
Google™
10

Appstats – RPC Instrumentation and Optimizations for App Engine

Guido van Rossum

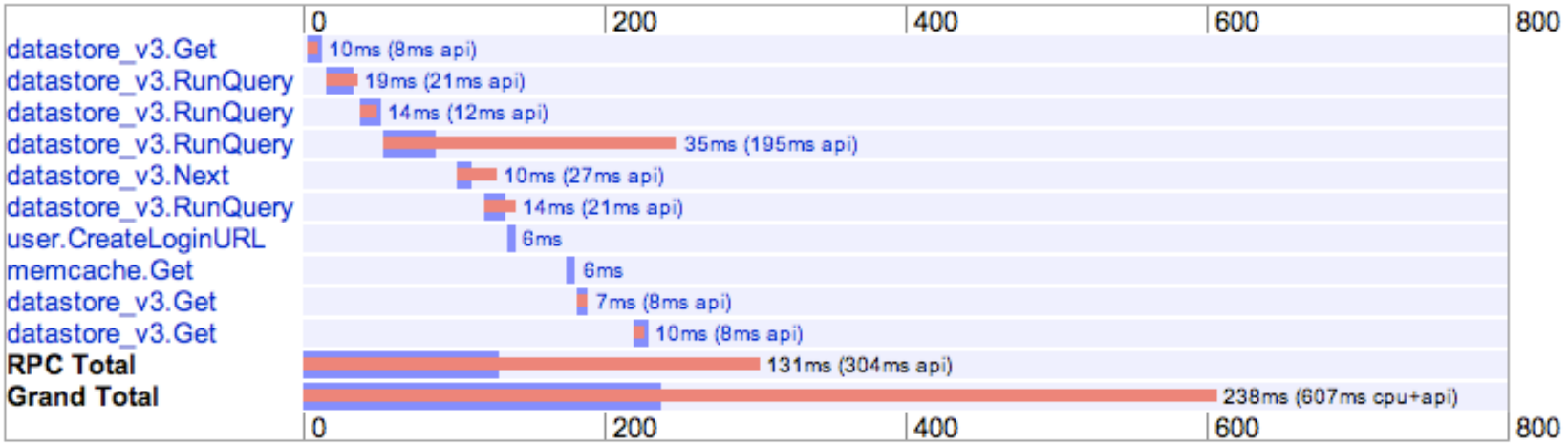
May 19, 2010

Overview

- **Introduction**
- Demo
- Background and implementation
- Configuration
- Specific performance issues
- Q&A (wave: http://bit.ly/appengine1)
- Hashtag: #appengine1

Appstats: High Order Bits

- Tool library specific to App Engine
- Analyzes RPC performance
- Python: released in February (1.3.1)
- Java: released in March (1.3.2)



"I used to be blind, but now I can see :-)"

Early Appstats User

Appstats: Basics

- All user-land code, linked with your app
  - Moderate run-time overhead (details later)
- Uses existing instrumentation APIs
  - (Some Python, some App Engine)
- No download required
  - But must enable / configure (a little)
- User interface is part of the library
  - Instant results

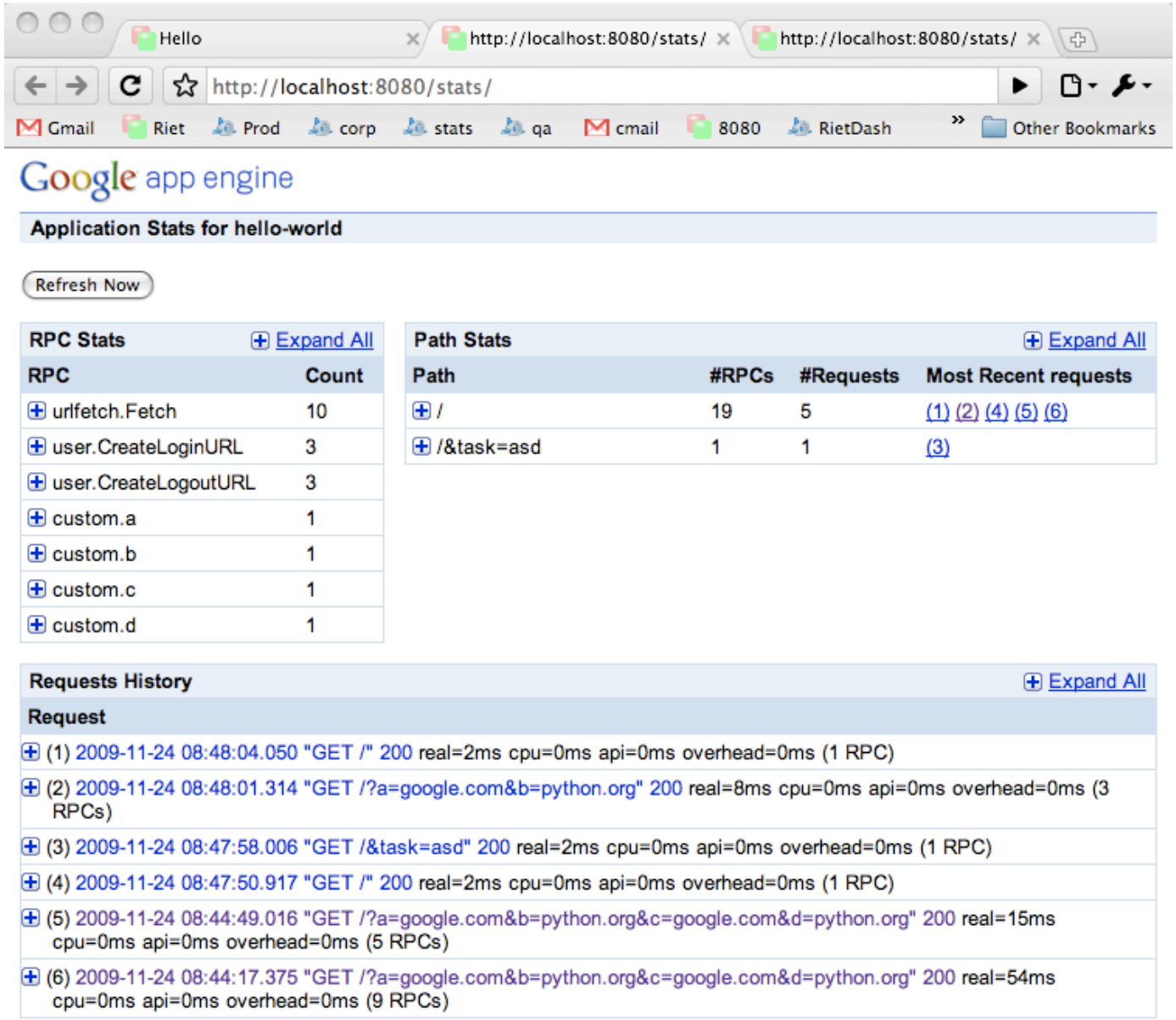
Appstats: What Is It Good For

- Finds “hot” requests and RPC calls
  - Tells you what to tackle first
- Records all RPCs, not just datastore calls
  - Properly handles async RPCs (like urlfetch)!
- Timeline view of individual requests
  - Shows where the time goes
- Drill down to stack frames and locals
  - (Python only)

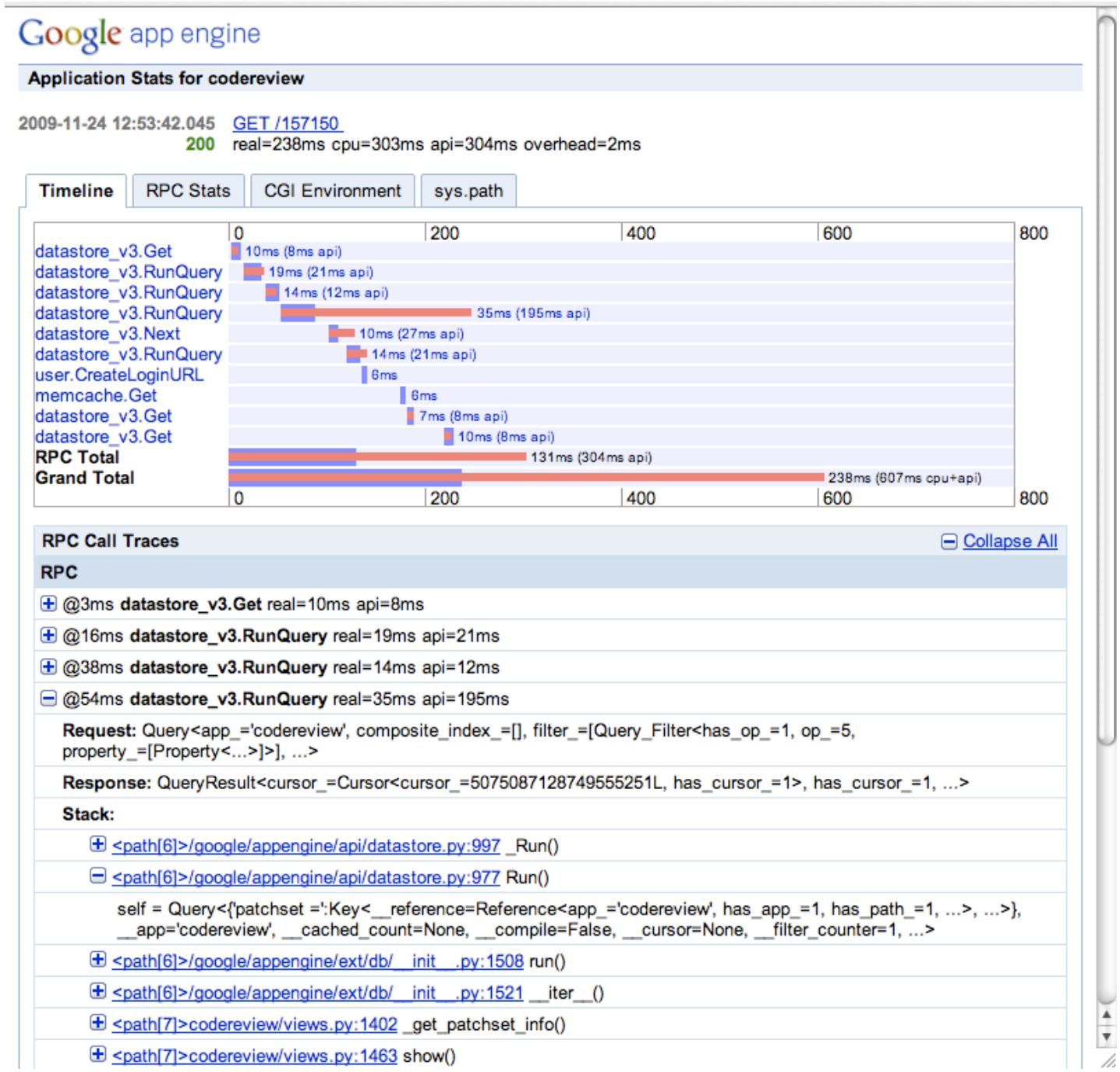
Overview

- Introduction
- **Demo**
- Background and implementation
- Configuration
- Specific performance issues
- Q&A (wave: http://bit.ly/appengine1)
- Hashtag: #appengine1

DEMO


Hello
http://localhost:8080/stats/
http://localhost:8080/stats/
http://localhost:8080/stats/
Gmail
Riet
Prod
corp
stats
qa
cmail
8080
RietDash
Other Bookmarks
Google app engine
Application Stats for hello-world
Refresh Now
RPC Stats
Expand All
RPC
Count
urlfetch.Fetch
10
user.CreateLoginURL
3
user.CreateLogoutURL
3
custom.a
1
custom.b
1
custom.c
1
custom.d
1
Path Stats
Expand All
Path
#RPCs
#Requests
Most Recent requests
/
19
5
(1) (2) (4) (5) (6)
/&task=asd
1
1
(3)
Requests History
Expand All
Request
(1) 2009-11-24 08:48:04.050 "GET /" 200 real=2ms cpu=0ms api=0ms overhead=0ms (1 RPC)
(2) 2009-11-24 08:48:01.314 "GET /?a=google.com&b=python.org" 200 real=8ms cpu=0ms api=0ms overhead=0ms (3 RPCs)
(3) 2009-11-24 08:47:58.006 "GET /&task=asd" 200 real=2ms cpu=0ms api=0ms overhead=0ms (1 RPC)
(4) 2009-11-24 08:47:50.917 "GET /" 200 real=2ms cpu=0ms api=0ms overhead=0ms (1 RPC)
(5) 2009-11-24 08:44:49.016 "GET /?a=google.com&b=python.org&c=google.com&d=python.org" 200 real=15ms cpu=0ms api=0ms overhead=0ms (5 RPCs)
(6) 2009-11-24 08:44:17.375 "GET /?a=google.com&b=python.org&c=google.com&d=python.org" 200 real=54ms cpu=0ms api=0ms overhead=0ms (9 RPCs)

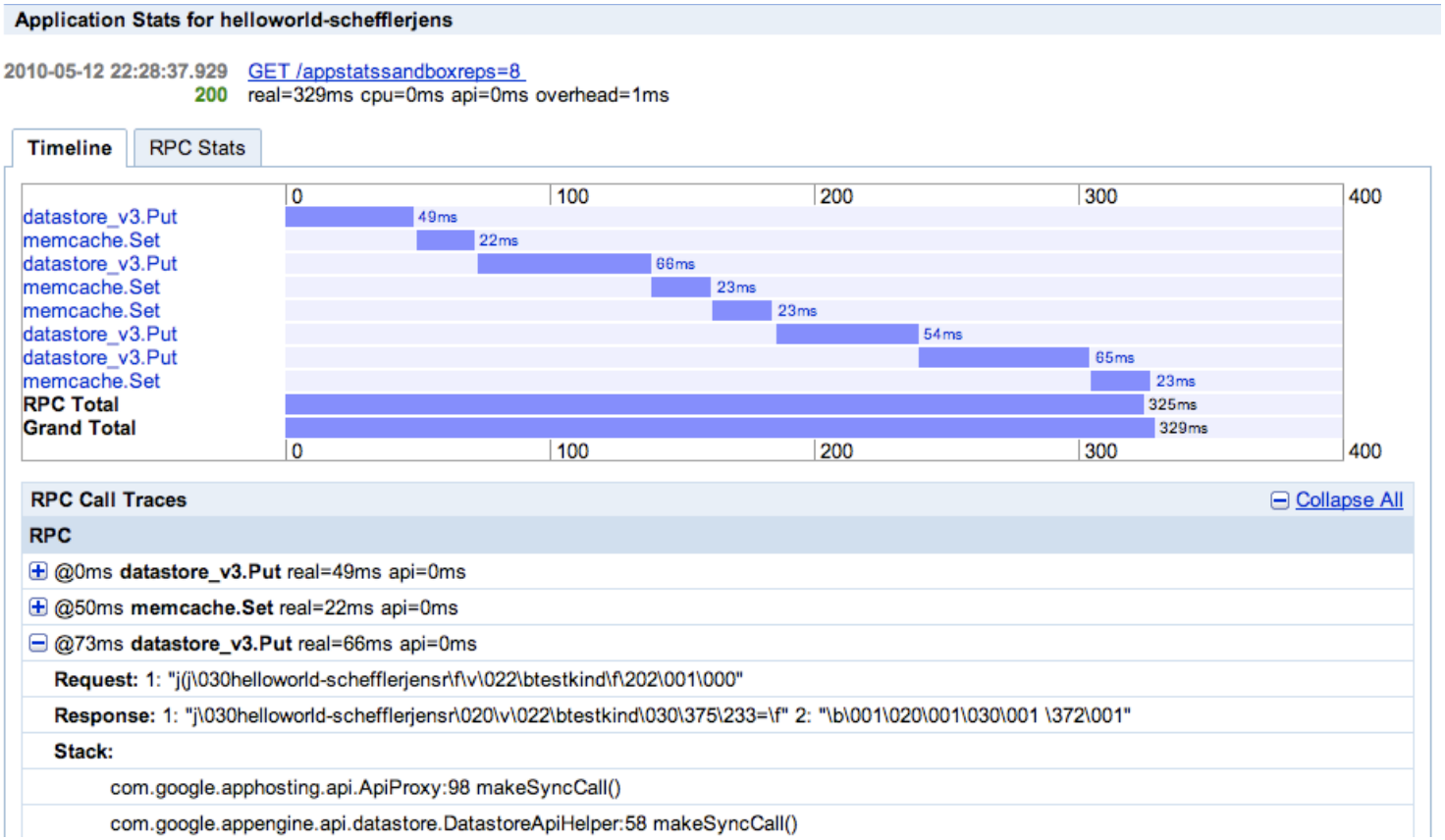
DEMO


Google app engine
Application Stats for codereview
2009-11-24 12:53:42.045 GET /157150
200 real=238ms cpu=303ms api=304ms overhead=2ms
Timeline
RPC Stats
CGI Environment
sys.path
0
200
400
600
800
datastore_v3.Get 10ms (8ms api)
datastore_v3.RunQuery 19ms (21ms api)
datastore_v3.RunQuery 14ms (12ms api)
datastore_v3.RunQuery 35ms (195ms api)
datastore_v3.Next 10ms (27ms api)
datastore_v3.RunQuery 14ms (21ms api)
user.CreateLoginURL 6ms
memcache.Get 6ms
datastore_v3.Get 7ms (8ms api)
datastore_v3.Get 10ms (8ms api)
RPC Total 131ms (304ms api)
Grand Total 238ms (607ms cpu+api)
RPC Call Traces
Collapse All
RPC
@3ms datastore_v3.Get real=10ms api=8ms
@16ms datastore_v3.RunQuery real=19ms api=21ms
@38ms datastore_v3.RunQuery real=14ms api=12ms
@54ms datastore_v3.RunQuery real=35ms api=195ms
Request: Query<app_='codereview', composite_index_=[], filter_=[Query_Filter<has_op_=1, op_=5, property_=[Property<...>]>], ...>
Response: QueryResult<cursor_=Cursor<cursor_=5075087128749555251L, has_cursor_=1>, has_cursor_=1, ...>
Stack:
<path[6]>/google/appengine/api/datastore.py:997 _Run()
<path[6]>/google/appengine/api/datastore.py:977 Run()
self = Query<{'patchset =':Key<__reference=Reference<app_='codereview', has_app_=1, has_path_=1, ...>, ...>}, __app='codereview', __cached_count=None, __compile=False, __cursor=None, __filter_counter=1, ...>
<path[6]>/google/appengine/ext/db/__init__.py:1508 run()
<path[6]>/google/appengine/ext/db/__init__.py:1521 __iter__()
<path[7]>codereview/views.py:1402 _get_patchset_info()
<path[7]>codereview/views.py:1463 show()

[Do a real demo at this point]

- Click around
- Point out stats
- Go to details page
- Show all four tabs briefly
- Expand one RPC
- Expand stack frame
- View source

Java Too!



See talk by Don Schwarz and Toby Reyelts:
“What’s hot In Java for App Engine” – 11:30 tomorrow

Google I/O 10

Overview

- Introduction
- Demo
- **Background and implementation**
- Configuration
- Specific performance issues
- Q&A (wave: http://bit.ly/appengine1)
- Hashtag: #appengine1

What Exactly Is An RPC?

- RPC = Remote Procedure Call
- 1 roundtrip between app and back-end
- Example:
  - db.get(), db.put(), db.query()
  - memcache.get(), memcache.get_multi()
  - urlfetch.fetch()
  - mail.send()
- Iterating over query *may* use multiple RPCs
  - however, results are batched (typically N=20)

Why Are RPCs So Important?

- Single datastore put: 50-100 msec real time
- Get: 10-20 msec; query: 20-100 msec
- Even a memcache call is 5-10 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

Appstats: How It Works

- Call Hooks used to record per-RPC details
- Kept in memory while request runs
- Written to memcache at end of each request
  - Two proto-buffers written for each request:
    - short summary
    - long details record
- UI loads data from memcache
  - overview page scans all summaries

Call Hooks

- Standard (Python) App Engine Feature
  - http://code.google.com/appengine/articles/hooks.html
  - User code called before and after each RPC
- How Appstats uses call hooks:
  - pre-call hook records start time, request object, stack frames
  - post-call hook records end time, response object, resource usage

Measuring Resource Usage

- Real time: Python's wall clock API: time.time()
- API usage: measured in virtual megacycles
  - reported as rpc.cpu_usage_mcycles (per RPC)
- CPU usage: measured in virtual megacycles
  - reported by quota.get_request_cpu_usage() (total for current request)
- Stack frame: Python's sys._getframe()

Memcache Key Space

- Use time mod N for key to limit memory use
- Configurable formula:
  - key = (time rounded to A msecs) mod B seconds
  - defaults: A=100ms, B=100sec; i.e. 1000 keys
- When QPS goes through the roof:
  - rate limited to min(QPS of 1 CPU, 1 QPS)
- Space used per record:
  - typically 10KB – 100KB; max 1MB

Appstats: Overhead

- Time overhead is logged, e.g.:
  - Saved; key: __appstats__:048400, part: 136 bytes, full: 117838 bytes, overhead: 0.004 + 0.049; link: http://...
  - IOW recording cost 4ms, saving 49ms *(YMMV!)*
- Memcache usage estimate:
    - 1000 keys x 100KB per key == 100MB
- Is it lean enough for popular apps?
  - In practice: fine in Rietveld (code review app)
  - Much can be learned from use with SDK too
  - Could use randomized fraction of requests

Overview

- Introduction
- Demo
- Background and implementation
- **Configuration**
- Specific performance issues
- Q&A (wave: http://bit.ly/appengine1)
- Hashtag: #appengine1

Appstats: Basic Configuration

- Enable recording in appengine_config.py:
  - def webapp_add_wsgi_middleware(app):
    from google.appengine.ext.appstats import recording
    app = recording.appstats_wsgi_middleware(app)
    return app
- Enable UI in app.yaml:
  - - url: /stats.*
    script: $PYTHON_LIB/google/appengine/ext/appstats/ui.py
- Interact with app for a while
- Point browser at /stats/ (admin restricted)
- Java users: see docs

Important Note

- You *must* use webapp.util.run_wsgi_app():

```
from google.appengine.ext.webapp.util import run_wsgi_app

def main():
  app = ......
  run_wsgi_app(app)

if __name__ == "__main__”:
  main()
```

- This does not mean you have to use the rest of webapp! (works fine with e.g. Django)
  - Don’t use wsgiref.handlers.CGIHandler().run(app)

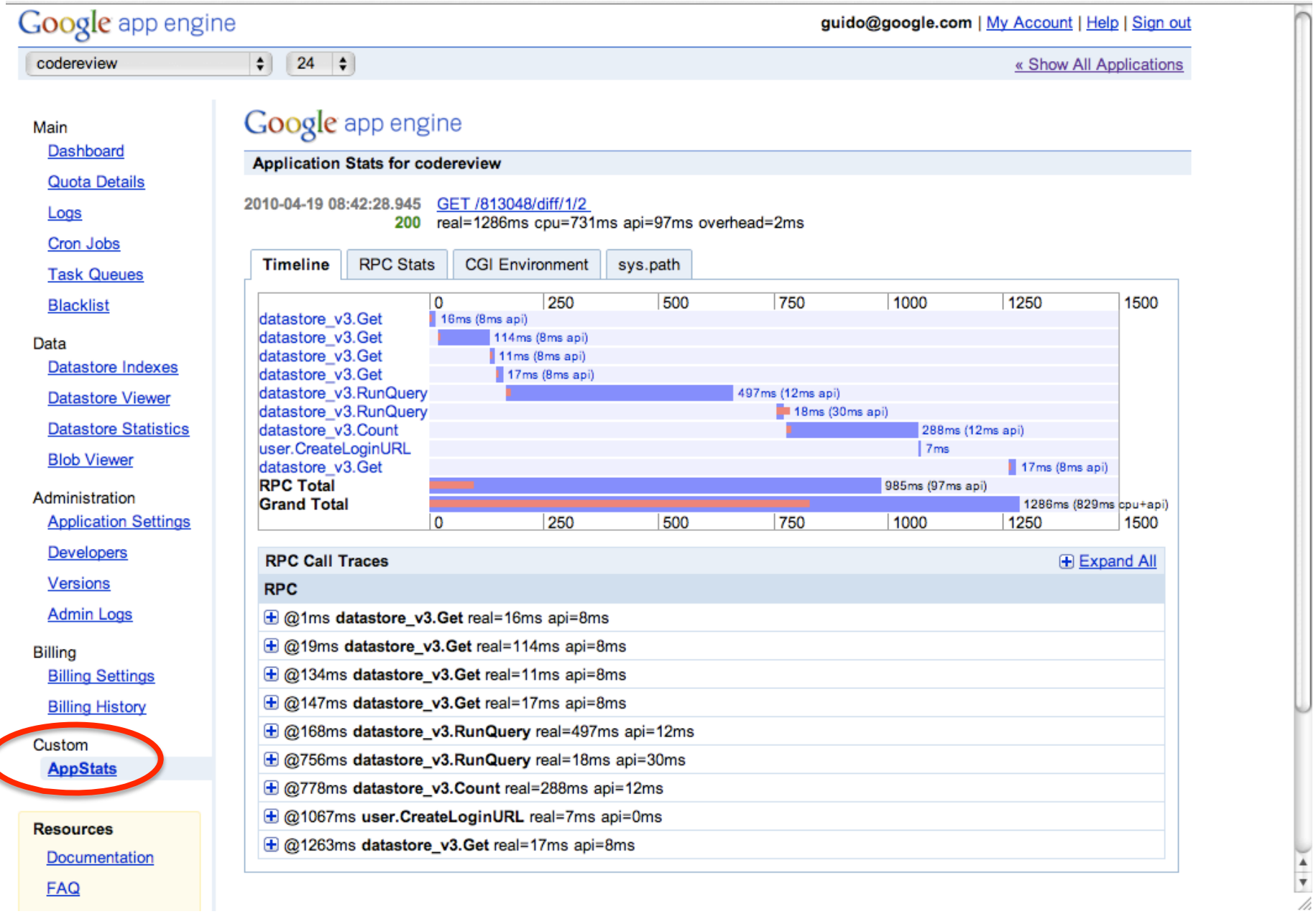
Appstats: Advanced Configuration

- Check out sample_appengine_config.py in SDK
- Can configure many things, e.g.:
  - Key rotation scheme
  - Max stack depth, max #variables, max repr size
  - Regex to skip certain stack frames
  - Record only selected events
  - Record a randomized fraction of all events
  - Path normalization (for the bins in the UI)

Advanced Tip / Feature Plug

- To link Admin Console to Appstats, add this to app.yaml:

```
admin_console:
 pages:
 - name: Appstats
   url: /stats
```


Google app engine
guido@google.com | My Account | Help | Sign out
codereview
24
« Show All Applications
Main
Dashboard
Quota Details
Logs
Cron Jobs
Task Queues
Blacklist
Data
Datastore Indexes
Datastore Viewer
Datastore Statistics
Blob Viewer
Administration
Application Settings
Developers
Versions
Admin Logs
Billing
Billing Settings
Billing History
Custom
AppStats
Resources
Documentation
FAQ
Google app engine
Application Stats for codereview
2010-04-19 08:42:28.945 GET /813048/diff/1/2
200 real=1286ms cpu=731ms api=97ms overhead=2ms
Timeline
RPC Stats
CGI Environment
sys.path
0 250 500 750 1000 1250 1500
datastore_v3.Get 16ms (8ms api)
datastore_v3.Get 114ms (8ms api)
datastore_v3.Get 11ms (8ms api)
datastore_v3.Get 17ms (8ms api)
datastore_v3.RunQuery 497ms (12ms api)
datastore_v3.RunQuery 18ms (30ms api)
datastore_v3.Count 288ms (12ms api)
user.CreateLoginURL 7ms
datastore_v3.Get 17ms (8ms api)
RPC Total 985ms (97ms api)
Grand Total 1286ms (829ms cpu+api)
0 250 500 750 1000 1250 1500
RPC Call Traces
Expand All
RPC
@1ms datastore_v3.Get real=16ms api=8ms
@19ms datastore_v3.Get real=114ms api=8ms
@134ms datastore_v3.Get real=11ms api=8ms
@147ms datastore_v3.Get real=17ms api=8ms
@168ms datastore_v3.RunQuery real=497ms api=12ms
@756ms datastore_v3.RunQuery real=18ms api=30ms
@778ms datastore_v3.Count real=288ms api=12ms
@1067ms user.CreateLoginURL real=7ms api=0ms
@1263ms datastore_v3.Get real=17ms api=8ms

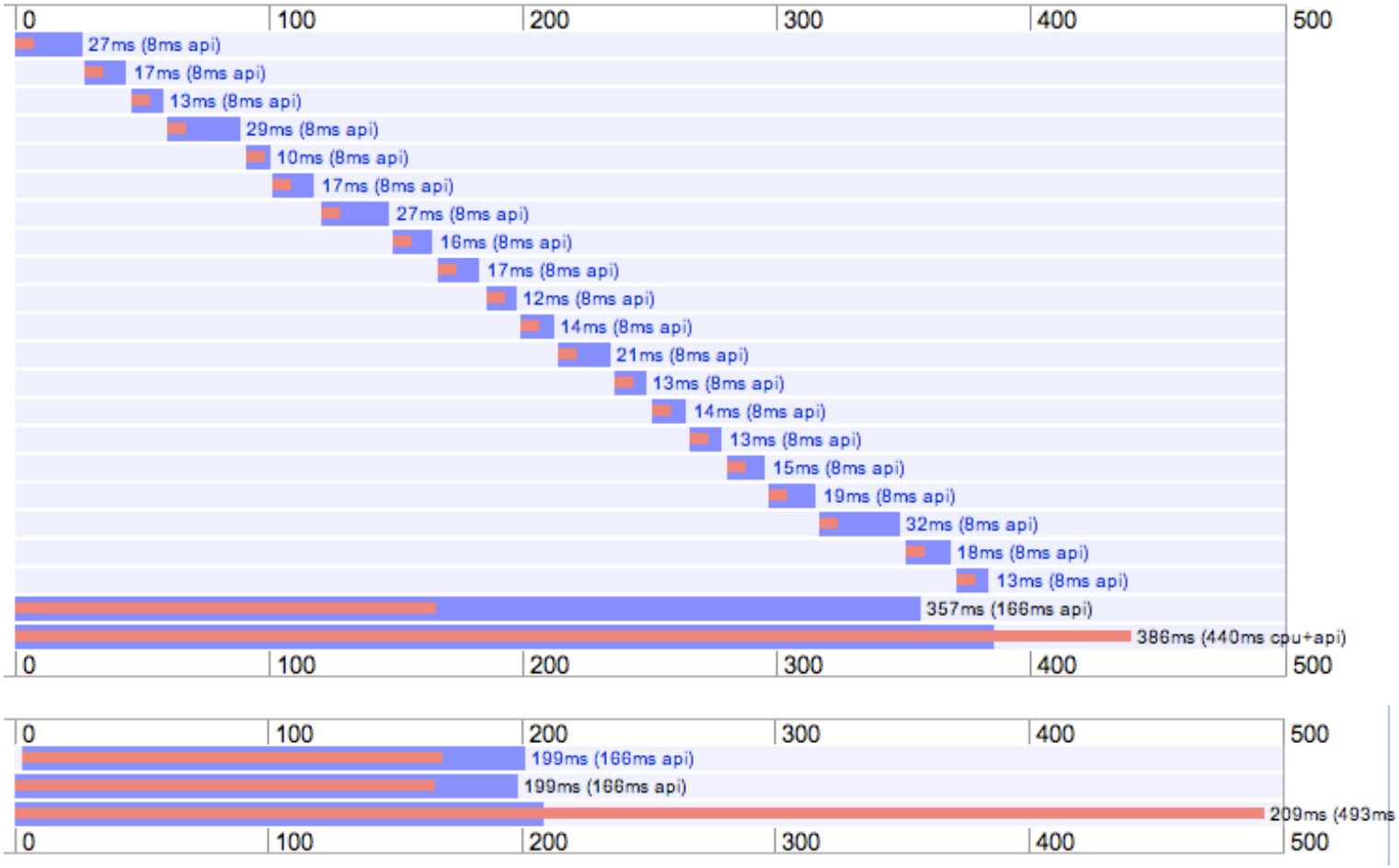
Overview

- Introduction
- Demo
- Background and implementation
- Configuration
- **Specific performance issues**
- Q&A (wave: http://bit.ly/appengine1)
- Hashtag: #appengine1

Why Perf Issues Are Common

- Don’t cause incorrect behavior, just slowness
- Typical tests don’t look for them
- It’s often death by a thousand cuts
- Web app performance is naturally variable

Why Fixing Perf Issues Matters



1. Not Caching Hot Items

- Problem: you don't know what to cache
- Solution: use Appstats to find out
- Surprise: you might not even realize which entities a request is loading!

2. Cache Management Bugs

- Problem: your cache code has a bug
  - Real-life example: Rietveld caching code was loading data from datastore *even if it had just retrieved it from the cache*
- Solution: Appstats shows that you are still loading from datastore
  - The detail view will show you what was returned from memcache.Get() too

3. Query Caching Bug

- Real code found in an internal app:

```
states = memcache.get(memcache_key)
if not states:
    states = models. State.gql("WHERE foobar = :1", foobar)
    memcache.set(memcache_key, states)
for state in states:
    ......
```

- Should have been:

```
models. State.gql("WHERE foobar = :1", foobar).fetch(1000)
```

- Was caching query object instead of results!

4. Fetching The Same Entity Twice

- Problem: different components of your app are each loading the same entity
  - Probable cause: too much abstraction causes the components to be unaware of each other's uses
- Solution: Appstats shows keys in requests
  - NOTE: memcache is *not* the solution here; you should be able to pass the entity around between components (maybe after some refactoring)

5. Fetching Items One At A Time

- Problem: Appstats shows a staircase pattern
  - many short (datastore or memcache) Get calls
- Likely cause: Get called in loop body
- Solution: use db.get([list_of_keys]) or memcache.get_multi([list_of_keys])
- http://blog.appenginefan.com/2010/04/patterns-of-doom-staircase-of-gets.html (Jens Scheffler)

6. Fetching Several External URLs

- Problem: urlfetch.fetch() can be slow
- Solution: use asynchronous urlfetch requests!
- http://blog.appenginefan.com/2010/04/patterns-of-doom-chain-of-fetches.html

7. Loop Over ReferenceProperties

- Especially nasty version of case 5
- Often caused by template following reference properties in a loop over query results
  - Python code does a single query
  - Template accesses properties of results
  - Each property access does an implied Get
- Solution: prefetching (due to Nick Johnson)

Prefetching ReferenceProperties

```
posts = Post.gql("WHERE ...").fetch(20)
prop = Post.author  # I.e. the ReferenceProperty object
author_keys = [prop.get_value_for_datastore(p)
                        for p in posts]
authors = dict((x.key(), x) for x in db.get(set(author_keys)))
for post, author_key in zip(posts, author_keys):
    post.author = authors[author_key]
template.render("posts.html", {"posts": posts})
```

- Nick Johnson's blog has generic solution:
  - http://blog.notdot.net/2010/01/ReferenceProperty-prefetching-in-App-Engine

Q & A

- *Live wave:* http://bit.ly/appengine1
- *Hashtag:* #appengine1

- *Docs:*
  - http://code.google.com/appengine/docs/**python**/tools/appstats.html
  - http://code.google.com/appengine/docs/**java**/tools/appstats.html