Using Statistical Inference for Capacity Planning
Dongfang Xu
Site Reliability Engineer
at Splunk Inc.
This presentation may contain forward-looking statements regarding future events, plans or the expected financial performance of our company, including our expectations regarding our products, technology, strategy, customers, markets, acquisitions and investments. These statements reflect management's current expectations, estimates and assumptions based on the information currently available to us. These forward-looking statements are not guarantees of future performance and involve significant risks, uncertainties and other factors that may cause our actual results, performance or achievements to be materially different from results, performance or achievements expressed or implied by the forward-looking statements contained in this presentation.

A discussion of factors that may affect future results is contained in our most recent annual report on Form 10-K and subsequent quarterly reports on Form 10-Q, copies of which may be obtained by visiting the Splunk Investor Relations website at www.investors.splunk.com or the SEC’s website at www.sec.gov, including descriptions of the risk factors that may impact us and the forward-looking statements made in this presentation. The forward-looking statements made in this presentation are made as of the time and date of this presentation. If reviewed after the initial presentation, even if made available by us, on our website or otherwise, it may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statement based on new information, future events or otherwise, except as required by applicable law.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. We undertake no obligation either to develop the features or functionalities described or to include any such feature or functionality in a future release.
Agenda:

- Why capacity planning?
- What is a realistic goal?
- Linear regression magic
- Recap
Why Capacity Planning?

- Avoid possible stability and reliability issues
- Elevate performance, identify the bottleneck
- Better cost model and saving money, while still meeting the needs of the business
- To support SLA that you want to achieve
Simple but Hard

Q: What is the maximum capacity your current system can handle under your SLO?

Throughput: QPS / TPS

SLO:
- Latency (Mean, P95, P99)
- Success/Error Rate (%)

Resource: under its safe threshold
- CPU
- Memory
- Disk
- Network
How to do a proper (.* )Test?

Performance Test/Load Test/Stress Test

Preparation:
- Identify the product use case
- Identify the upstream/downstream dependency
- Identify the core data pipeline
- Identify the base traffic flow & pattern
- Identify the peak traffic flow & pattern
- Tooling (Load generator/etc)

Model:
- Identify the highly-correlated resource (CPU/Mem/Heap) for the service
- Set your target threshold
  - ie, cpu (50%, 75%, 95%)
- Setup dashboard and alerts to avoid issue

Test Scenarios:
- Organic Growth
- Durability at high Throughput
- Pulse Throughput with interval
Peek at the Test Result

Environment:
Prod

Testing Window:
4 hour

Testing Load:
20mil AMTS(Active Metric Time Series), 80k mts/min

Screenshot services:
2 stateless - Time series creation
1 stateful - metadata storage

Results:
CPU Util (2% - 90%)
Latency: < 200ms
*R-Squared: > 95%

R-squared is the percentage of the dependent variable variation that a linear model explains. Usually, the larger the R^2, the better the regression model fits your observations.
Linear regression saves the day!
No magic, but statistics

#Single Linear Regression - (Req VS CPU) \( Y = ax + b \)

Request = data('tscreationservice.createManyTimeSeries.work', filter='clientType: sbingest', rollup='rate').sum(by=['clientType', 'tscreation_name']).sum()
cpu_p95 = data('jvm.cpu.load', filter='sfx_service: tscreation*').percentile(pct=95)

from signalfx.stats.linear_model import regression
solution1 = regression.fit([Request], cpu_p95, window=duration('5m'), fit_intercept=True)

coefficient = solution1['coef'][0]
intercept_single = solution1['intercept']
std_err = solution1['std_err']
R_square = solution1['R2']

Model1 = ((Request * coefficient) + intercept_single).publish(label='Model1_cpu')

#Multiple Linear regression - (Read, Write VS CPU) \( Y = a * x1 + b * x2 + c \)

cass_read_cluster = data('counter.cassandra.client.read-latency.count', filter='sfx_cluster: cassandra-metabase', rollup='rate').sum()
cass_write_cluster = data('counter.cassandra.client.write-latency.count', filter='sfx_cluster: cassandra-metabase', rollup='rate').sum()
cass_cpu_p95 = data('cpu.utilization', filter='sfx_service: cassandra-metabase').percentile(pct=95)

from signalfx.stats.linear_model import regression
solution2 = regression.fit([cass_read_cluster, cass_write_cluster], cass_cpu_p95, window=duration('5m'), fit_intercept=True)

coefficient1 = solution2['coef'][0]
coefficient2 = solution2['coef'][1]
intercept_multi = solution2['intercept']
std_err = solution2['std_err']
R_square = solution2['R2']

Model2 = (cass_read_cluster * coefficient1) + (cass_write_cluster * coefficient2) + intercept_multi).publish(label='Model2_cpu')
Model example

Model: \( Y = a \times X + b \)

CPU \sim\text{coefficient} \times \text{request} + \text{intercept}
Is it a mighty model?
Yes & No

Yes. If your loadtest covered the resource threshold range you want to model on.
Yeah!!!

No, If you are trying to predict the utilization range not covered by the test.
This is not too bad, but your final results might be vary a lot, and do not be surprised if the deviation > 20%.
Projecting CPU
150k(real) VS 150k mtsm (projected without test covered)

<table>
<thead>
<tr>
<th>Request</th>
<th>CPU Util</th>
<th>Coefficient</th>
<th>Intercept</th>
<th>std_err(%)</th>
<th>R^2(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projected</td>
<td>66307</td>
<td>51.7339647</td>
<td>0.0007</td>
<td>7.3301</td>
<td>0.82</td>
</tr>
<tr>
<td>Real</td>
<td>66307</td>
<td>21.7895336</td>
<td>0.0002</td>
<td>7.4926</td>
<td>0.98</td>
</tr>
</tbody>
</table>
Other limits of a model

Yes & No

Yes, If you are trying to predict capacity on different hardware type (even same cores/mem/etc)
Of course, Diff AWS hardware type have different CPU freq, different architectures.

Yes, If service config/code updated, kernel upgraded, etc any underlying system change.
The butterfly Effect applies there, any change in the underlying system will produce different results to your model!

... Don't be too greedy!

System is not follow the same linear under different load, make sure your test covered enough range that you need. Do not try to solve your capacity problem using your “perfect” model.
Recap
Last but not least

- A regular loadtest is the key to keep your model updated, might be a “perfect” model.
- It is much easier to perform regression on dashboards and figure out key relationships.
- Production is the ideal place to do the test. However, be careful and be humble!
- Always keep redundancy to provisioning, No test data can compare to real customer traffic.
Please meet me in the chat lounge for the Q&A
Than you !!

Email: dfxu.james@gmail.com
LinkedIn: https://www.linkedin.com/in/dongfang-xu-36993553