

# Sending million pushes

with threadpoolexecutor

# Requirements

- send batch pushes to android / iOS.
- A campaign sends to one platform and language.
- send them as fast as possible.
- Track when a send failed and persist it.

## Pitfalls

- Apple allows max 300-http2 connections
- getting millions of row from PostgreSQL need another approach of fetching data.
- django-push-notification couldn't handle ~1.5mio pushes that time.
- APNS no send\_batch

**Ultron** APP 14:04

Send Push-Notification-Campaign completed:  
Campaign: **Android Community Inspiration DE**  
Message Title: **Inspiration aus der Vergangenheit!**  
Audience: **Android Users DE**  
Installations: **264643**  
Failed: **2077**  
Delivered: **262566**  
Duration: **0:02:07.254299s**

**Ultron** APP 14:58

Send Push-Notification-Campaign completed:  
Campaign: **iOS Community Inspiration DE**  
Message Title: **Inspiration aus der Vergangenheit!**  
Audience: **iOS Users DE**  
Installations: **876413**  
Failed: **960**  
Delivered: **875453**  
Duration: **0:54:13.201422s**

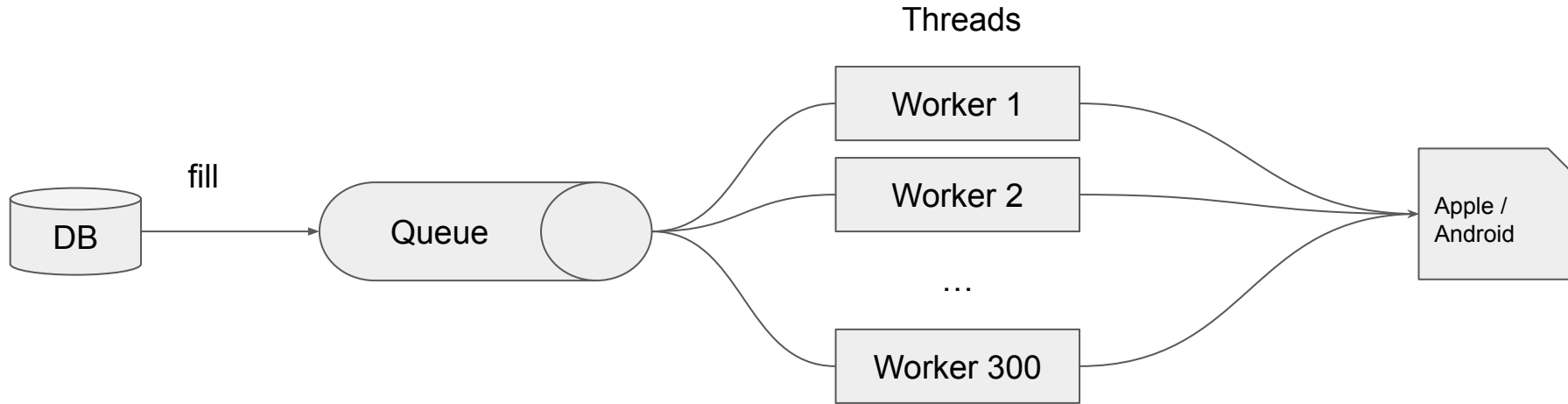
**Ultron** APP 19:03

Send Push-Notification-Campaign completed:  
Campaign: **Android Community Inspiration EN**  
Message Title: **Getting inspired by the past**  
Audience: **Android Users EN**  
Installations: **178195**  
Failed: **1936**  
Delivered: **176260**  
Duration: **0:02:01.158933s**

**Ultron** APP 20:34

Send Push-Notification-Campaign completed:  
Campaign: **iOS Community Inspiration EN**  
Message Title: **Getting inspired by the past**  
Audience: **iOS Users EN**  
Installations: **1517305**  
Failed: **3790**  
Delivered: **1513515**  
Duration: **1:30:50.782105s**

# Initial implementation



# Initial implementation

```
1 class SenderThread(threading.Thread):
2     def __init__(self, thread_id, service):
3         super(SenderThread, self).__init__()
4
5         self.thread_id = thread_id
6         self.queue = service.queue
7         self.service = service
8         self.client = service.get_client()
9
10    def run(self):
11        while not self.service.thread_exit_flag:
12            self.service.queue_lock.acquire()
13            if not self.queue.empty():
14                insta_id, push_token = self.queue.get()
15                self.service.queue_lock.release()
16                self.service.send_push_notification_to(
17                    installation_id,
18                    push_token
19                )
20            else:
21                self.service.queue_lock.release()
```

```
1 class Service:
2     # stuf
3
4     def fill_queue(self):
5         installations_qs = self.get_queryset()
6         self.installation_count = installations_qs.count()
7
8         self.queue = queue.Queue(self.installation_count)
9
10        # Fill Queue
11        self.queue_lock.acquire()
12        for installation in installations_qs.iterator():
13            self.queue.put(installation)
14        self.queue_lock.release()
15
16    def create_worker_threads(self):
17        for e in range(300):
18            fret = SenderThread(e, self)
19            self.threads.add(fret)
20            fret.start()
21
22        # Wait for queue to empty
23        while not self.queue.empty():
24            pass
25
26        # Queue is empty threads can exit now
27        self.thread_exit_flag = True
28
29        # Wait for all threads to complete
30        for fret in self.threads:
31            fret.join()
32
```

# Threadpool implementation

- Make use of `threading.local()` for thread-specific variable.
- Attach object to `threading_local`
- Queue management and locking is managed by `ThreadPoolExecutor`.
- Just handover the function and params.
- function can also be a class function with context of a class.

```
1 import threading
2
3
4 thread_local = threading.local()
5
6
7 def send_push(installation, payload):
8     if not hasattr(thread_local, 'client'):
9         thread_local.client = APNSClient(**conf)
10
11     thread_local.client.send_notification(
12         push_token=installation.token, payload=payload
13     )
14
15
16 with ThreadPoolExecutor(max_workers=300) as ex:
17     installations_qs, self.installation_count = self.get_queryset()
18
19     for installation in installations_qs.iterator():
20         ex.submit(send_push, installation, payload)
21
22     ex.shutdown(wait=True)
23
```

# Threadpool in python

```
155 def submit(self, fn, /, *args, **kwargs):
156     with self._shutdown_lock, _global_shutdown_lock:
157         if self._broken:
158             raise BrokenThreadPool(self._broken)
159
160         if self._shutdown:
161             raise RuntimeError('cannot schedule new futures after shutdown')
162         if _shutdown:
163             raise RuntimeError('cannot schedule new futures after '
164                               'interpreter shutdown')
165
166         f = _base.Future()
167         w = _WorkItem(f, fn, args, kwargs)
168
169         self._work_queue.put(w)
170         self._adjust_thread_count()
171         return f
172     submit.__doc__ = _base.Executor.submit.__doc__
173
```

```
210 def shutdown(self, wait=True, *, cancel_futures=False):
211     with self._shutdown_lock:
212         self._shutdown = True
213         if cancel_futures:
214             # Drain all work items from the queue, and then cancel their
215             # associated futures.
216             while True:
217                 try:
218                     work_item = self._work_queue.get_nowait()
219                 except queue.Empty:
220                     break
221                 if work_item is not None:
222                     work_item.future.cancel()
223
224             # Send a wake-up to prevent threads calling
225             # _work_queue.get(block=True) from permanently blocking.
226             self._work_queue.put(None)
227         if wait:
228             for t in self._threads:
229                 t.join()
230     shutdown.__doc__ = _base.Executor.shutdown.__doc__
```

# Tank you

contact:

Linkedin:

<https://www.linkedin.com/in/trung-phan-869a47130/>

