaderboard of people called Theo?”
<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
<th>DevOps</th>
</tr>
</thead>
<tbody>
<tr>
<td>THEO SCHLOSSNAGLE</td>
<td>OmniTI</td>
<td>116</td>
</tr>
</tbody>
</table>
“Can we get a leaderboard of people in London?”
“Can we get a leaderboard of people with linked Twitter accounts?”
“Can we get a leaderboard of people with no Twitter account?”
“Can we get a leaderboard of people you follow on Twitter?”
“Why is the site so slow?”
Do slow calculations when inserting the data

OR

Do slow calculations when displaying the data
Or you can use a Job Queue
Do slow work outside the HTTP Request Handler
Todo List:

• Send password reminder to user
  16
• Recalculate vote score for user
  16
• Resize image
  4356

Application Server

“Queue this task”

“Got it”

Job Queue Server
Todo List:
- Send password reminder to user
  16
- Recalculate vote score for user
  16
- Resize image
  4356
Gearman
Beanstalkd
Resque
Kafka
Kestrel
Celerey
Celerey
RabbitMQ
Amazon SQS
Build your own
...

I’m not going to talk about any of these today.
I’m going to talk about everything else you need to know.
#1 There is more than one way to use them.
Edit this tag

sometag

Or, use a different format: sometag

SAVE

Note: Because this tag is attached to 40 things, we'll put this particular job into a queue to make sure it completes. This just means that this particular update will take a few minutes.

Or, cancel this and return to your tag list.

Did you know?

sometag is currently attached to 40 things. Editing the tag here will update it everywhere.

You can use this form to add new tags to all the things that are already tagged with sometag, and/or change the something else completely.

(Separate each tag with a space, or if you want to join 2 words together into one tag, use double quotes: "early commute")
We've placed that particular editing job into a queue. It should be done in a few minutes.
The webfontloader network graph

All branches in the network using typekit/webfontloader as the reference point. Read our blog post about how it works.

Show Help

Loading graph data
The webfontloader network graph

All branches in the network using typekit/webfontloader as the reference point. Read our blog post about how it works.
Anything not shown on the next page load can be silently queued.
Emails, tweets, external apis, webhooks...
User: 50–200 ms
Follower: 1–10 seconds
Stranger: 1–2 minutes
Think about the user interface.
Errors happen.
class Job::HandleVote
  def perform
    ...
    db.select(...)  
    ...
    end
  end
end
What happens when the database connection fails?
# deep inside most web frameworks

begin
  handle_http_request
catch exception
  display_internal_error
end
If you’re outside the HTTP request handler then you can’t just show an error page.
# many queue systems
job = queue.pop
begin
  job.run
catch exception
  log(exception)
  # drop job on floor
end
# some systems
job = queue.pop
begin
    job.run
catch exception
    log(exception)
    job.requeue
end
Errors can occur anywhere
# some systems

```ruby
job = queue.pop
begin
  job.run
catch exception
  log(exception)
  job.requeue
end
```
# resilient systems
job = queue.reserve_job
begin
  job.run
  queue.remove(job)
catch exception
  log(exception)
end
It is impossible to guarantee a job will be run exactly once.
class SendPasswordReminder {
    def perform {
        ...
        email.send()
        ...
    }
}
Job will be run one or zero times

OR

Job will be run one or many times
Reliable job queues are really slow.
Slow reliable queue
AND
Fast unreliable queue
Job will be run one or zero times
OR
Job will be run one or many times
Job will be run one or zero times
OR
Job will be run one or many times
(or possibly zero if things go really wrong)
#3 Idempotency is awesome.
idempotent ≠ nilpotent
foo = foo + 1
Reindexing a document in the search index
Fetching data from a read-only API ✔
Resizing a photo (in some cases)
Sending an email
Posting a tweet
Writing to an external API
Idempotent means it’s OK if it runs twice
Update canonical source in request. Then queue an idempotent job to update denormalized copies.
If a job fails to run it’ll fix itself next time.
#4 Jobs don’t run in order.
No shared state between workers.
“Reindex user 20 from the database”

NOT

“Reindex user 20 with these attributes”
Jobs that create other jobs when finished.
#5  Lock contention hurts.
Sometimes jobs stomp on each other when run in parallel.
So you add locking.
class ReindexUser {
    def perform {
        raise LockedError unless lock(user)
        ...
    }
}
Very common for one user to generate multiple jobs in a short period of time.
Queue churn
Do you really need to lock?
Can your locks be more fine-grained?
#6 Alerting is hard.
Queue length
Jobs run per second
Jobs queued per second
Jobs failed per second
Lock fails per second
(Jobs queued - Jobs run)
Job run time
Job queue time
Job total time
Graph all of these metrics. They’re really addictive.
Graph all of these metrics. But don’t alert on most of them.
Job queue alerts aren’t usually actionable.
“Wait for it to go back down”
Thresholds are hard to choose.
Normal usage changes over time.
Job queues provide flexibility.
Time based metrics are most useful.
Find good thresholds through trial and error.
Base thresholds on your SLAs or user experience needs.
“Reindexing jobs cannot queue for more than half a second”
Autoscale capacity based on queue length?
#7 You might not want to use a queue.
If you need a job to run immediately then you don’t need a queue, you need a worker pool.
Connect to a worker over the network
Multi-threaded servers
Event loop based servers
Fork a new process
Needs monitoring and capacity planning
#8 You need to think.
1. There is more than one way to use them.
2. Errors happen.
3. Idempotency is awesome.
4. Jobs don’t run in order.
5. Lock contention hurts.
6. Alerting is hard.
7. You might not want to use a queue.
8. You need to think.
Thank you

Slides will soon be available at paulhammond.org/2012/queueing