Making sense of hierarchical log data

Damon Wischik
Dept. of Computer Science and Technology
Debugging in a data center

Query RecentPosts
in parallel,
query AdServer

AdServer:
query FriendGraph

AdServer:
JOIN(FriendGraph)

JOIN(RecentPosts, AdServer)

transaction end

time

transaction start

Query RecentPosts

query AdServer

AdServer:
query FriendGraph

AdServer:
JOIN(FriendGraph)

JOIN(RecentPosts, AdServer)

transaction end

time

transaction start

Query RecentPosts

query AdServer

AdServer:
query FriendGraph

AdServer:
JOIN(FriendGraph)

JOIN(RecentPosts, AdServer)

transaction end

time

transaction start

Query RecentPosts

query AdServer

AdServer:
query FriendGraph

AdServer:
JOIN(FriendGraph)

JOIN(RecentPosts, AdServer)

transaction end

time

transaction start

Query RecentPosts

query AdServer

AdServer:
query FriendGraph

AdServer:
JOIN(FriendGraph)

JOIN(RecentPosts, AdServer)

transaction end

time

transaction start

Query RecentPosts

query AdServer

AdServer:
query FriendGraph

AdServer:
JOIN(FriendGraph)

JOIN(RecentPosts, AdServer)

transaction end

time

transaction start

Query RecentPosts

query AdServer

AdServer:
query FriendGraph

AdServer:
JOIN(FriendGraph)

JOIN(RecentPosts, AdServer)

transaction end

time
DevOps notices problems with response time.

- Was there a network update, which messed things up?
- Was there a code update?
- Is there some part of the system that started churning?
- Is it because of changes in demand?
- Is it a knock-on effect from elsewhere?
Debugging in a data center

True story:
- Student’s peer-to-peer code runs happily, until it suddenly becomes horribly slow
- It’s only slow on the campus network, not over ADSL
- Diagnosis:
  Certain workload patterns trigger a cascade of \( \approx 12 \) packets
  This overwhelms the switch buffers, and the final few packets get dropped
  The code goes into timeout / recovery

Network engineer view
Programmer view
DevOps view
Challenge, version 1

If we have logs of flows, plus the process ID that each flow belongs to, plus occasional programmer-generated log messages, but we don’t know the network state or the code, how can we diagnose problems?
State of the art: AppDynamics

- The operator manually assigns labels: to transaction types, and to classes of server
- AppDynamics reports average statistics grouped by these labels
Based on logs at an intermediate level of abstraction, we want unsupervised learning:

- of clusters of code structure
- of how code behaviour is influenced by network infrastructure
- of how the network infrastructure responds to code choices

To give the operator insight into what’s happening and why.

First, write the transaction as a sentence in a grammar of state transitions and actions.

(Here I’m restricting attention to trees, encoded via depth-first traversal. All the bells and whistles, e.g. interaction with infrastructure, parallel calls, and programmer log messages, can be added.)

\[ S_0 = 0 \]
\[ u_0 = \"fork[host5]\" \]
\[ S_1 = f(S_0) \]
\[ u_1 = \"join\" \]
\[ S_2 = j(S_0,S_1) \]
\[ u_2 = \"fork[host8]\" \]
\[ S_3 = f(S_2) \]
\[ u_3 = \"fork[host3]\" \]
\[ S_4 = f(S_3) \]
\[ u_4 = \"join\" \]
\[ S_5 = j(S_3,S_4) \]
\[ u_5 = \"join\" \]
\[ S_6 = j(S_2,S_5) \]
\[ u_6 = \"join\" \]
Second, train an autoencoder: a neural network that learns to reconstruct the sequence.


$$s_0, s'_0$$  
high-dimensional vectors

$$Z$$  
low-dimensional vector: the latent representation

$$u_i$$  
actions, embedded into vectors

$$p_i$$  
vectors that encode distributions over actions

$$f, j, f', j', e, d, r$$  
functions parameterized by weights, trained to minimize

$$\text{loss} = \sum_{\text{transactions } t} \sum_{\text{steps } i \text{ in } t} \log p_i(u_i) + \text{regularizer(distribution of } Z)$$
The latent variable $Z$ encapsulates

- **intent:**
  how a transaction’s code is split into subrequests

- **experience:**
  how the next choice is affected by the experiences of subrequests

It is an unsupervised way to group similar transactions. It should be a good basis for interactive visualization / investigation.
This type of data arises in many societal networks.

- **Public transport user**: commute
  - bus leg
  - train leg
  - log records of tap-in tap-out
  - train 1
  - train 2
  - underlying public transit infrastructure

- **Taxi driver**: shift
  - ride 1
  - ride 2
  - GPS
  - GPS
  - GPS
  - GPS
  - underlying road infrastructure

- **Log records of each ride’s start and end**
Log files are everywhere, but they’re hard to make sense of.

Analyzing Windows telemetry, to identify common call stacks at the time of crashes.

Debugging in the (very) large: ten years of implementation and experience
Glerum, Kinshumann, et al. (SOSP 2009)

ReBucket: a method for clustering duplicate crash reports based on call stack similarity
Dang, Wu, et al. (ICSE 2012)

Mining event logs of processes (businesses, hospitals, home sensors) to discover dependencies and to highlight common subprocesses.

Structure identification in layered precedence networks
Kong, Katselis, Beck, Srikant (CCTA 2017)

Mining context-dependent and interactive business process maps using execution patterns
Li, Bose, van der Aalst (Business Process Management 2010)
All sorts of structured human activity fall in this general category (and data centers are the perfect laboratory, because of reproducibility and privacy)

**Challenge v3**

Based on log records at a low level of abstraction, infer the latent hierarchical structure of the activity.

(Deep learning for Natural Language Processing manages to learn something like grammar. Log records are surely easier!)
Challenge v4
Build systems for working with this sort of data.

- Build databases that make it easier / faster to manipulate hierarchical data
  
  *FDB: A query engine for factorised relational databases*
  Bakibayev, Olteanu, Závodny (Proc VLDB 2012)

- Excel, Tableau, etc. are wedded to flat tabular data. Invent tools for users to interact with hierarchical data.

  Example interaction:
  “Label some GPS coordinates, use these to up-label the trips, use these to down-label all their GPS coordinates.”
Challenge v4
Build systems for working with this sort of data.

- The Synecdoche Engine
  To understand data about richly structured behaviour, it’s often helpful to look at illuminating anecdotes.