Principled Performance Analytics
Narayan Desai, Brent Bryan
	sre.google • twitter.com/googlesre
Acknowledgements

Our team: Jeff Borwey, Tyler Sanderson, Jacob Freilinger (& Narayan and Brent)

Jez Humble, Adam Kramer, Christian Webb, Will Patterson, Chris DeForeest, Julius Plenz

Eric Brewer, Sam McVeety, Chris Heiser, Niall Murphy, Nicole Forsgren
Is it working?
Reliability

**Availability**
Is the service there when you need it?

**Performance**
How effectively is work performed?

**Correctness**
Does a service do what’s expected?
Whenever there’s a problem, just page me
SLOs

1. Encode system goals
2. Specify behavior expectations
3. Determine when to page
4. Bound emergency behavior
5. Enable error budgets
6. Indemnify for dependency problems
7. Coordinate priorities between teams
8. Estimate outage magnitude
9. Signal service maturity
10. Bound supported behavior
Reliability in Practice

**Availability**
- ✔ Count the number of failed requests
- ✗ 400s vs 500s
- ✗ Deadlines
- ✗ Malformed Requests
- ✗ Retries Magnify Errors

**Performance**
- ✔ Set P99 latency SLO
- ✔ Create Probers
- ✗ Workload dependent
- ✗ Probers are narrow

**Correctness**
- ✔ Integration Tests
- ✔ Golden Datasets
- ✗ Limited, non-adaptive coverage
- ✗ Hope is not a strategy
Yo Dawg, I heard you like SLOs
Errors are shallow data

All happy families are alike; each unhappy family is unhappy in its own way.

---

Leo Tolstoy
Anna Karenina

- SLOs require recognized errors
- Errors are ambiguous
- Bugs can result in over/undercounting
- Calibration errors result in over/undercounting
- Lots of room for problems
- No regular maintenance cycle
- Results in poor data products

---

Errors

Total
What now?
Reliability via Performance Analytics
Taking a Step Back

As a Customer:
- Is service meeting expectations?

As a Service Provider:
- Is the system working as it should?

Shared Concerns:
- Is it you, is it me, or is it both of us?
What do Service Providers See?

Workload performance ... across all customers

Complications:

- Mix shifts in workload
- Environmental factors like contention
- Mixed environments, job priorities, etc
What do customers see?

You may not know if a workload is performant but your customers do

Services should be consistent

```
SELECT img
FROM DogPics AS DP
LEFT JOIN FriendsFavs AS FF
  USING (img_id)
WHERE DP.cute = 'very'
  AND FF.stars >= 4
ORDER BY FF.favs DESC
LIMIT 1000
```
A High Level Model

Workload \rightarrow Performance \leftarrow Service

Reliability
No Consensus Elephant

Requirements
Performance expectations
Correctness
Client Metrics & Logs
Application source code

Service architecture
Infrastructure
Environment
Dependency graph
Internal Metrics & Logs
Service source code
Applying to the Model

Workload → Performance ↘ Reliability ← Service

Job Runtime

CPU Seconds
Applying to the Model

Workload → Performance ← Service

Reliability

Job Runtime

CPU Seconds

Google

Site Reliability Engineering
Steps to Solve Service Reliability:

1. Partition Workloads by Intent
2. Analyze Performance
3. Profit!
$2\sigma$ Technique
Hypothesis:
Self-Similar Workloads Should Have Consistent Performance

Technique Overview:
- Partition workloads into Cohorts ← Approximate Intent via Workload Features
- Build Performance Baselines ← Estimate Distributional Form (e.g. Normal)
- Estimate Likelihood of Delivered Performance ← Test For Stationary

Result:
- Set of Events with Predicted Likelihoods
- Time-series of summary statistics describing concentration of extreme outliers
Assume:

- Metric distributions can be approximated by Normal distribution
- Modeling errors excluded via baseline qualification

Then:

- Workload z-scores are a proxy for likelihood
- Workload performance should be IID
- Z-scores follow a standard Normal distribution
- Baseline distribution computation is “embarrassingly parallelizable”
- Z-scores are combinable (across cohorts!)
Mechanics

Strategy:

- Aggregate z-scores across workloads
- Monitor fraction of workloads with z-scores $\geq 2$, in windows
- Expect 2-5% $2\sigma$ outliers in any given window
- When >10% of workloads are $>2\sigma$, BE AFRAID.

Detection is based on fraction of workloads exhibiting regression
Leveraging Structure: $2\sigma$ Technique

**“Model”**

- Historical Service Data
- Partition into Cohorts
- Compute Baselines

Cohort Metrics
Leveraging Structure: $2\sigma$ Technique

“Model”

Historical Service Data ➔ Partition into Cohorts ➔ Compute Baselines

“Measure”

Current Service Data ➔ Compute Z-Scores ➔ Monitor Z-Scores
Leveraging Structure: 2σ Technique

“Model”

Historical Service Data → Partition into Cohorts → Compute Baselines → Cohort Metrics

“Measure”

Current Service Data → Compute Z-Scores → Monitor Z-Scores
Leveraging Structure: 2σ Technique

“Model”

Historical Service Data → Partition into Cohorts → Compute Baselines

“Measure”

Current Service Data → Compute Z-Scores → Monitor Z-Scores
Frequently Asked Questions

- Do performance metrics actually follow Normal distributions?
- How do you know if approximations hold?
- How do you define cohorts?
- How do deal with “singleton” / infrequent workloads?
- Aren’t there a lot of singleton workloads?
- Ok, but does this really work?
Backtesting

Lowest availability experienced by X% of the highest availability projects per Location

% of baselined requests with latency >2τ by cell (choose region above)
Applications
Sensitive Detection of Service Problems

![Chart: Percent Slow Queries Total Time]

18 hours
Streamlined Diagnosis

- Total Time
  - Queue Time
  - Execution Time
  - I/O Time

Total Time

I/O Time

Percent of Slow Queries By Reason

Google

Site Reliability Engineering
Excursion Impact Assessment

![Excursion Impact Chart]

- Percent Slow Queries Total Time

Excursion Impact

*Graph showing the increase in percent slow queries total time from early April to late April.*
Measuring unexpected correlations
Approximate Cohort A/B Testing
Conclusions
Key Observations

- Reliability is a shared property (between customer & service)
  - Reconstruction of end to end behavior is critical
- Variability is what customers actually care about
- Distributed systems often produce decorrelation
  - We can measure it, and its absence
- Workload correlation can identify proximate causes
- Metric combinability is critical for analysis
- Error recognition is a gestalt of human judgements over time
- Due to the unrecognized problems in error recognition, SLOs aren’t feasible
Contributions

2σ is a method that:
- Incorporates user intent in order to model expected performance
- Tests an IID hypothesis to infer when systems diverge from expected behavior
- To produce data products that are comparable and combinable

We use these data products in order to:
- Perform change point detection when systems diverge from expectations
- Estimate the duration, severity, and specific impact of these excursions
- Localize subsystem performance problems
- Compare relative and absolute performance over time and arbitrary workload dimensions
- Directly measure correlation across subsystems and isolation domains

Resulting in:
- Calibration-free insights that characterize the consistency of a system
- The ability to test system invariants continuously
- Data building blocks that can be reprocessed to answer many questions
Closing Thoughts

- We can do a lot better than SLOs, and we must
- Performance data >> Availability data
- We need more models
- We need help!
  - (and have openings, talk to me or Brent)
Questions