Google

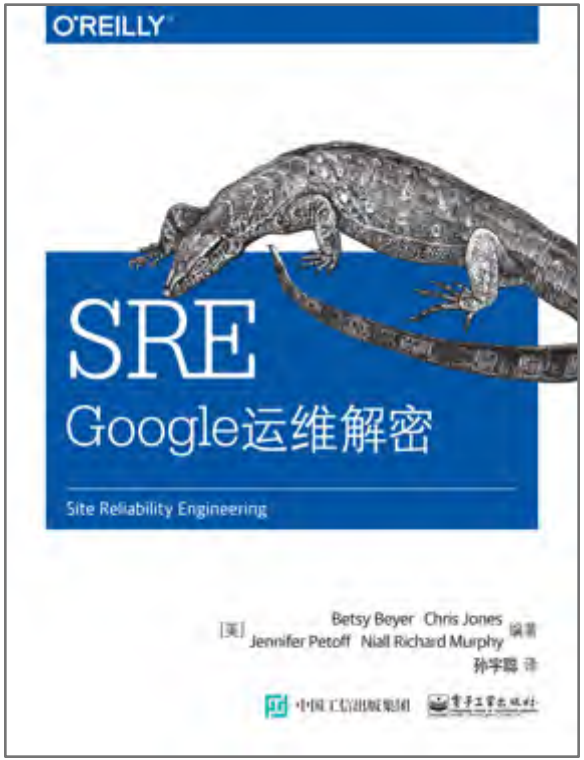
Service Levels and Error Budgets

GOPS, Shanghai
2016-09-23

Chris Jones

Site Reliability Engineering

- Published in April 2016
  - Now with Chinese translation!
- Talk is based on three chapters:
  - Service Level Objectives
  - Error Budgets
  - Introduction



Agenda

1. How good is a service?
2. How good should it be?
3. When do we need to make it better?
4. How do we make it better?

Agenda

1. **How good is a service?**
2. How good should it be?
3. When do we need to make it better?
4. How do we make it better?

Service Level Indicators (SLIs)

- An **indicator** (SLI) is a *quantitative measure* of how good some *attribute* of the service is.
- An *attribute* is a dimension the service's users care about, such as:
  - *throughput*, how much work the service can do
  - *latency*, how long the work takes
  - *availability*, how often the service can do work

Choosing Indicators

1. Figure out what service properties users care about
2. Collect data about those properties
    a. Start analyzing server logs
    b. Instrument the service and export metrics to monitoring
3. Choose *a few* metrics and carefully define indicators

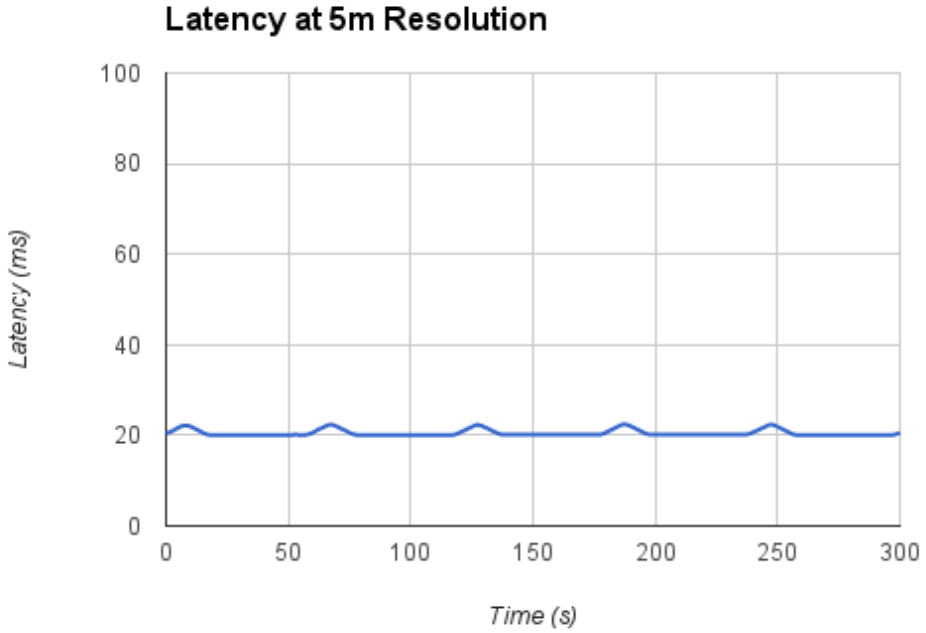
Defining Indicators

1. Start with a property: “latency”
2. Specify the property: “HTTP GET latency”
3. Choose how to measure it: “measured at the client”
4. Choose the universe of measurement: “all frontend servers”

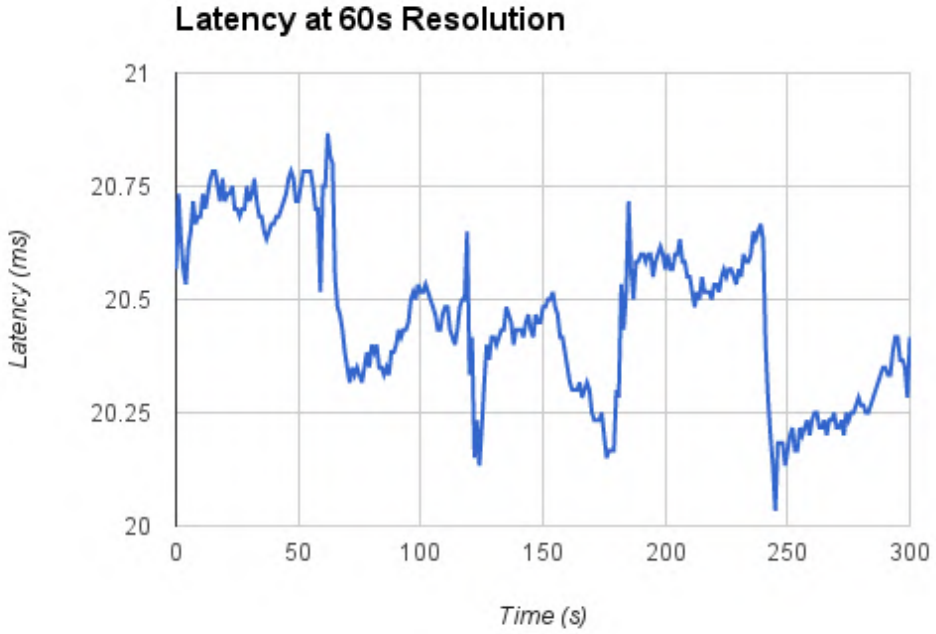
Result: “HTTP GET latency measured at the client every 10 seconds, by a black-box prober, across all frontend servers”

Shorthand: “client-side frontend GET latency”

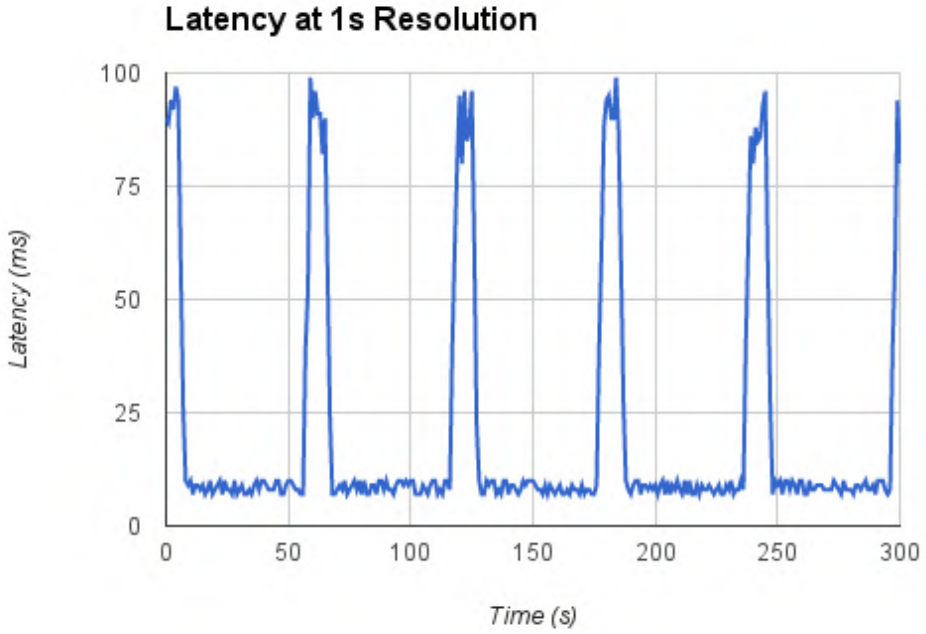
Understand Your Metric (1)



Understand Your Metric (1)

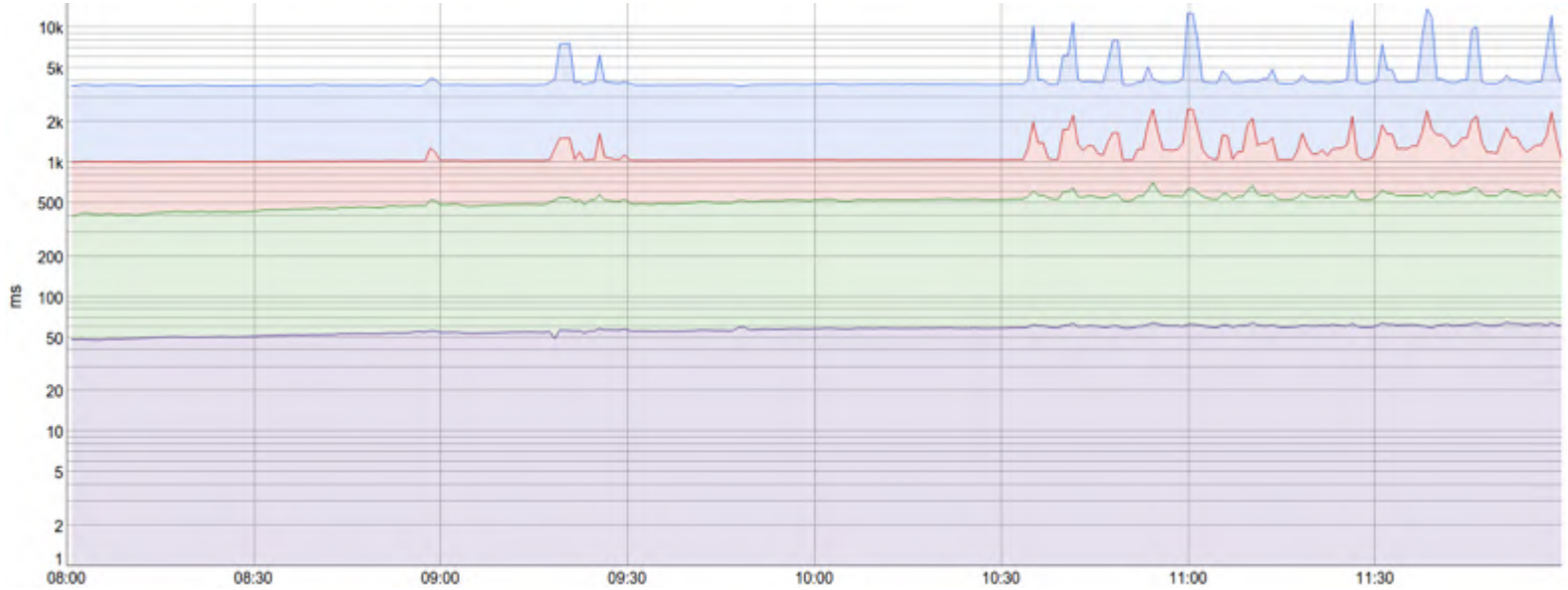


Understand Your Metric (1)



Understand Your Metric (2)

- Use histograms and percentiles instead of averages



Re-Defining Indicators

Previous Result: “HTTP GET latency measured at the client every 10 seconds, by a black-box prober, across all frontend servers”

New Result: “95th percentile of HTTP GET latency measured by a black-box prober every second, aggregated every 10 seconds, across all frontend servers”

Shorthand: “p95 client-side frontend GET latency”

Agenda

1. How good is a service?
2. **How good should it be?**
3. When do we need to make it better?
4. How do we make it better?

How Good Should A Service Be?

- How {fast, reliable, available, …} a service should be is fundamentally a *product* question
- “100% is the wrong reliability target for (nearly) everything”
  - cost of marginal improvements grows ~exponentially
- Can always make service better on *some* dimension, but involves tradeoffs with $, people, time, and other priorities
  - Product & dev management best placed for tradeoffs

Set Achievable Targets

- Understanding what users need from the service, product management, dev management, and SRE management jointly agree on targets
  - “p95 latency should be < 250 ms, 99.9% of the month”
  - “should respond to requests at least 99.9% of the time”
- Targets should be ambitious but *achievable*

Service Level Objectives

- An SLO is a mathematical relation like:
  - SLI ≤ target
  - lower bound ≤ SLI ≤ upper bound
- Publish SLO to users & try to be *slightly* better, just in case
  - defines what users can reasonably expect & design for
  - but don't be *too much* better or users will depend on it

Agenda

1. How good is a service?
2. How good should it be?
3. **When do we need to make it better?**
4. How do we make it better?

Error Budgets

- An SLO implies acceptable levels of errors
  - 99.9% availability $\Rightarrow$ 0.1% *un*availability
- Tolerable errors accommodate
  - rolling out new software versions (which might break)
  - releasing new features
  - inevitable failure in hardware, networks, etc.

Balance Reliability and Velocity

- Error budgets balance reliability with feature velocity
  - SRE's job is *not* "zero outages", default answer isn't "no"
  - instead, maximize velocity given reliability constraint
- Simple version:
  - release features until error budget exhausted, then
  - focus devs on reliability improvements until budget refill

Sophisticated Error Budgets

- change pace of feature releases given remaining budget
- keep a “rainy-day fund” for unexpected events
- use budget exhaustion rate to drive alerting
  - e.g. alert if recent errors > 1% of remaining budget
- small number of “silver bullets” for true emergency launches despite budget launch freeze

Agenda

1. How good is a service?
2. How good should it be?
3. When do we need to make it better?
4. **How do we make it better?**

What Should SREs Do?

- build monitoring systems to measure indicators
- provide input on feasibility of achieving targets
- work with devs to improve *both* reliability and velocity
  - standardize infrastructure
  - consulting on system design
  - build safe release & rollback systems

What Must SREs Have?

- SRE must have (and use) authority to halt launches which exceed error budget
  - requires strong support from management
- SRE must have ability to return pager to devs
  - strong SRE & dev partnership essential, no “code thrown over wall”
- SRE must have similar experience to devs

Things To Remember

1. Use SLIs to understand service's key metrics.
2. Use SLOs to specify how good the service needs to be.
3. Use error budgets to control release velocity.