Operation (2)

November 17, 2022

Byung-Gon Chun

(Credit: Slides from UCB CS169 taught by Armando Fox, David Patterson)

Avoiding Abusive Queries

Be kind to the database

- Outgrowing single-machine database == big investment: sharding, replication, etc.
- Alternative: find ways to relieve pressure on database so can stay in “PaaS-friendly” tier
  1. Use caching to reduce number of database accesses
  2. Avoid “n+1 queries” problem in association
  3. Use indices judiciously

n+1 queries problem

- **Problem:** you are doing n+1 queries to traverse an association, rather than 1 query
  (a common performance anti-pattern)
- E.g., you need to iterate through all the cars, and for each one, print out a list of the wheels. The naive O/R implementation

  ```
  SELECT * FROM Cars;
  for each Car:
     SELECT * FROM Wheel WHERE CarId = ?
  ```

  one select for the Cars, and then N additional selects, where N is the total number of cars.

n+1 queries problem

- select_related: django to do a join when fetching data.
  You should use it on any lookup where you know you'll need related fields.

```
e = Entry.objects.get(id=5) # Hits the db
b = e.blog # Hits the db again

=> e = Entry.objects.select_related(‘blog’).get(id=5) # Hits the db
      b = e.blog # Doesn’t hit the db
```

```
class Blog(models.Model):
…
class Entry(models.Model):
      blog = models.ForeignKey(Blog, on_delete=models.CASCADE)
…
```

Table Scan Solved by Indices

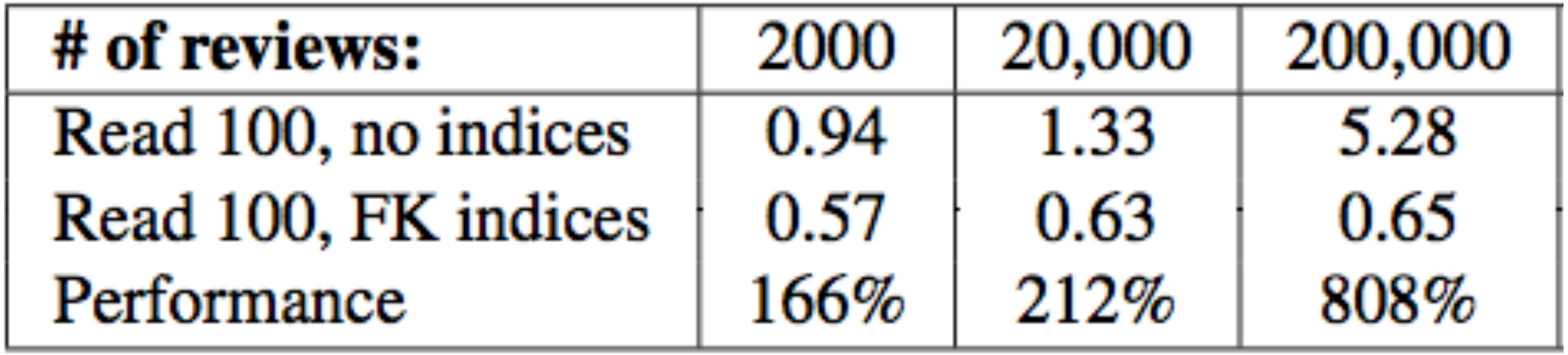
- Speeds up access when searching DB table by column other than primary key
- Similar to using a hash table
  - alternative is *table scan*—bad!;
    Taking time O(n) for a table with n rows
  - even bigger win if attribute is unique-valued
- Why not index *every* column?
  - takes up space
  - all indices must be updated when table updated

What to index?

- Foreign key columns
- Columns that appear in `where()` clauses
- Columns on which you sort

How much does indexing help?

(Numbers depend on environments, workloads, etc.)



# of reviews:	2000	20,000	200,000
Read 100, no indices	0.94	1.33	5.28
Read 100, FK indices	0.57	0.63	0.65
Performance	166%	212%	808%

Database as a Service

- Access to a database without the need for setting up physical hardware, installing software or configuring for performance

- Autoscaling (hopefully) handled

Monitoring

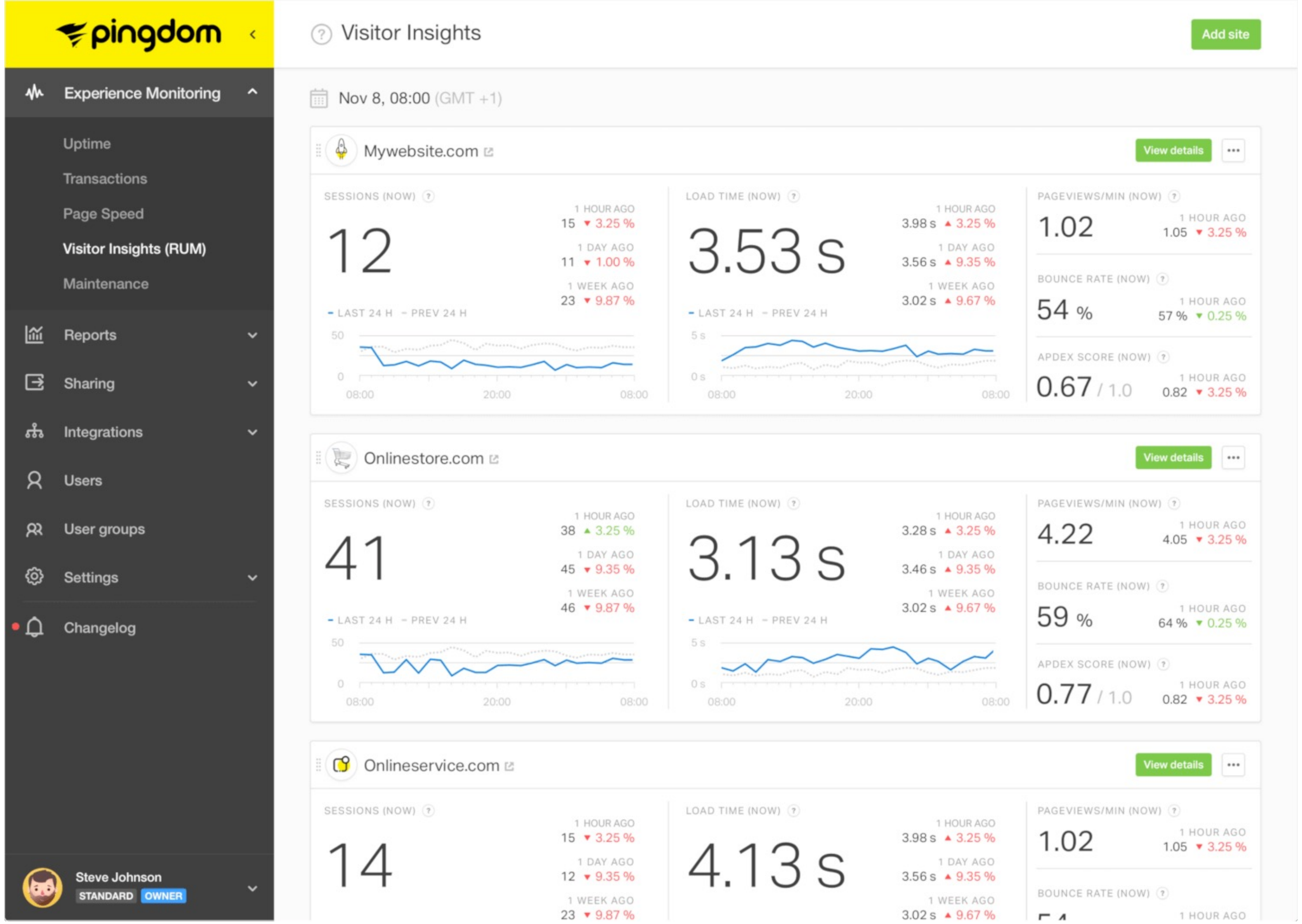
Kinds of monitoring

- **“If you’ re not monitoring it, it’ s probably broken”**
- At development time (*profiling*)
  - – Identify possible performance/stability problems *before* they get to production
- In production
  - – Internal: instrumentation embedded in app and/or framework
  - – External: active probing by other site(s).

Why use external monitoring?

- Detect if site is down
- Detect if site is slow for reasons outside measurement boundary of internal monitoring
- Get user’s view from many different places on the Internet
- Example: Pingdom

pingdom



Visitor Insights

Nov 8, 08:00 (GMT +1)

Mywebsite.com

SESSIONS (NOW) 12
1 HOUR AGO 15 ▼ 3.25 %
1 DAY AGO 11 ▼ 1.00 %
1 WEEK AGO 23 ▼ 9.87 %

LOAD TIME (NOW) 3.53 s
1 HOUR AGO 3.98 s ▲ 3.25 %
1 DAY AGO 3.56 s ▲ 9.35 %
1 WEEK AGO 3.02 s ▲ 9.67 %

PAGEVIEWS/MIN (NOW) 1.02
1 HOUR AGO 1.05 ▼ 3.25 %

BOUNCE RATE (NOW) 54 %
1 HOUR AGO 57 % ▼ 0.25 %

APDEX SCORE (NOW) 0.67 / 1.0
1 HOUR AGO 0.82 ▼ 3.25 %

Onlinestore.com

SESSIONS (NOW) 41
1 HOUR AGO 38 ▲ 3.25 %
1 DAY AGO 45 ▼ 9.35 %
1 WEEK AGO 46 ▼ 9.87 %

LOAD TIME (NOW) 3.13 s
1 HOUR AGO 3.28 s ▲ 3.25 %
1 DAY AGO 3.46 s ▲ 9.35 %
1 WEEK AGO 3.02 s ▲ 9.67 %

PAGEVIEWS/MIN (NOW) 4.22
1 HOUR AGO 4.05 ▼ 3.25 %

BOUNCE RATE (NOW) 59 %
1 HOUR AGO 64 % ▼ 0.25 %

APDEX SCORE (NOW) 0.77 / 1.0
1 HOUR AGO 0.82 ▼ 3.25 %

Onlineservice.com

SESSIONS (NOW) 14
1 HOUR AGO 15 ▼ 3.25 %
1 DAY AGO 12 ▼ 9.35 %
1 WEEK AGO 23 ▼ 9.87 %

LOAD TIME (NOW) 4.13 s
1 HOUR AGO 3.98 s ▲ 3.25 %
1 DAY AGO 3.56 s ▲ 9.35 %
1 WEEK AGO 3.02 s ▲ 9.67 %

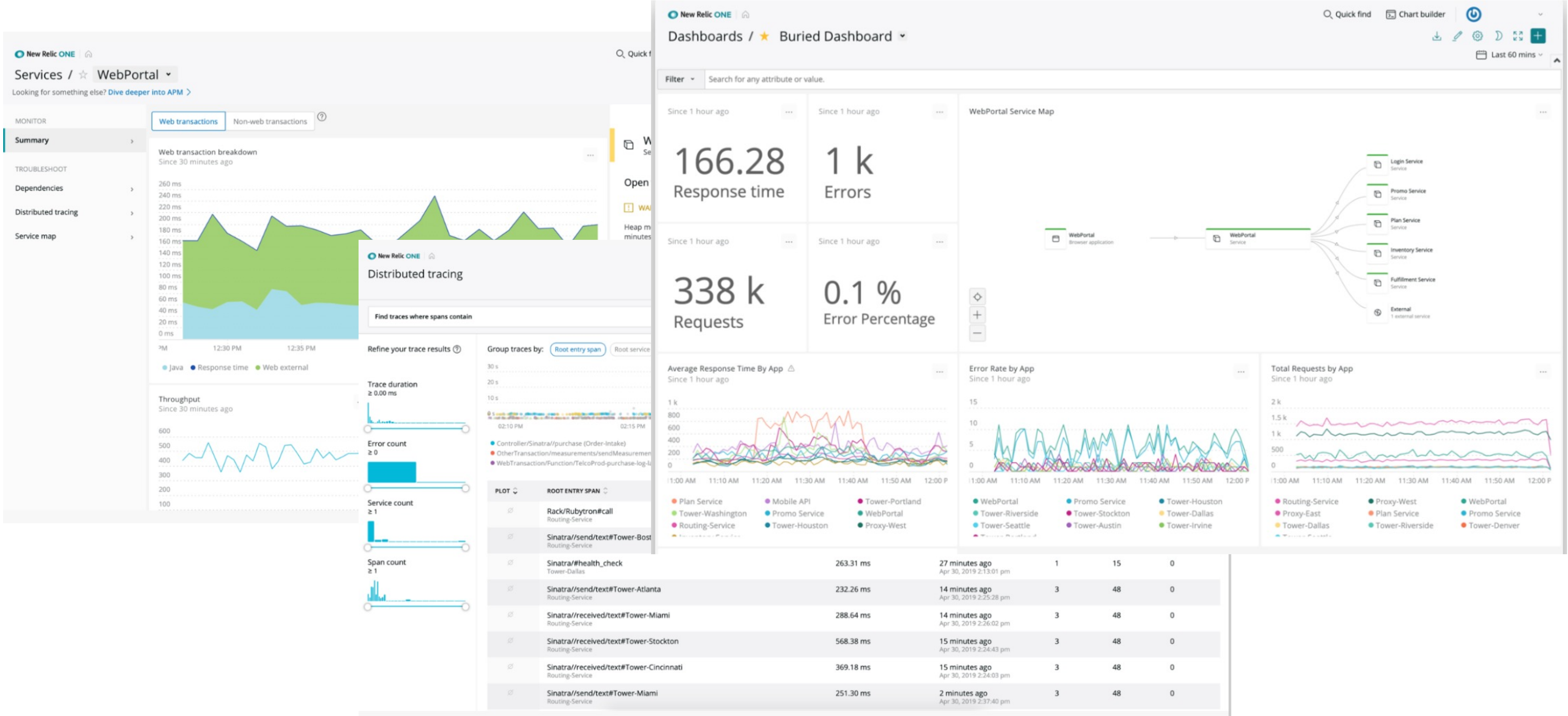
PAGEVIEWS/MIN (NOW) 1.02
1 HOUR AGO 1.05 ▼ 3.25 %

BOUNCE RATE (NOW)
1 HOUR AGO

Internal monitoring

- pre-SaaS/PaaS: *local*
  - Info collected & stored locally, e.g., Nagios
- Today: *hosted*
  - Info collected in your app but stored centrally
  - Info available even when app is down
- Example: Facebook ODS, New Relic
  - conveniently, has both a development mode and production mode

New Relic



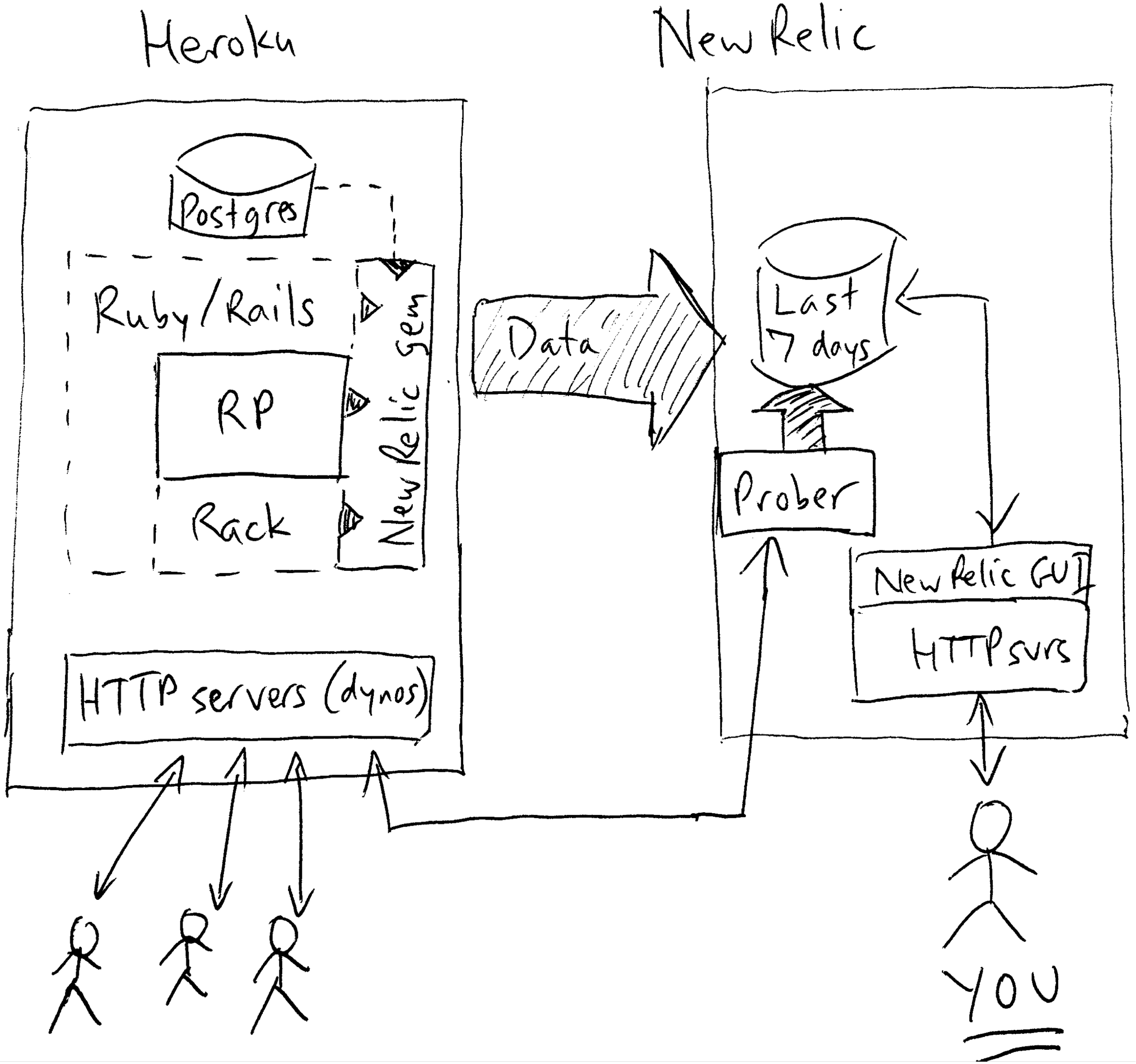
166.28 Response time

1 k Errors

338 k Requests

0.1 % Error Percentage

PLOT	ROOT ENTRY SPAN					
	Rack/Rubytron#call Routing-Service					
	Sinatra//send/text#Tower-Bost Routing-Service					
	Sinatra/#health_check Tower-Dallas	263.31 ms	27 minutes ago Apr 30, 2019 2:13:01 pm	1	15	0
	Sinatra//send/text#Tower-Atlanta Routing-Service	232.26 ms	14 minutes ago Apr 30, 2019 2:25:28 pm	3	48	0
	Sinatra//received/text#Tower-Miami Routing-Service	288.64 ms	14 minutes ago Apr 30, 2019 2:26:02 pm	3	48	0
	Sinatra//received/text#Tower-Stockton Routing-Service	568.38 ms	15 minutes ago Apr 30, 2019 2:24:43 pm	3	48	0
	Sinatra//received/text#Tower-Cincinnati Routing-Service	369.18 ms	15 minutes ago Apr 30, 2019 2:24:03 pm	3	48	0
	Sinatra//send/text#Tower-Miami Routing-Service	251.30 ms	2 minutes ago Apr 30, 2019 2:37:40 pm	3	48	0


Heroku
New Relic
Postgres
Ruby/Rails
RP
Rack
New Relic gem
HTTP servers (dynos)
Data
Last 7 days
Prober
New Relic GUI
HTTP svrs
YOU

Sampling of monitoring tools

What is monitored	Level	Example tool	Hosted
Availability	site	pingdom.com	Yes
Slow controller actions or DB queries	app	newrelic.com (also has dev mode)	Yes
Clicks, think times	app	Google Analytics	Yes
Process health & telemetry	process	monit, nagios	No

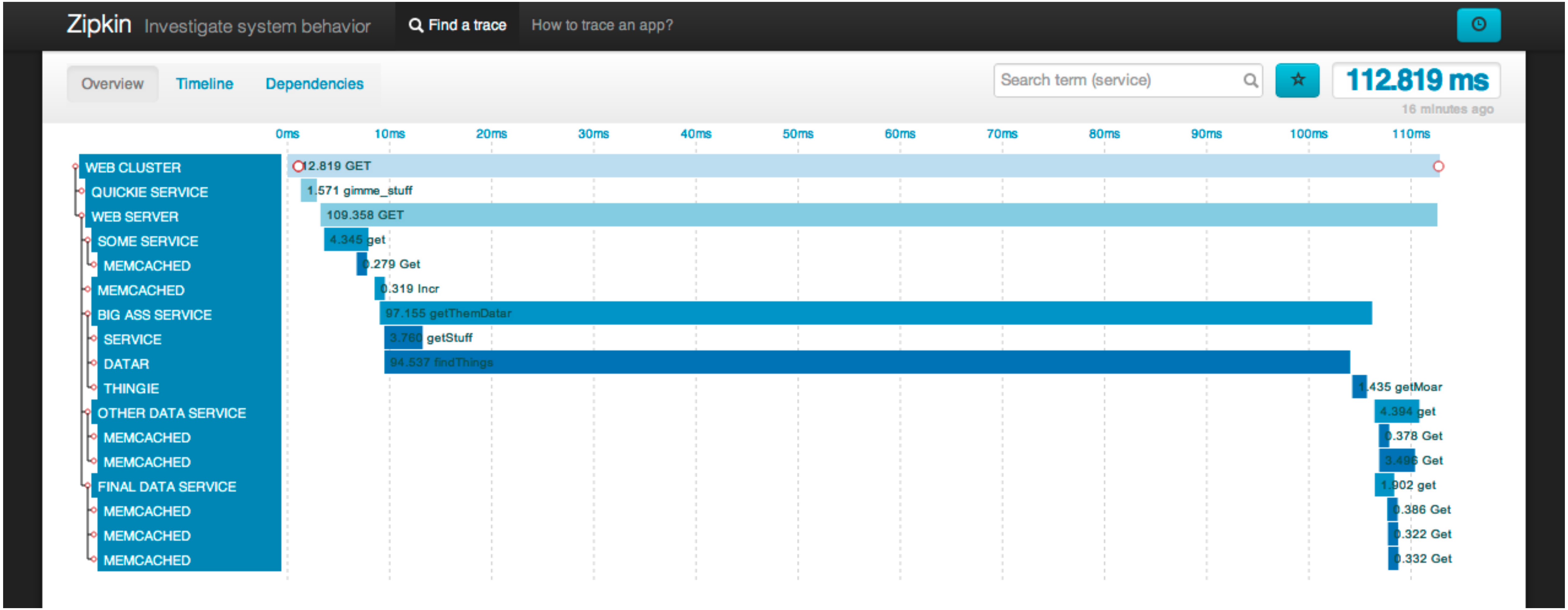
What to measure?

- *Stress testing* or *load testing*: how far can I push my system...
  - ...before performance becomes unacceptable?
  - ...before it gasps and dies?
- Usually, one component will be *bottleneck*
  - a particular view, action, query, …
- Load testers can be simple or sophisticated
  - request on a single URI over and over
  - do a fixed sequence of URI's over and over
  - play back a log file

Request Tracing

- Typically aggregating metrics such as latency over many requests
- A contrasting approach: request tracing
  - Simplified version of request tracing
    - Db query time, controller action time, rendering time
  - True request tracing is fine grained, following a request through every software component in every tier and timestamping it along the way
    - Google Dapper, Twitter Zipkin, Linkedin Htrace, …

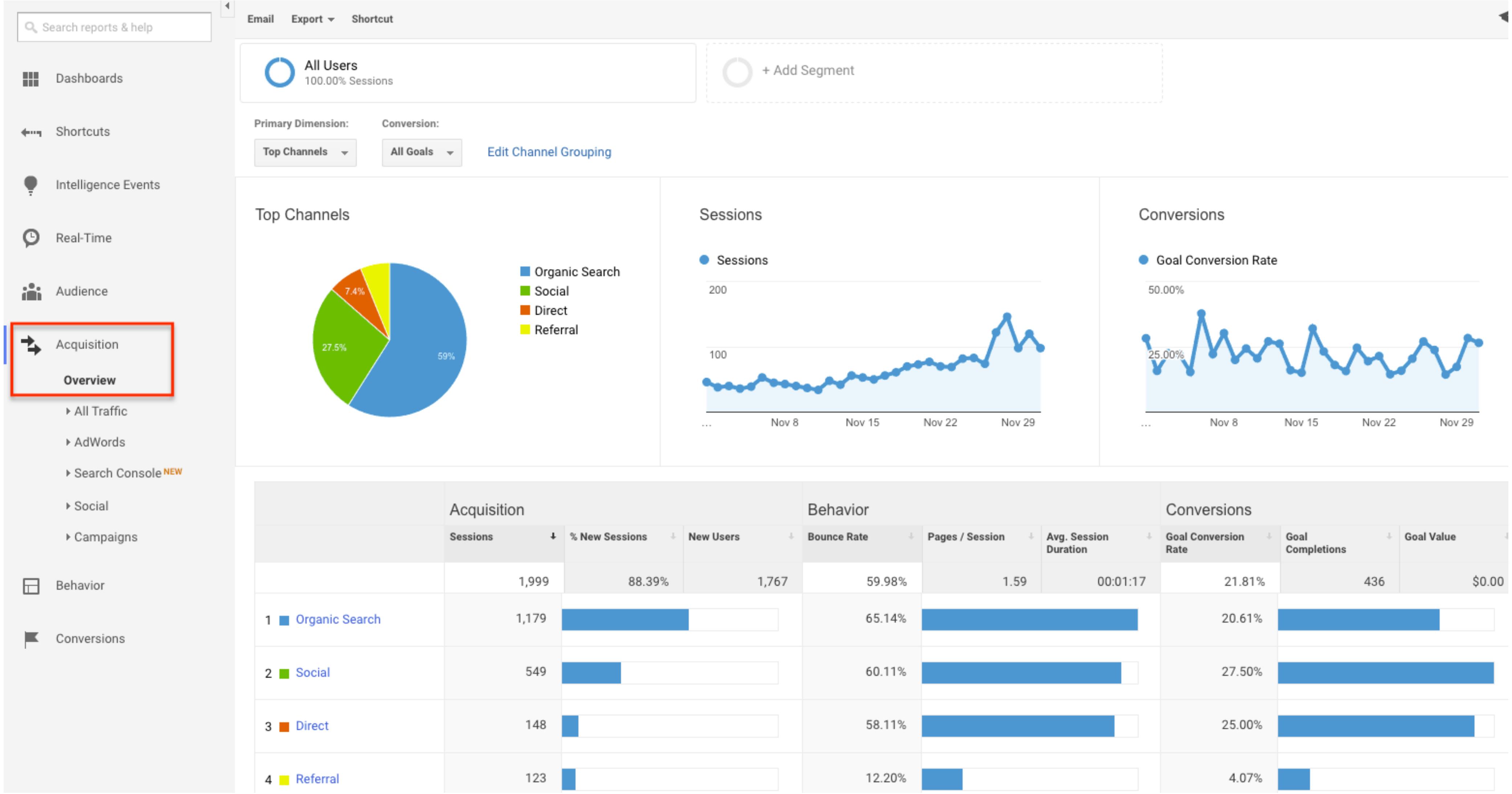
Example: Zipkin



WEB CLUSTER	12.819 GET
QUICKIE SERVICE	1.571 gimme_stuff
WEB SERVER	109.358 GET
SOME SERVICE	4.345 get
MEMCACHED	0.279 Get
MEMCACHED	0.319 Incr
BIG ASS SERVICE	97.155 getThemDatar
SERVICE	3.760 getStuff
DATAR	94.537 findThings
THINGIE	1.435 getMoar
OTHER DATA SERVICE	4.394 get
MEMCACHED	0.378 Get
MEMCACHED	3.496 Get
FINAL DATA SERVICE	1.902 get
MEMCACHED	0.386 Get
MEMCACHED	0.322 Get
MEMCACHED	0.332 Get

Monitoring for Understanding Customers’ Behavior

- Clickstreams: what are the most popular sequences of pages your users visit?
- Dwell times: how long does a typical user stay on a given page?
- Abandonment: if your site contains a flow that has a well-defined termination, such as making a sale, what percentage of users “abandon” the flow rather than completing it and how far do they get?
- E.g., Google analytics
- Advertising! – Most of Google’s Revenue



Primary Dimension: Top Channels

Conversion: All Goals

	Acquisition			Behavior			Conversions		
	Sessions	% New Sessions	New Users	Bounce Rate	Pages / Session	Avg. Session Duration	Goal Conversion Rate	Goal Completions	Goal Value
	1,999	88.39%	1,767	59.98%	1.59	00:01:17	21.81%	436	$0.00
1 🟦 Organic Search	1,179			65.14%			20.61%		
2 🟩 Social	549			60.11%			27.50%		
3 🟧 Direct	148			58.11%			25.00%		
4 🟨 Referral	123			12.20%			4.07%		