

# RPC Instrumentation on Google App Engine

Guido van Rossum  
*guido@google.com*

App Engine Meetup, Palo Alto, 10/5/09

- Work in progress
- Not released yet
- Will be open source
- Will be part of SDK and runtime library
- Will be developed iteratively
- *May* be integrated with Admin Console

# Why Are Apps Slow?

Let me make a bold guess:

- Too many datastore calls
- Not enough (or ineffective) caching

*(This is not so different from why web pages are slow – see e.g. YSlow)*

# How To Find The Culprits?

- Instrumentation!
  - Always measure before spending time optimizing
  - Another lesson learned from YSlow + Firebug, etc.
    - (Of course, it's timeless wisdom :-)
- Remember the 80/20 Rule:
  - 80% of the time is spent in 20% of the code
  - (Sometimes 90/10 :-)

# What Is An RPC?

- RPC = Remote Procedure Call
- Roundtrip between app and back-end server
- Example:
  - `db.get()` or `db.put()`
  - `urlfetch.fetch()`
  - `mail.send()`
- Iterating over query can be multiple RPCs
  - however, results are batched (typically N=20)

# Why Are Datastore Calls Important?

*(Disclaimer: these are pretty unscientific numbers!)*

- Typical db.put(): 50-100 msec real time
- Typical db.get(): 10-50 msec
- Even a memcache call is 3-10 msec real time
- Much time spent waiting for network or disk
- For comparison, template expansion for a beefy template in my app is 80-100 msec  
*(obviously this depends on the app :-)*

# Performance Insights

- For RPCs like datastore and memcache, network latency is a big factor
- A single RPC to get 2 objects using the multiple-get API is much faster than two RPCs!
- Same for memcache—use the ‘multi’ APIs
- Conclusion: find unnecessary round trips!

# Common Performance “Bugs”

- Not caching “hot” items
- Fetching the same entity twice
- Fetching many entities one at a time
- Cache management bugs
- Using a query instead of `db.get()`

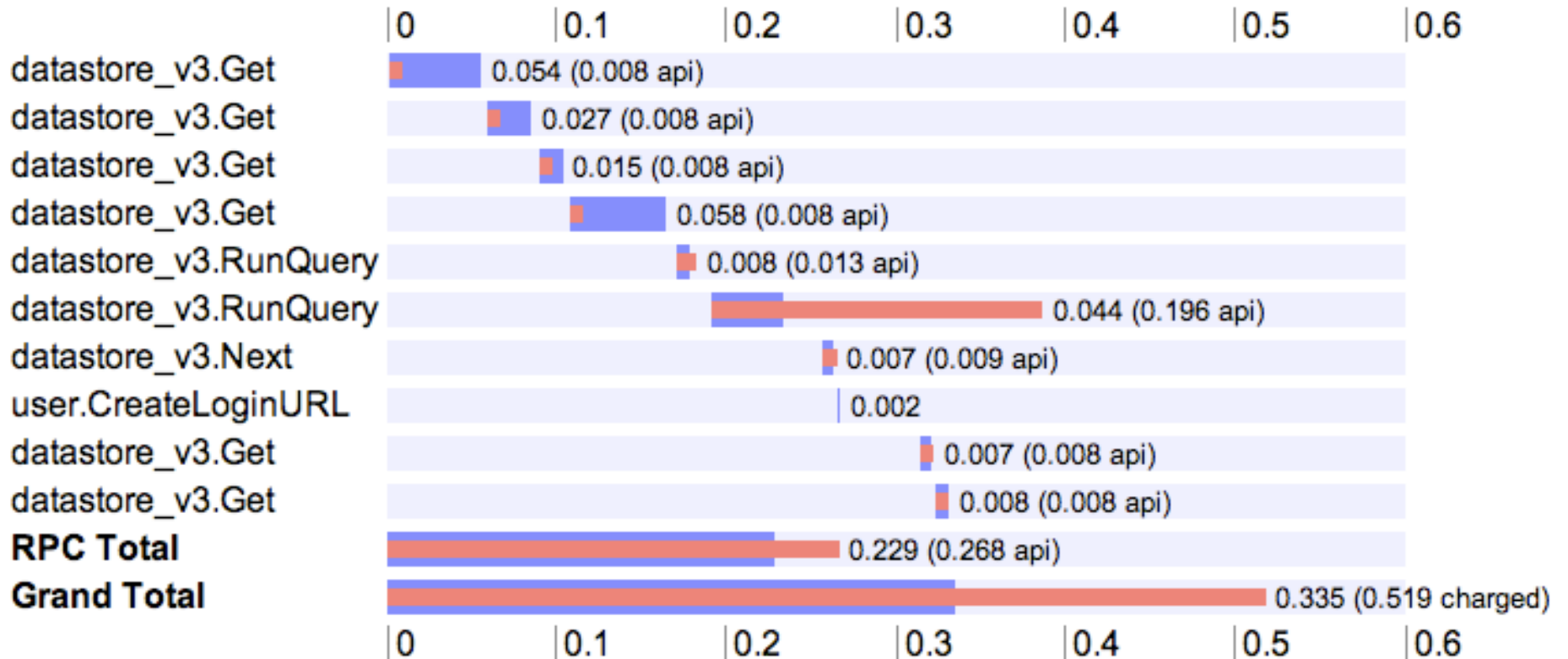
*(These are often easy to miss because they don't cause incorrect behavior, just lack of speed)*

# How Does AppStats Help?

- RPC instrumentation with lots of extras
- Helps find “hot” requests and RPC calls
- Drill down to stack frames and locals
- Not just datastore; instruments all RPCs
- Timeline view of individual request shows exactly where your time goes

# DEMO

## Timeline (seconds)



*How to read this chart:* Blue bars show the real time spent on an API call; the left and right edges of each bar show the call's start and end times. All times are in seconds. Superimposed red bars show API time charged for the call. At the bottom, **RPC Total** shows the total real time spent in RPCs, with the total API time superimposed, and **Grand Total** shows the total real time spent processing the entire request, with the total CPU time charged (server time and API time) superimposed. Actual time charged may be more due to the overhead of saving the stats data.

# How It Works

- Uses Call Hooks (see Rafe's talk)
  - pre-call hook records start time, request, stack
  - post-call hook records end time, mcycles, response
- Measuring times:
  - real time using Python's wall clock API
  - API megacycles using `rpc.cpu_usage_mcycles`
  - CPU megacycles using `quota.get_request_cpu_usage()`
- Get stack contents from Python's `sys._getframe()`
- Recording accumulates in memory object
- Written to memcache at end of each request

# Tricky Things (For Me :-)

- Making a fast and robust repr()-look-alike (the built-in is too slow)
- How to log without requiring infinite storage
  - solution: use time mod N as key
- Minimizing overhead (still only partially resolved)
- What to do if the app reaches many QPS
  - solution: memcache-based advisory lock with time-out
- Making the UI fast enough
  - solution: store small summary records and large details records
- Making it easy to configure (both turning it on and customizing it)
- Making the “by path” statistics useful
  - solution: custom path normalization function
- Displaying the custom GANTT-style timeline chart
- Making the UI pretty (not my thing :-)

# TO DO

- Finish the UI
- Support custom events with duration
- Reduce overhead
- Admin Console integration?
- Add CPU profiling option?
- Java version

# AppStats Was Inspired By...

- Real bugs in real apps
- Firebug (timeline)
- Google's internal rpc stats instrumentation
  - (much more powerful, but not easily adapted to App Engine's Python runtime)
- Lunchtime discussions with Googlers
- Early trusted testers