PGQ Cooperative Consumers

Dimitri Fontaine & Marko Kreen

25 Oct. 2012

Database processing oriented batches

If you're managing an *OLTP* system, you probably have out of line processing to get done, and probably are using cron batches and home made *daemons*.

Example

```
while True:
  // what a nice daemon!
```

Of course you want them

- reliable, easy to monitor and control (logs)
- out of a developer `screen` session
- easy to stop & restart
- to reuse existing models

Database processing oriented batches

If you're managing an *OLTP* system, you probably have out of line processing to get done, and probably are using cron batches and home made *daemons*.

Example

```
while True:
  // what a nice daemon!
```

Of course you want them

- reliable, easy to monitor and control (logs)
- out of a developer `screen` session
- easy to stop & restart
- to reuse existing models

Database processing oriented batches

If you're managing an *OLTP* system, you probably have out of line processing to get done, and probably are using cron batches and home made *daemons*.

Example

```
while True:
  // what a nice daemon!
```

Of course you want them

- reliable, easy to monitor and control (logs)
- out of a developer `screen` session
- easy to stop & restart
- to reuse existing models

Queueing with *PGQ*

Off-line processing is better done with PGQ

- Mainly written in PLpgSQL (and C)
- Client *API* for python
- and PHP
- some work is happening for Java
- **Cooperative Worker** (Skytools 3)

PGQ: Stable, Reliable, Easy to monitor

PGQ SQL API

PGQ is bundled as a PostgreSQL Extension, with a nice API for you to use. Let's produce some events:

- `CREATE EXTENSION pgq;`
- `select pgq.create_queue('LogEvent');`
- `select pgq.insert_event('LogEvent', 'data', 'DataFor123');`

don't forget to run the ticker daemon: `pgqd`

PGQ SQL API

PGQ is bundled as a PostgreSQL Extension, with a nice API for you to use. Let's produce some events:

- `CREATE EXTENSION pgq;`
- `select pgq.create_queue('LogEvent');`
- `select pgq.insert_event('LogEvent', 'data', 'DataFor123');`

don't forget to run the ticker daemon: `pgqd`

PGQ Consumer

PGQ Consumer: each consumer sees all the batches.

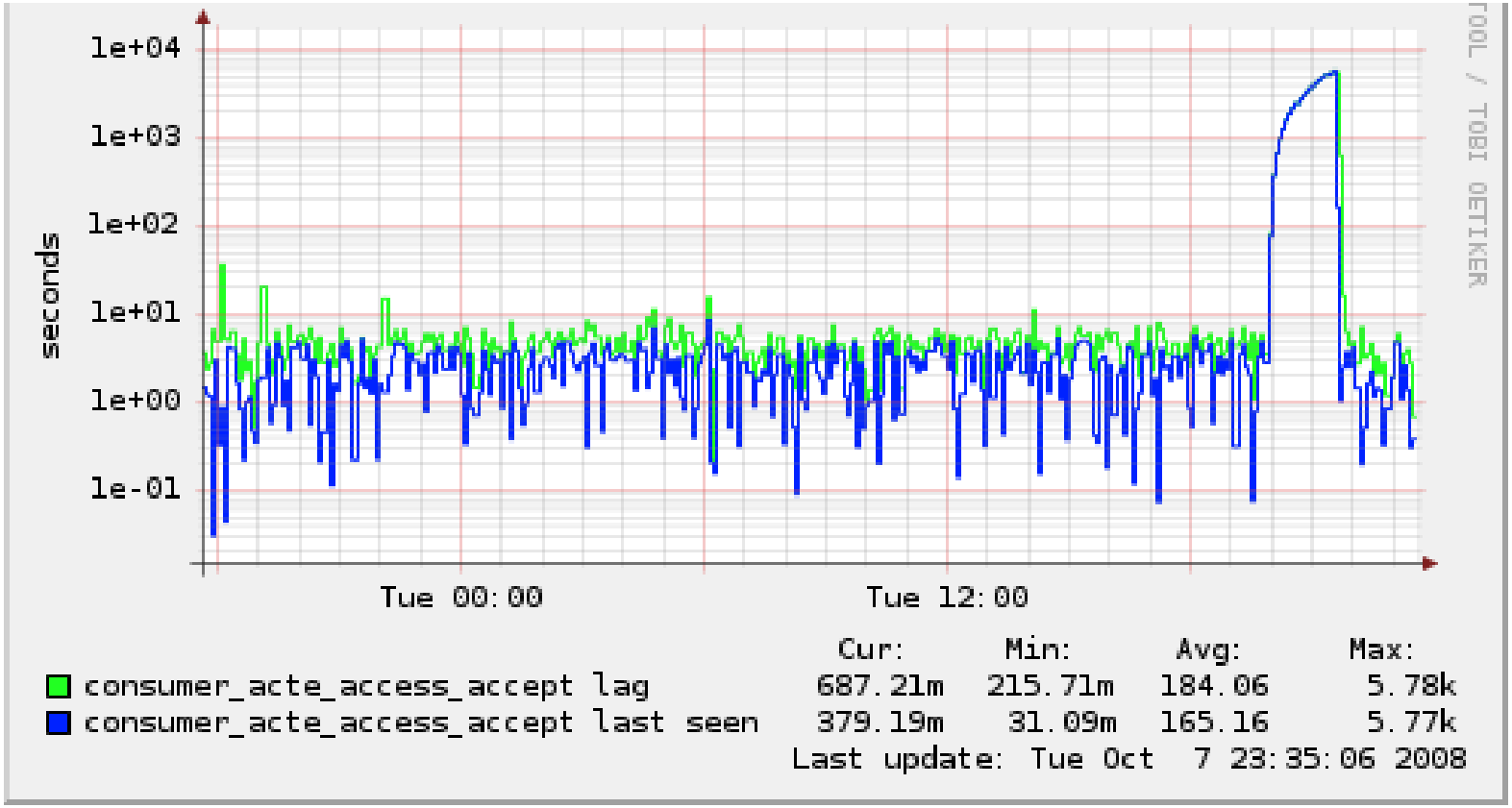
- `select pgq.register_consumer('LogEvent', 'TestConsumer');`
- `select pgq.next_batch('LogEvent', 'TestConsumer'); [into batch_id]`
- `select * from pgq.get_batch_events(batch_id);`
- `select pgq.finish_batch(batch_id)`

PGQ Consumer

PGQ Consumer: each consumer sees all the batches.

- `select pgq.register_consumer('LogEvent', 'TestConsumer');`
- `select pgq.next_batch('LogEvent', 'TestConsumer'); [into batch_id]`
- `select * from pgq.get_batch_events(batch_id);`
- `select pgq.finish_batch(batch_id)`

Monitoring PGQ Consumer



PGQ Cooperative Consumer

PGQ Cooperative Consumers share the load in between them.

- `select pgq_coop.register_subconsumer('testqueue', 'maincons', 'subcons1');`
- `select pgq_coop.next_batch('testqueue', 'maincons', 'subcons1');`
- `select * from pgq.get_batch_events(batch_id);`
- `select pgq_coop.finish_batch(batch_id)`

PGQ Cooperative Consumer

PGQ Cooperative Consumers share the load in between them.

- `select pgq_coop.register_subconsumer('testqueue', 'maincons', 'subcons1');`
- `select pgq_coop.next_batch('testqueue', 'maincons', 'subcons1');`
- `select * from pgq.get_batch_events(batch_id);`
- `select pgq_coop.finish_batch(batch_id)`

Monitoring Cooperative Consumers