PACIFIC NW SOFTWARE QUALITY CONFERENCE

Automated Detection of Individual Anomalies and General Behavioral Changes in Time Series Metrics

Tifany Yung, Renato Martins

October 2018

Agenda

- Introduction
- Challenges
- Existing methods
- Solution
- Algorithm
- Example usage
- Results
- References

Introduction

- Hundreds of metrics to monitor during deployment process
  - System health, performance, stability, etc.
  - Testing -> staging -> production environments
- Detect issues and regressions…
  - Before they affect end users
  - That slipped into production anyways
- Drawbacks of manual analysis by human analysts:
  - Time-consuming
  - Subjective, error-prone
- We would like a solution that allows us to address the drawbacks, in addition to reducing the time spent on analysis

Challenges

- Some automation is possible using automatic alerts triggered by passing a threshold
  - Takes time to reach threshold
  - May never reach critical levels, moderate changes will slip through
  - Still need to inspect metrics, whether manually or automatically, for deviant behavior within the thresholds
- Would like to know not just whether a metric began to deviate after a deployment, but also the times at which it happened
  - Helps to rule out noise from dependencies, outside factors, etc. and ensure that the change is due to the deployment

Existing methods

- Look for individual anomalies within a data set
  - Flag a point that differs from its neighbors by more than the threshold
- Look for general changes in behavior, i.e. compare metric data from before deployment (model data) to that of after deployment (test data)
  - Encode model and test data as strings, generate detector strings that do not match model data
  - Encode model and test data as strings, compute edit distance between them and flag the test data if distance exceeds a threshold

Solution

- Automated detection algorithm ingests a metric's model and test data
- Identifies individual points within the test data that are anomalous w.r.t. the model data
- Based on the number and distribution of the individual anomalies, judges whether the test data overall deviates from the behavior seen in the model data
  - If *most* test points are deviant and are *more recent* in the test data set, then the metric is flagged
  - How *most* and *more recent* are defined will be covered in detail later
- Will keep implementation details to a minimum to focus on the algorithm itself

Algorithm: Initial setup

- One-time setup to create a configuration file for each set of metrics to be monitored
- Configuration file describes parameters for each metric
  - Time ranges for model and test data
  - Season size, if any
  - More parameters to be described in the algorithm

Algorithm: Modeling expected behavior

- Retrieve the model data as a timeseries, i.e. a list of data points with corresponding timestamps
- Remove outliers from the model data so that they do not affect the generated model
  - Identify outliers via the interquartile range method (IQR)
  - Outliers defined as points below *p25 - n*IQR* or above *p75 + n*IQR*
    - Where p25 and p75 are the 25th and 75th percentiles, respectively
    - Common value of *n* for normally-distributed data is 1.5, but can be set in config file to allow for more/less sensitive alerts
    - Three-sigma rule not used since timeseries data is frequently not normally distributed

Algorithm: Modeling expected behavior (cont.)

- Generate ARIMA model of the data
  - If metric is marked as seasonal in the config file, first generate a new timeseries of the seasonal difference and use it instead
  - Python statsmodels package to automate selecting model parameters and fitting model coefficients
- Compute errors between actual model data and model predictions
  - Define *model error* as (*actual – predicted*)
  - Define *model error distribution* as the distribution of all of the model errors
  - Model error distribution will be used to determine how much error is expected for this model

Algorithm: Identifying anomalous test points

- Retrieve the test data as a timeseries
- Use the generated ARIMA model to predict values for the test points
- Compute prediction errors for the test points
- Check if each point's prediction error is an outlier w.r.t. the model error distribution
  - Use the same IQR method as when removing outliers from the model data
  - If a test point's prediction error is an outlier, the actual value deviated from the prediction more than expected
  - A test point whose prediction error is an outlier is flagged as an *individual anomaly*

Algorithm: Identifying overall behavioral changes

- If most of the test points are not anomalous, then the metric's test data is **not deviant** and no alert will be generated for it
  - Default definition of *most* is >=50%, but this can be changed using the config file
- Else, if *most* of the anomalous points in the more recent part of the test data, then the metric's *general behavior* is considered to have changed and is flagged as **deviant**
  - Default definition of *more recent* is the later 50% of the test data, but this can also be changed in the config file
- Email alert is sent, detailing which metrics deviated and in what direction

Algorithm: Making a call on a release

- Human reviews the metrics flagged in the alert
- Determine if the changes in the metrics were expected/acceptable, for example:
  - Increase in latency after the addition of a computationally heavy new feature could be expected
  - Decrease in latency could be an improvement
  - But if latency decreased while error rates increased, the release could have introduced a bug and the system could just be erroring out faster

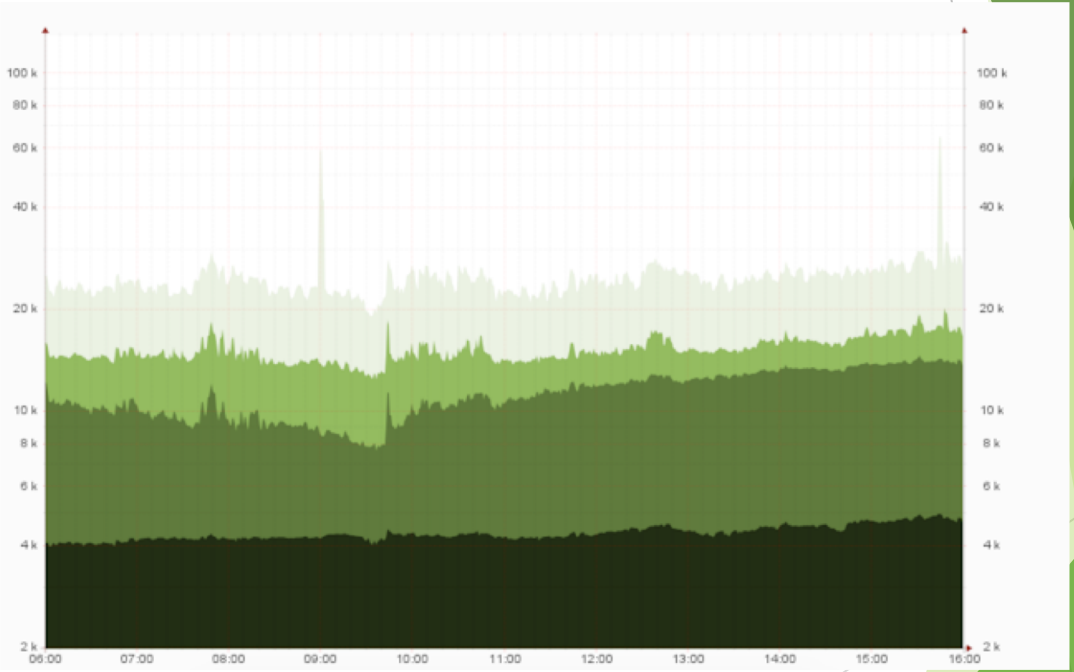
Example usage

- Model data:

  6:00 - 9:00
- Test Data:

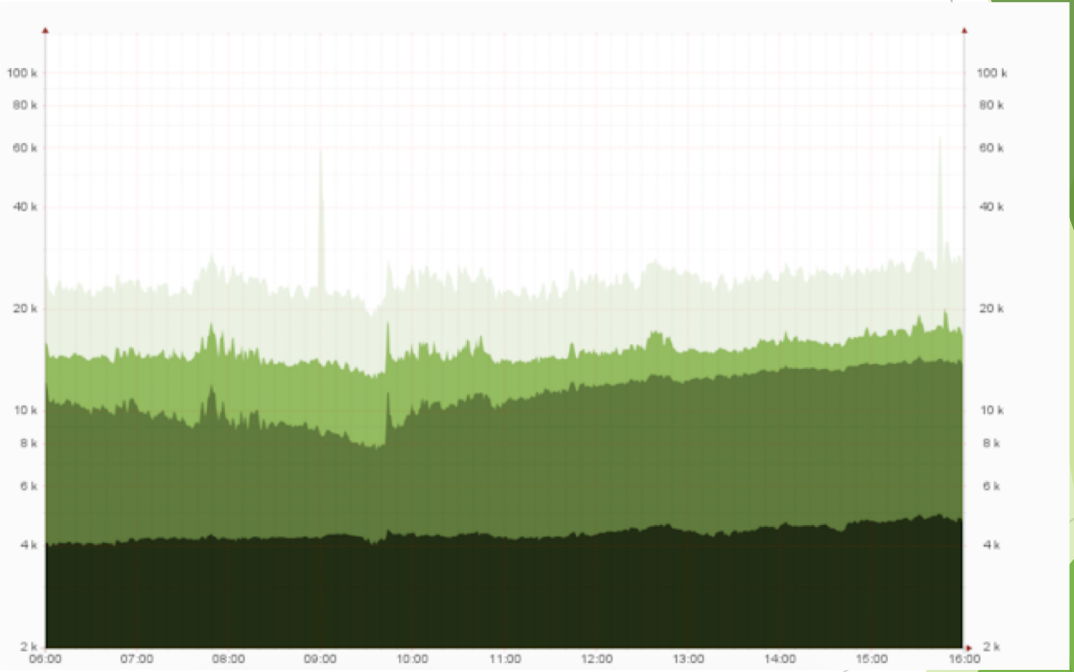
  15:00 - 16:00

- P50 (black)
- P95 (dark green)
- P99 (green)
- Max (pale green)



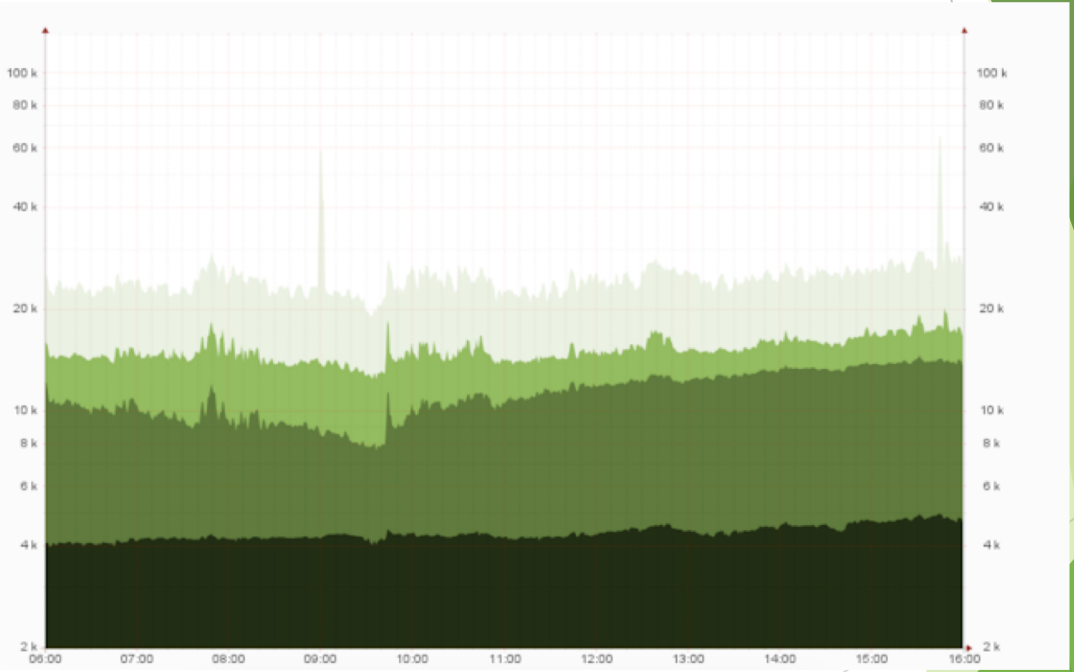
Example usage (cont.)

- Visible changes:
  - Large increase in P95
  - Possible increases in P99 and max
  - Increase in median??



Example usage (cont.)

- Changes found by the algorithm:
  - Actually found increases in all four metrics



Results

- Compared performance of algorithm against manual analysts
- **Increased rate of detection** of anomalous points and behavioral changes
  - Better detection of small changes that are not obvious when ”eyeballing” a graph
  - E.g. The P50 example metric
- **Reduced number of metrics to review** by 92-98% -> **reduced time spent on analysis**
- Remaining metrics require minimal human analysis
  - Already judged deviant by the algorithm, just confirm if the change was expected

References

Tifany Yung:
Email – tiyung@groupon.com
LinkedIn – www.linkedin.com/in/tifany-yung-99b47291

Renato Martins:
Email – rmartins@groupon.com