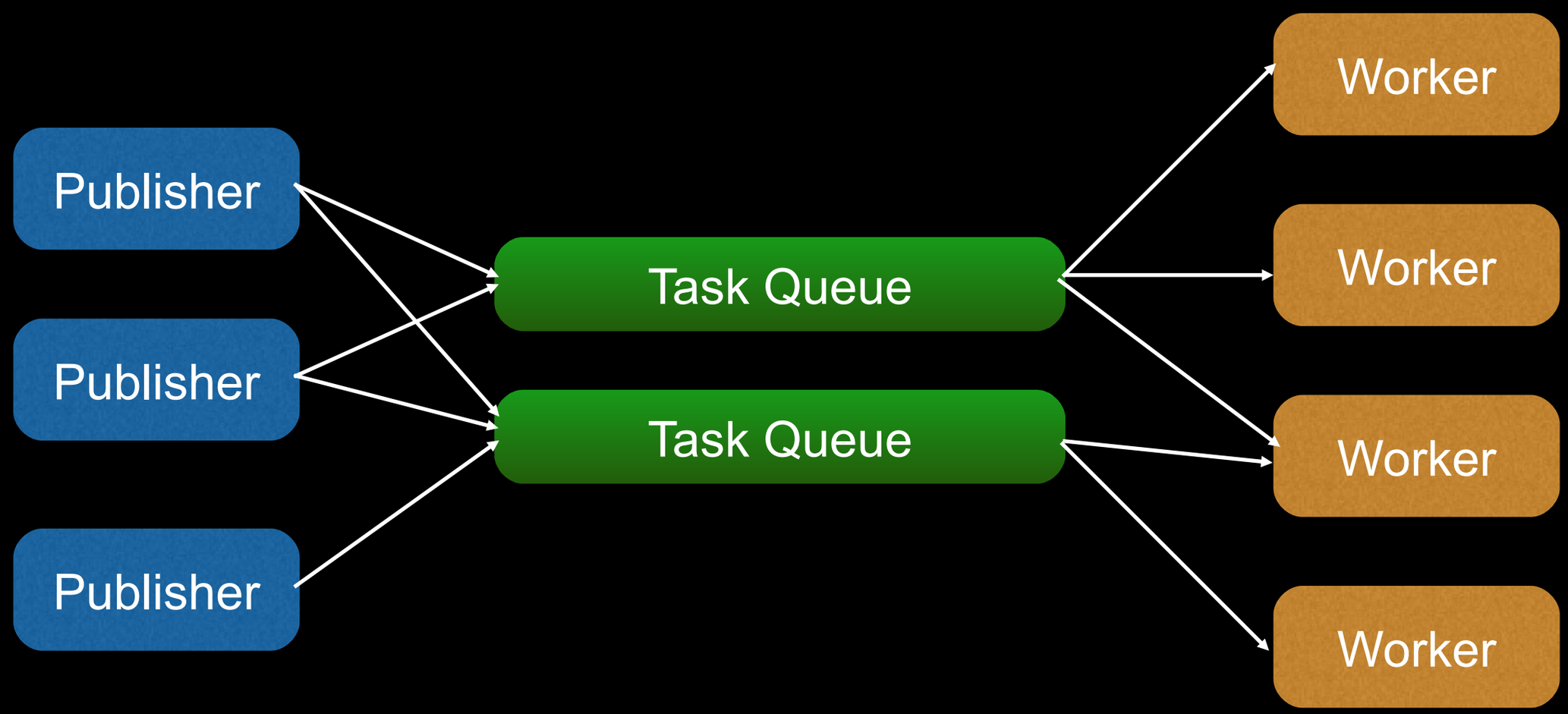
Celery 分布式任务队列

叶剑烨
Glow Inc
http://glowing.com

关于我

- 4 年 Python 开发经验，主要用于 web/mobile 的 server 端开发，数据分析及自动化环境布署的脚本
- Glow 的技术负责人，曾就职于 Google, Slide 和 Nvidia
- 去年升级成为爸爸，女儿现在一岁半了
- 博客： http://ryanye.me
- 微博：http://weibo.com/yejianye
- 微信： @ryanyewx

什么是任务队列？


Publisher
Publisher
Publisher
Task Queue
Task Queue
Worker
Worker
Worker
Worker

任务队列是如何工作的？

- Producer 发送一个任务到 Broker
- Broker 根据任务类型，将其发送到某个任务队列中
- 每个任务队列一般都有至少一个 Worker
- Worker 不断的从任务队列中取出任务来执行，并将执行完的任务从队列中删除

任务队列有什么用？

- 降低服务器对用户的响应时间
- 将重负载的任务隔离到单独的服务器，避免对其他进程的影响
- 在预定的时间执行某段代码

使用场景

- 发送电子邮件或是 Push notification
- 对用户上传的文件进行病毒扫描
- 用户图片的压缩与优化
- Feed 的传播
- 租赁模式下的服务到期提醒

可能解决的方案 ...

- Cronjob
  - 优点：简单可靠
  - 缺点：明显的更新延迟，难以将 workload 分配到多台机器上
- 自己编写 Walker 进程
  - 优点：定制化能力强，没有外部依赖
  - 缺点：开发成本高

Celery 概览

- Website: http://celeryproject.org
- github: https://github.com/celery/celery
  - 2261 Star, 703 Fork
- 很详尽的使用文档，并包含大量的例子
- 共有 7631 个 commit ，首个 commit 在 2009 年 4 月
- 50,000 行代码，其中 23,000 行为测试代码

Celery 中重要概念

- Task
- Broker
- Worker

Celery 支持的 Broker

- 成熟
  1. RabbitMQ (最佳)
  2. Redis (功能完备)
- 试验性
  3. SQLAlchemy
  4. MongoDB
  5. Amazon SQS
  6. CouchDB
  7. Beantalk
  8. IronMQ

Worker 的并发模式

- Process
- Thread
- Gevent
- Eventlet
- 推荐对于 CPU 依赖的任务使用 Process 并发模式，对于 IO 依依赖的任务使用 Gevent 并发模式。
  在一个系统中两种并发可以共存

Hello World!

定义任务，tasks.py

```
from celery import Celery

app = Celery('tasks', broker='redis://localhost:6379/0')

@app.task
def add(x, y):
    return x + y
```

启动worker

```
celery -A tasks worker --loglevel=info
```

调用任务

```
from tasks import add
result = add.delay(4, 4)
print result.get(timeout=1)
```

设定任务开始的时间

- 在指定时间执行任务

```
from datetime import datetime
from tasks import send_email
send_email.apply_async(
    ('ryan@glowing.com', 'PyCon 2013 China has started!'),
    eta=datetime(2013, 12, 8, 9, 30)
)
```

- 在指定延时后执行任务

```
from tasks import send_notification
send_notification.apply_async(
    ('ryanye', 'The barrack is completed'),
    countdown=60
)
```

多任务队列

- 将任务分配至不同的队列

```
app = Celery('tasks', broker='redis://localhost:6379/0')
app.conf.update(
    CELERY_ROUTES = {
        'tasks.send_email'        : 'short-time-job',
        'tasks.send_notification' : 'short-time-job',
        'tasks.generate_pdf'      : 'long-time-job',
    }
)
```

- 让 worker 只监听指定的队列

```
celery -A tasks worker -Q long-time-job
```

HTTP 调用

- Webhook 使 Celery 可以通用 HTTP 调用将外部 Service 的任务调用放入队列

```
from celery.task.http import URL
map_api = URL('http://maps.googleapis.com/maps/api/geocode/json')
res = map_api.get_async(
        address=u'上海市西藏中路336号',
        sensor='true'
    )
print res.get(timeout=10)
```

任务回调

```
@app.task
def email_sent_success(result):
    return logger.info("email sent success: %s" % result)

@app.task
def email_sent_failed(task_id):
    result = AsyncResult(task_id)
    return logger.error("email sent failed: %s" % result.traceback)
```

```
from tasks import send_email
from tasks import email_sent_success
from tasks import email_sent_failed

send_email.apply_async(
    ('ryan@glowing.com', subject, content),
    link=email_sent_success.s(),
    link_error=email_sent_failed.s()
)
```

Celery 的常见问题

- Assert the world
- ETA 任务与 Redis
- 查询任务详情
- 撤消任务

Assert the world

- 任务实际执行时的环境可能与你预想的不同
- 任务最好是幂等的 (idempotent)
- 设置任务的失效时间 (expired)

ETA 任务与 Redis

- 需要设置合理的 visibility_timeout
  - visibility_timeout >= max(eta_timestamp) - now
- 但 visibility_timeout 过大，将导致由于 worker 崩溃而丢失的任务无法被其他 worker 重新执行

查询任务的详情

- 很可惜， Celery 无法根据 task_id 方便的任务的方法与参数信息
- Celery 的 Result backend 只保存任务的结果
- 需要自己来维护任务详情的存储
- 建设使用 Redis 来做为任务详情的存储

撤消任务

- 任务的撤消状态 (Revoked) 是保存在 Worker 进程中
- 若要持久化这个状态，在 Worker 启动时需加 statedb 参数，或是在 Celery 的 config 文件中加 CELERYD_STATE_DB，将 Worker 状态保存在本地文件中

Celery 在 Glow 中的使用

- 主要负责发送 Push notification, Email 以及 Subscription charge
- 布署在 2 台 AWS EC2 服务器上，每台服务器起一个 Worker，并发模式 Process，并发数 4
- 由于之前提到的 Redis 与 ETA 任务的问题，我们自己开发了一个守护进程 (Etad) 来管理 ETA 任务

其他任务队列的实现

- rq: https://github.com/nvie/rq
- 优点：
  - 相较 Celery，项目的代码量小很多 (1600 行)
  - 使用 Redis 做为默认的 Broker
  - 分布式布署非常简单
- 缺点：
  - 功能单一，只能做简单的任务提交
  - RabbitMQ 在可靠性上优于 Redis。如果对可靠性要求高， RabbitMQ 是更好的 Broker

Bonus

- Celery 的 Code Philoshophy
  - The API is more important than Readability
  - Readability is more important than Convention
  - Convention is more important than Performance. Unless the code is proven hotspot

谢谢！

http://glowing.com

Slide 下载： http://vdisk.weibo.com/s/z9h0YPJPJAoEC