Intelligent Cloud Operations
Part 5. Distributed Trace Analysis

Definition (Gartner) [**AIOps**]

AIOps platforms utilize big data, modern machine learning and other advanced analytics technologies to directly and indirectly enhance IT operations (monitoring, automation and service desk) functions with proactive, personal and dynamic insight.

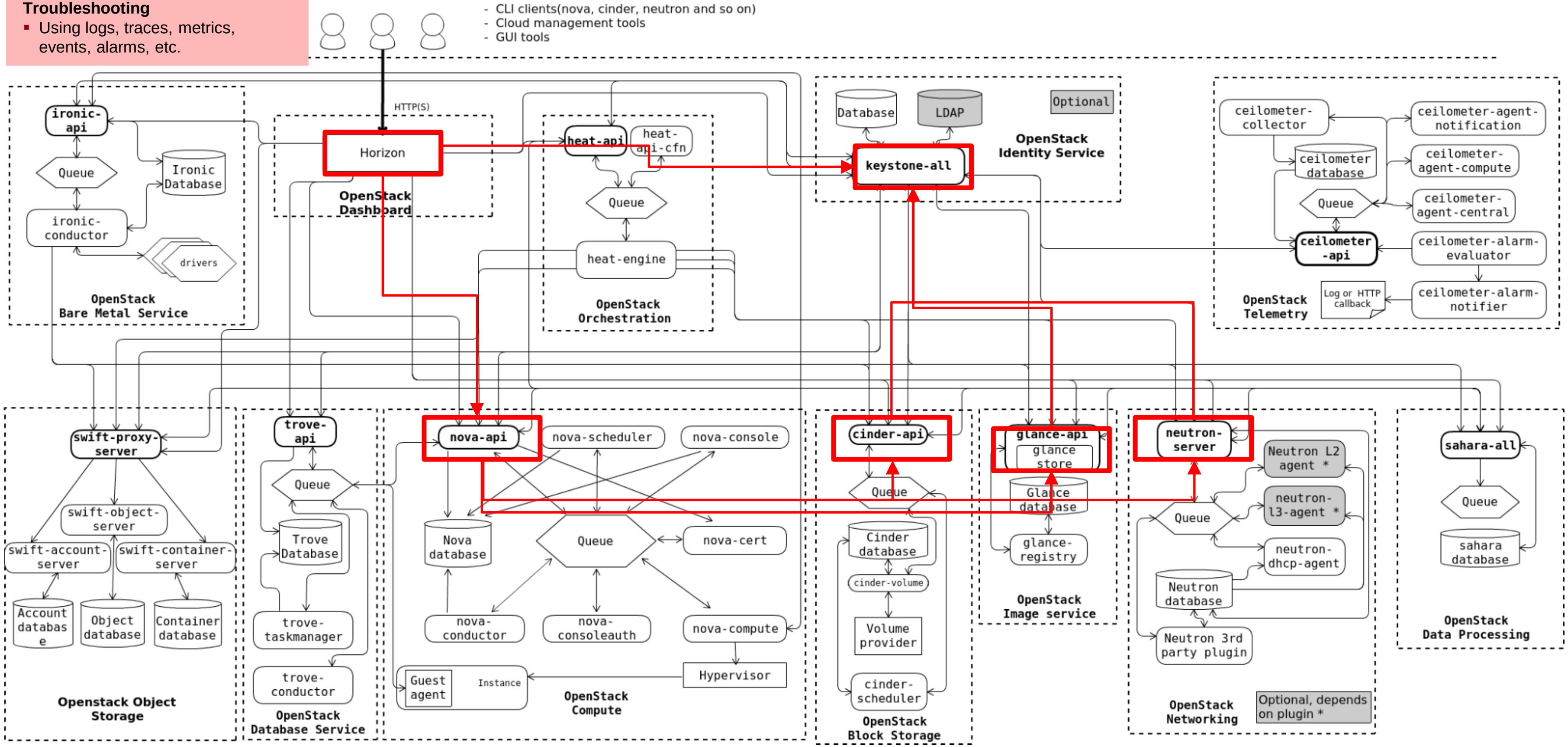
Prof. Jorge Cardoso
E-mail: jorge.cardoso@huawei.com
Intelligent Cloud Operations/SRE Dept.
Ireland and Munich Research Centers

2020

OpenStack Troubleshooting

Troubleshooting
- Using logs, traces, metrics, events, alarms, etc.

- CLI clients(nova, cinder, neutron and so on)
- Cloud management tools
- GUI tools



https://docs.openstack.org/install-guide/get-started-logical-architecture.html

Troubleshooting
Monitoring Data Sources

System's Components (e.g., OBS, EVS, VPC, ECS) are monitored and generate various types of data: Logs, Metrics, Traces, Events, Topologies

Logs. Service, microservices, and applications generate logs, composed of timestamped records with a structure and free-form text, which are stored in system files.

2017-01-18 15:54:00.467 32552 ERROR oslo_messaging.rpc.server [req-c0b38ace - default default] Exception during message handling

Metrics. Examples of metrics include CPU load, memory available, and the response time of a HTTP request.

{"tags": ["mem", "192.196.0.2", "AZ01"], "data": [2483, 2669, 2576, 2560, 2549, 2506, 2480, 2565, 3140, …, 2542, 2636, 2638, 2538, 2521, 2614, 2514, 2574, 2519]}

Traces. Traces records the workflow and tasks executed in response to, e.g., an HTTP request.

{"traceId": "72c53", "name": "get", "timestamp": 1529029301238, "id": "df332", "duration": 124957, "annotations": [{"key": "http.status_code", "value": "200"}, {"key": "http.url", "value": "https://v2/e5/servers/detail?limit=200"}, {"key": "protocol", "value": "HTTP"}, "endpoint": {"serviceName": "hss", "ipv4": "126.75.191.253"}]

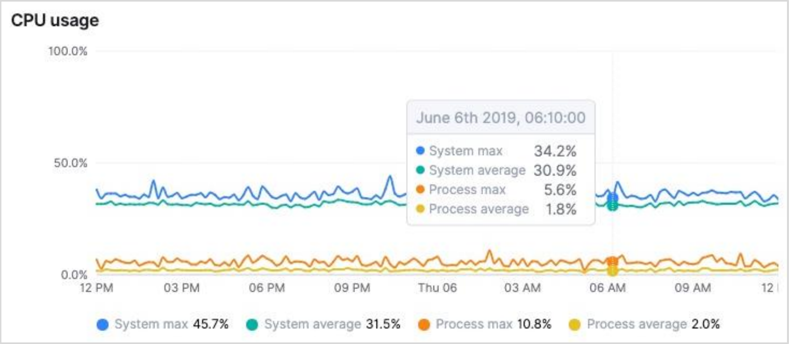
Events. Major milestones which occur within a data center can be exposed as events. Examples include alarms, service upgrades, and software releases.

{"id": "dns_address_match", "timestamp": 1529029301238, …}
{"id": "ping_packet_loss", "timestamp": 152902933452, …}
{"id": "tcp_connection_time", "timestamp": 15290294516578, …}
{"id": "cpu_usage_average ", "timestamp": 1529023098976, …}

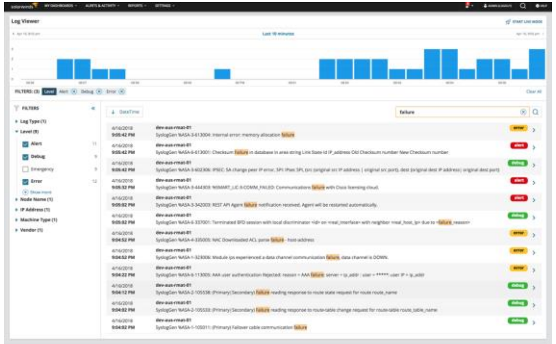
Topology. --

Troubleshooting
Monitoring Data Sources

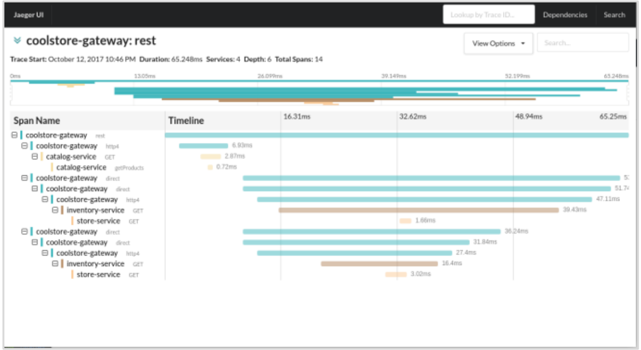
Metric analysis



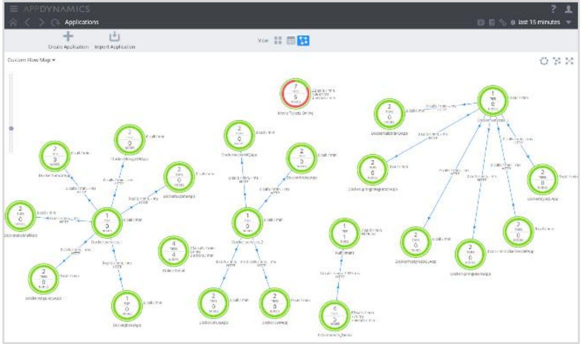
Log analysis



Trace analysis



Topology analysis



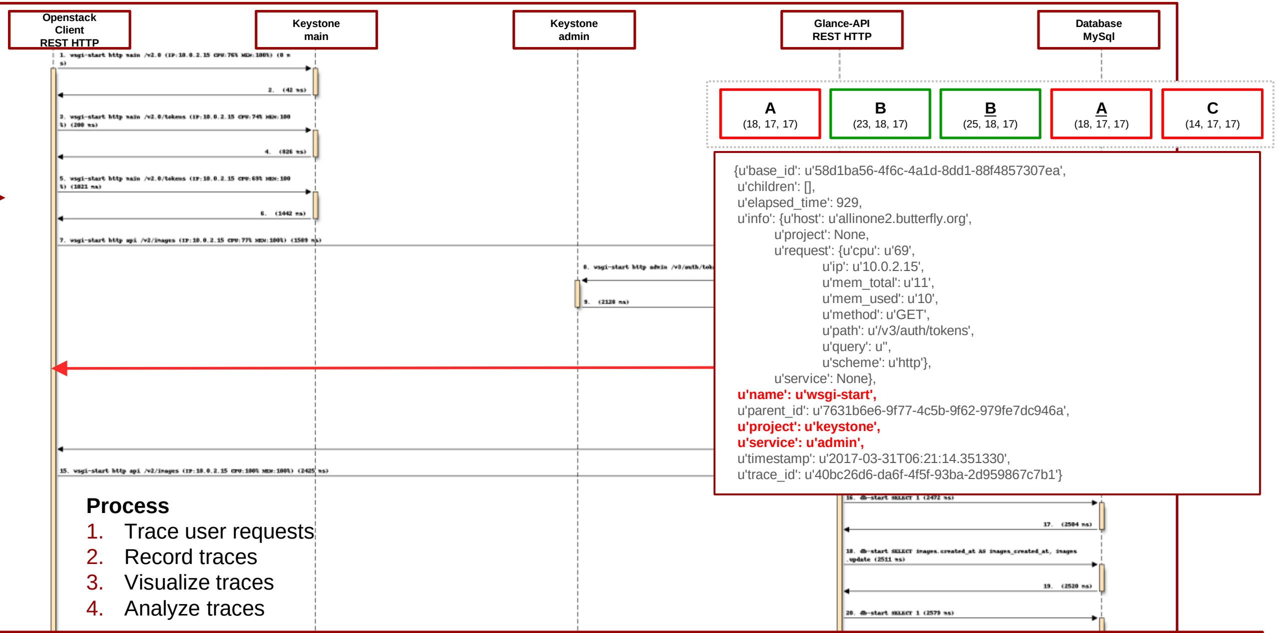
OpenStack
Troubleshooting using Distributed Tracing

OpenStack

Tracing Infrastructure

Distributed Tracing
- Distributed context propagation is still new to many people

Applications
- Distributed transaction monitoring
- Anomaly detection
- Performance analysis
  - Latency optimization
- Dependency analysis
  - Who are my upstream and downstream dependencies?
  - How many different workflows depend on my service?
  - Is my service a critical (tier 1) service for core business flows?
  - How do my SLIs affect other services?
  - Will my service survive Halloween?
- Root cause analysis



{u'base_id': u'58d1ba56-4f6c-4a1d-8dd1-88f4857307ea',
u'children': [],
u'elapsed_time': 929,
u'info': {u'host': u'allinone2.butterfly.org',
u'project': None,
u'request': {u'cpu': u'69',
u'ip': u'10.0.2.15',
u'mem_total': u'11',
u'mem_used': u'10',
u'method': u'GET',
u'path': u'/v3/auth/tokens',
u'query': u'',
u'scheme': u'http'},
u'service': None},
u'name': u'wsgi-start',
u'parent_id': u'7631b6e6-9f77-4c5b-9f62-979fe7dc946a',
u'project': u'keystone',
u'service': u'admin',
u'timestamp': u'2017-03-31T06:21:14.351330',
u'trace_id': u'40bc26d6-da6f-4f5f-93ba-2d959867c7b1'}

Process
1. Trace user requests
2. Record traces
3. Visualize traces
4. Analyze traces

1.
```
service_tracing(ctx)
process_tracing(ctx)
progress_task_tracing
horizontal_process
end_tracing(ctx)
hold_tracing(ctx)
start_tracing(ctx)
```

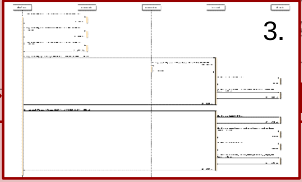
Trace Requests

2.

{
"info": {
"host": "request": {
"ip": "10.0.2.15",
"mem_used": "8",
"mem_total": "11",
"query": "",
"path": "/v2.
0",
"scheme": "http",

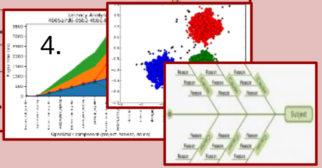
Record Traces

3.



Visualize Traces

4.



Performance analysis, Root-cause analysis, anomaly detection

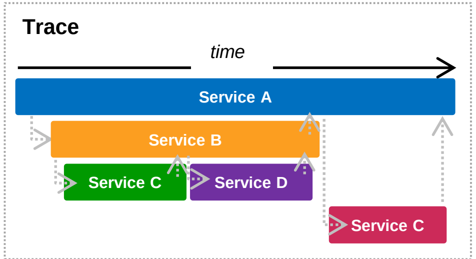
Troubleshooting using Distributed Tracing
Feature Selection

Trace content

- Context
  - Response time
  - Parent-child relationship
  - Timestamp
  - Host, IP, port
- Application payload
  - MEM, CPU, SQL query

Span feature selection

- Which features to select? Irrelevant features adversely impact model performance
- Use domain knowledge from the field of distributed systems to build a set of ad hoc features
- *Filter based (univariate selection)*
  - Statistical tests find with a strongest relationship with the output variable
  - Nominal variables, e.g., chi-squared (chi²) and mutual information
  - Ordinal variables, e.g., Kendall's Tau
  - Numerical, e.g., Pearson Correlation
- Wrapper-based
  - Selection is viewed as a search problem
- Embedded
  - Use algorithms with built-in feature selection methods
  - e.g., Random Forests



Raw data


```
{u'base_id': u'58d1ba56-4f6c-4a1d-8dd1….',
 u'children': [],
 u'elapsed_time': 929,
 u'info': {u'host': u'allinone2.butterfly.org',
         u'project': None,
         u'request': {u'cpu': u'69',
                    u'ip': u'10.0.2.15',
                    u'mem_total': u'11',
                    u'mem_used': u'10',
                    u'method': u'GET',
                    u'path': u'/v3/auth/tokens',
                    u'scheme': u'http'},
         u'service': None},
 u'name': u'wsgi-start',
 u'parent_id': u'7631b6e6-9f77-4c5b-9f6…,
 u'project': u'keystone',
 u'service': u'admin',
 u'timestamp': u'2017-03-31T06:21:14….',
 u'trace_id': u'40bc26d6-da6f-4f5f-93ba-2...'}
```

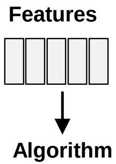



Figure. (above) Trace structure. (bellow) Content of a span. The field info contains the application payload.

Distributed Traces
Trace Abstraction

- **Markov Chain**
  - A stochastic process is a Markov chain if:

$$p(x_1 \ldots x_n) = p(X_1 = x_1) \cdot \prod_{i=2}^{n} p(X_i = x_i \mid X_{i-1} = x_{i-1}) = p(x_1) \cdot \prod_{i=2}^{n} a_{x_{i-1} x_i}$$

  - The probability distribution of a state X_i depends on the previous state X_{i-1} and does not depend on the previous states
  - K-th order Markov Chain:

$$p(x_1 \ldots x_n) = p(X_1 = x_1, \ldots, X_k = x_k) \cdot \prod_{i=k}^{n} p(X_i = x_i \mid X_{i-1} = x_{i-1}, X_{i-2} = x_{i-2}, \ldots, X_{i-k} = x_{i-k})$$

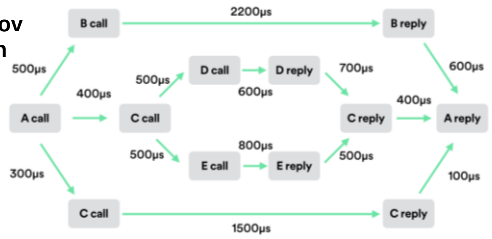
- **Tree Structure**
  - Finite set of one or more nodes A, B, C, …
  - Special nodes: the root node has no parent. Leaf nodes have children.
  - Remaining nodes are partitioned into *n* >= 0 disjoint sets T1, ..., Tn, where each set is also a tree

- **Sequence of Events**
  - Given a set *E* = {*e*1,...,*en*} of event types, an event is a pair (*A*, t), where *A* ∈ *E* and *t* ∈ *N* is the occurrence time of the event
  - An event sequence *s* on *E* is an ordered sequence of events:

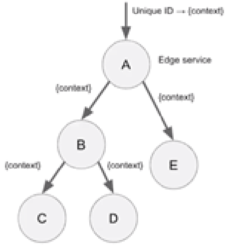
$$s = (A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n)$$

  - Tree representation: (A (B (C, D), E)

Markov Chain



Tree

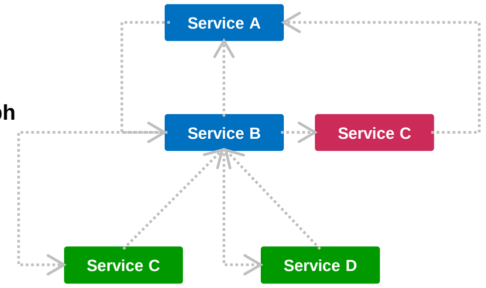


Sequence

Distributed Traces
Trace Abstraction

- **Graph Structure**
  - Graph is a set of vertices V, with edges connecting some of the vertices (edge set E).
  - An edge can connect two vertices.
  - Use maximum common sub-graph isomorphism (MCS) and the graph edit distance (GED) to compare graphs
  - e.g., GED defines the minimal number of operations (node/edge substitution, insertion, removal) needed to transform one graph into another
  - MCS and GED problems are NP-hard

Graph



Troubleshooting using Distributed Tracing
Techniques and Methods

Distributed Trace Analysis

- *Time series analysis*
  - Spans and traces seen as time series of response time
  - Statistical methods
  - Parametric and non-parametric
- *Sequence analysis*
  - Traces seen as sequences of spans
  - Methods: Clustering, classification, knowledge-based
  - Neural networks, Bayesian, SVM, decision trees, DBSCAN, K-mean, k-NN
- *Graph analysis*
  - Traces seen as graphs or trees
  - Methods: Markov chains, graph distance metrics, sub-graph isomorphism

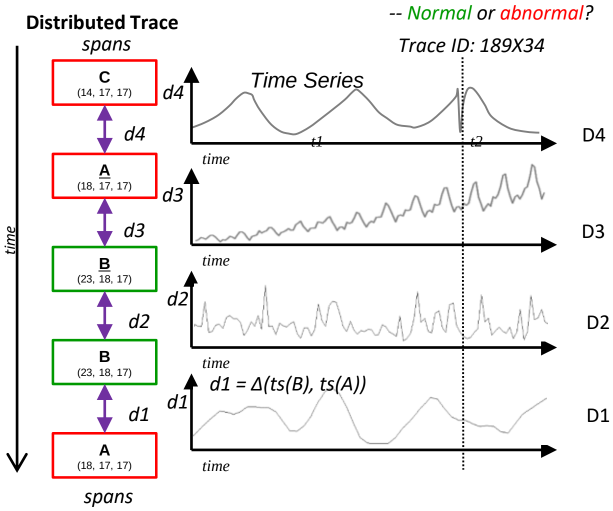
Distributed Trace Analysis
Time Series Analysis

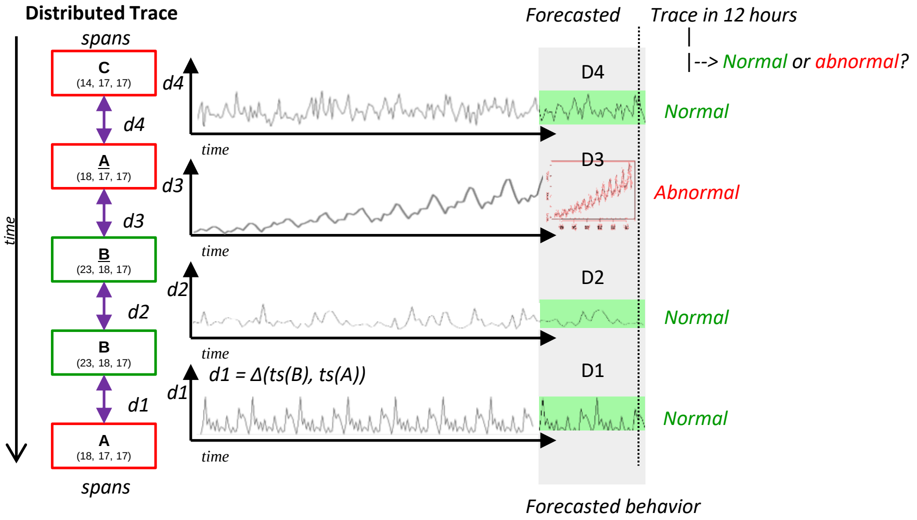
- **Time Series** $\{x_t, t = 0,1,2,\cdots\}$
  - Sequence of observations, measured at successive equally spaced time intervals collected from a process
- **Time series analysis**
  - Targets to understand the context of observations or to make predictions.
- **Time series forecasting**
  - Relies on models to forecast future observations based on previous ones.

- **Anomaly detection using exponential smoothing**
  - Exponential smoothing weighs recent observations more than older ones

$$S_t = ax_t + (1-\alpha)\cdot S_{t-1}$$

  - α is the smoothing constant
  - S_t is the smoothed value of observations
  - Forecast $F_{t+1} = S_t$





Representation: Discrete Fourier transformation, singular value decomposition, piecewise aggregate approximation, symbolic aggregate approximation, spline representation, etc.

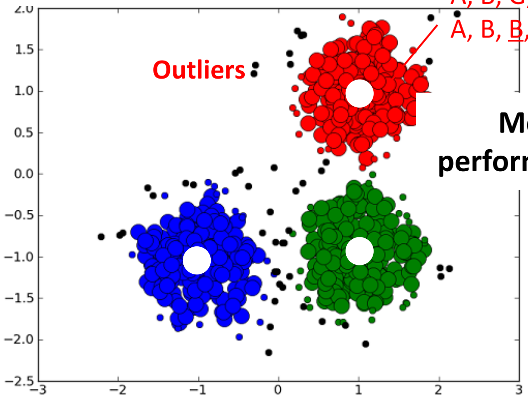
Distributed Trace Analysis
Sequence Analysis

Distributed Trace

		Span	*Span*						
A (18, 17, 17)	**B** (23, 18, 17)	**<u>B</u>** (23, 18, 17)	**<u>A</u>** (18, 17, 17)	**C** (14, 17, 17)	**D** (26, 14, 17)	**E** (34, 26, 17)	**<u>E</u>** (34, 26, 17)	**<u>D</u>** (26, 14, 17)	**<u>C</u>** (18, 17, 17)

time →

Clustering

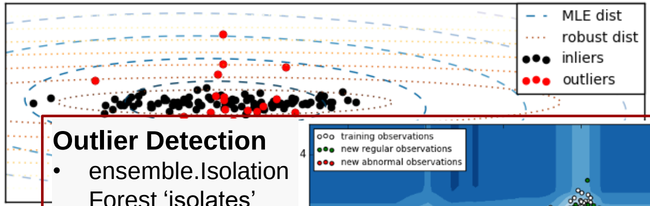


A, B, <u>B</u>, <u>A</u>, C, D, E, <u>E</u>, <u>D</u>, <u>C</u>
A, B, G, <u>G</u>, <u>B</u>, <u>A</u>, C, D, E, <u>E</u>, D, <u>C</u>
A, B, <u>B</u>, <u>A</u>, C, D, E, <u>E</u>, D, <u>C</u>, F, <u>F</u>

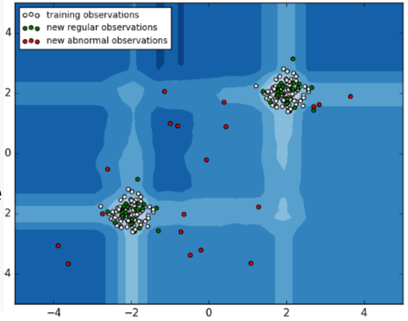
More accurate performance estimation

Extreme Value Analysis
- Assume a distribution (Gaussian) for duration of traces
- Look for values more than 2 or 3 standard deviations from the mean or 1.5 times from the first or third quartile
- Filter out outliers candidates

Outlier Detection
- Unsupervised learning from traces:
  - estimator.fit(X_train)
  - covariance.EllipticEnvelope
- new traces can then be sorted as outliers:
  - estimator.predict(X_test)
- Inliers are labeled 1, while outliers are labeled -1.



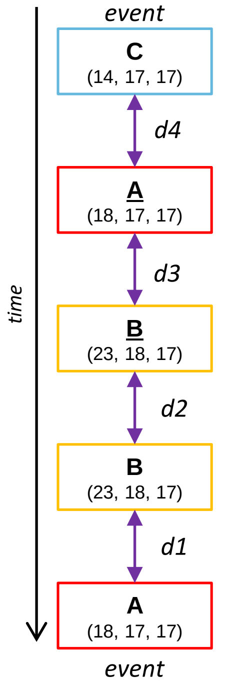
Outlier Detection
- ensemble.Isolation Forest 'isolates' traces by randomly selecting a feature
- Then randomly selects a split value between the maximum and minimum values

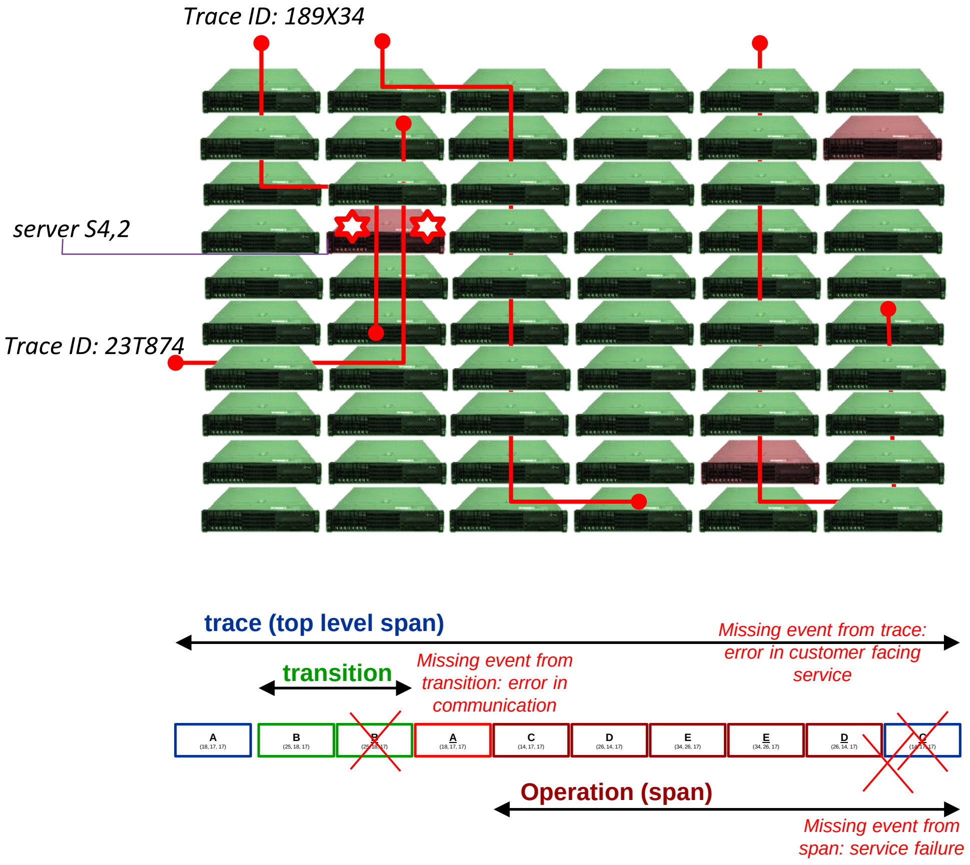


- **Proximity Methods**
  - Use clustering to identify the natural clusters of similar traces
  - Identify and mark the cluster centroids
  - Identify traces that are a fixed distance or percentage distance from cluster centroids
  - Filter out outliers candidates
- **Distance Measure**
  - Eucledian distance
  - Dynamic time warping
  - Longest common subsequence
  - Etc.

Distributed Trace Analysis
Sequence Analysis

Distributed Trace



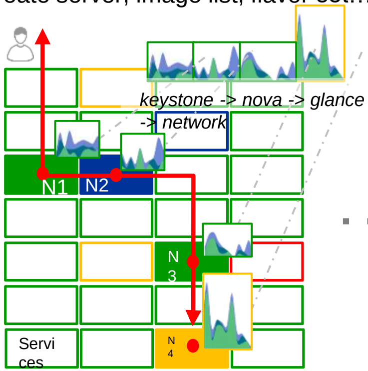


Trace change analysis

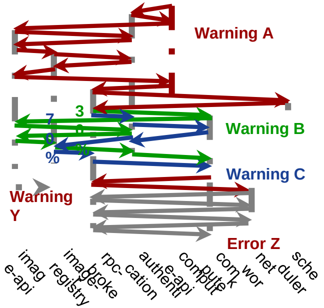
- Trace invariants. A program invariant is a predicate that always holds the same value under different workloads or inputs.
- By checking whether a trace sequence violates the invariants, we can detect system problems.
- A timeout anomaly occurs when an expected event in a trace is not seen within an expected time interval.

Distributed Trace Analysis
Sequence Analysis

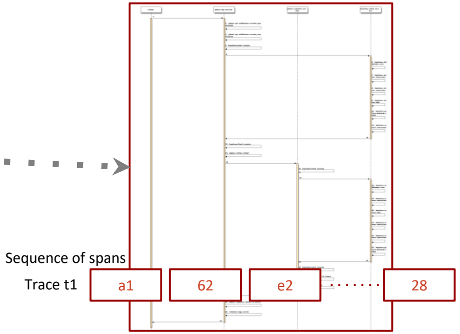
1. User commands: image delete, create server, image list, flavor set…



2. A **distributed trace** is generated



3. The distributed trace is encoded into a **sequence**



Trace of Openstack create server: scheduler, authentication, image-api, rpc-broker, image-registry, compute-api, compute, …. [a1, 62, e2, …., 28]

{"traceId": "72c53", "name": "get", "timestamp": 1529029301238, "id": "df332", "duration": 124957, “annotations": [{"key": "http.status_code", "value": "200"}, {"key": "http.url", "value": "https://v2/e5/servers/detail?limit=200"}, {"key": "protocol", "value": "HTTP"}, "endpoint": {"serviceName": "hss", "ipv4": "126.75.191.253"}]

4. Sequence Classification: identify invalid sequences

Function call sequences

Similar call sequence

Encoding matching
53.47%

auth_tokens_v3	1314421019193f1313162f253233191e341a3c3b402f2624192f214521162f163e2746282c192f2e2e24161d2c2c161628241635287a
images_v2	4342421119131**616161**a311316192216162f23213e1d2127272728241621191d5e5c471d132c4a16241616213c63
_var__detail_flavors_v2.1	5d1342131415131**6161**b193b21242113364024192f162f131d21161621212c27282c2c2d162f5e5c132c16164d
image_schemas_v2	1413191331131 62f1644161b331e351a213b1613133b13192f5116132113131616252121282c2c162d2158 2e2e161d13643b132c4a78134d213c21
gather_result	1912141613252128211a1f402f221921162f2713282b2d2416131d13211a5a21
security-groups_v2.0	4c1319131d19131316132f
_var__images_v2	121313251b2f192f35211f241916223b1613231616242713462c53241654192c285a21
_var___var__action_servers_v2.1	61135d106c12131325191b1c193d281a4022136d13162f3b292116231d1627272741283b2416131d5c2f4a 5121352f**13**
neutronclient.v2_0.client.retry_request	**16**2f1319132f165124191316

Service call encoding
13: **{'v3', 'tokens', 'auth‘}**

Function call sequences
[{'v3'}=200, {'v3', 'tokens', 'auth'}=200, {'v...

Service call encoding
16: **{'v2.0', 'ports'}**
51 : {'v2.0', '_var_', 'ports'}=200
24 : {'nova.compute.api.API.get'}=0
…

Tracing has applicability in many fields besides microservices, e.g., end-to-end enterprise applications orchestrating **mobile apps**, **edges**, **websites**, **storage**, and **centralized systems**.

Distributed Trace Analysis
Introduction to Sequence Analysis

- **Sequence Prediction**
  - Predict elements of a sequence on the basis of the preceding elements
  - Given si, si+1,…, sj, predict sj+1, si,si+1, i.e., sj →sj+1
  - Input: 1, 2, 3, 4, 5; output: 6
  - Applications: weather forecast

1, 2, 3, 4, 5 ▶ 6

- **Sequence Classification**
  - Predict a class label for a given input sequence.
  - Given si,si+1,…,sj, determine if it is legitimate, i.e., si,si+1,…,sj →yes or no
  - Input: 1, 2, 3, 4, 5; output: “normal” or “anomalous”
  - Applications: trace anomaly detection, DNA sequence classification

1, 2, 3, 4, 5 ▶ NORMAL

- **Sequence Generation**
  - Generate new output sequence with same general characteristics as the input
  - Input: [1, 3, 5], [7, 9, 11]; output: [3, 5 ,7]
  - Applications: text generation

1, 3, 5 and 7, 9, 11 ▶ 3, 5 ,7

- **Sequence to Sequence Prediction**
  - Predict an output sequence given an input sequence.
  - Input: 1, 2, 3, 4, 5; output: 6, 7, 8, 9, 10
  - Applications: text summarization

1, 2, 3, 4, 5 ▶ 6, 7, 8, 9, 10

Sequence learning: from recognition and prediction to sequential decision making, IEEE Intelligent Systems, 2001, http://www.sts.rpi.edu/~rsun/sun.expert01.pdf

Related Work
Facebook Mystery Machine

Measure end-to-end performance of requests

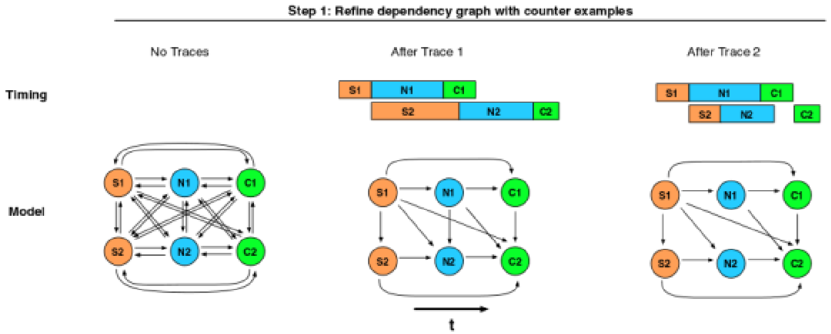
- **Infer causal relationships from logs**
  - Adding instrumentation retroactively is an expensive task
  - Hypothesize and confirm relationships in messages
- **Log Data**
  - Request & host id, timestamp, unique event label
- **Finding Relationships**
  - Samples requests, store logs in Hive and run Hadoop jobs to infer causal relationships
  - 2h Hadoop to analyze 1.3M requests sampled 30 days
  - Assumes a hypothesized relationship between two segments until finding a counterexample
  - 3 types of relationship inferred: happens-before, mutual exclusion, pipeline
- **Applications**
  - Find critical path and slack segments for performance optimization
  - Anomaly detection: 1) select top 5% of end-to-end latency, 2) identify segments with proportionally greater representation in the outlier set of requests than in the non-outlier set.

***The Mystery Machine:* End-to-end Performance Analysis of Large-scale Internet Services**

Michael Chow, *University of Michigan;* David Meisner, *Facebook, Inc.;*
Jason Flinn, *University of Michigan;* Daniel Peek, *Facebook, Inc.;*
Thomas F. Wenisch, *University of Michigan*

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chow

facebook



As the number of traces analyzed increases, the observation of new counter examples diminishes, leaving behind only true relationships

https://blog.acolyer.org/2015/10/07/the-mystery-machine-end-to-end-performance-analysis-of-large-scale-internet-services/
https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-chow.pdf

Sequence Prediction using Logs

- **Parsing**
  - Spell tool (ICDM'16) parses logs into patterns that represent the fixed part of printf-like statements
  - Log messages ► Log key
  - https://www.cs.utah.edu/~lifeifei/papers/spell.pdf
- **Processing**
  - Workflow models are built to help anomaly diagnosis
  - Log Key ► Workflow
  - Log Key + Parameters ► Behavior Model
  - LSTM is used to model system execution paths and log parameter values
- **Precision**
  - F1-score: 96%
- **How many scenarios need to be labeled? 10 or 10,000?**
- **Dataset distribution**
  - HDFS: 2.9% labeled anomalies
  - Openstack: 7% anomalies
- **Do precision numbers hold in more realistically distributed logs?**

log message (log key underlined)	log key	parameter value vector
t_1 Deletion of **file1** complete	k_1	$[t_1 - t_0, \text{file1Id}]$
t_2 Took **0.61** seconds to deallocate network ...	k_2	$[t_2 - t_1, 0.61]$
t_3 VM Stopped (Lifecycle Event)	k_3	$[t_3 - t_2]$
...	...	...

Table 1: Log entries from OpenStack VM deletion task.

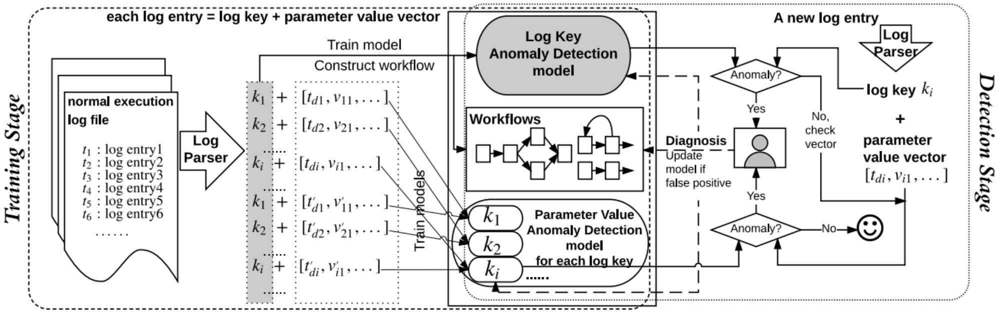


Figure 1: DeepLog architecture.

	PCA	IM	TFIDF	N-gram	DeepLog
false positive (FP)	277	2122	95833	1360	833
false negative (FN)	5400	1217	1256	739	619

Table 4: Number of FPs and FNs on HDFS log.

Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). ACM, New York, NY, USA, 1285-1298.

Distributed Trace Analysis Using LSTMs

1 Generate Synthetics Traces

```
OPENSTACK_SEQUENCE = [
    ('keystone', 1.0, 'authenticate user with credentials and generate auth-token'),
    ('nova-api', 1.0, 'get user request and sends token to Keystone for validation'),
    ('keystone', 0.2, 'validate the token if not in cache'),
    ('nova-api', 1.0, 'starts VM creation process'),
    ('nova-database', 1.0, 'create initial database entry for new VM'),
    ('nova-scheduler', 1.0, 'locate an appropriate host using filters and weights'),
    ('nova-database', 1.0, 'execute query'),
    ('nova-compute', 1.0, 'dispatches request'),
    ('nova-conductor', 1.0, 'gets instance information'),
    ('nova-compute', 1.0, 'get image URI from image service'),
    ('glance-api', 1.0, 'get image metadata'),
    ('nova-compute', 1.0, 'setup network'),
    ('neutron-server', .5, 'allocate and configure network IP address'),
    ('nova-compute', 1.0, 'setup volume'),
    ('cinder-api', .75, 'attach volume to the instance or VM'),
    ('nova-compute', 1.0, 'generates data for the hypervisor driver')
]
```

Cache Simulation: Probability of execution = 75%

```
[0,
 ['keystone',
  'nova-api',
  'nova-api',
  'nova-database',
  'nova-scheduler',
  'nova-database',
  'nova-compute',
  'nova-conductor',
  'nova-compute',
  'glance-api',
  'nova-compute',
  'nova-compute',
  'cinder-api',
  'nova-compute']]
```

Trace #0

3 Transform traces with service names into sequences of integers

Train dataset

```
                                         span_list
trace_id
0            [3, 5, 5, 8, 9, 8, 6, 7, 6, 2, 6, 6, 1, 6]
1            [3, 5, 5, 8, 9, 8, 6, 7, 6, 2, 6, 6, 1, 6]
2         [3, 5, 5, 8, 9, 8, 6, 7, 6, 2, 6, 4, 6, 1, 6]
3            [3, 5, 5, 8, 9, 8, 6, 7, 6, 2, 6, 4, 6, 6]
4         [3, 5, 3, 5, 8, 9, 8, 6, 7, 6, 2, 6, 6, 1, 6]
...                                                ...
95           [3, 5, 5, 8, 9, 8, 6, 7, 6, 2, 6, 6, 1, 6]
96        [3, 5, 3, 5, 8, 9, 8, 6, 7, 6, 2, 6, 4, 6, 6]
97           [3, 5, 5, 8, 9, 8, 6, 7, 6, 2, 6, 6, 1, 6]
98        [3, 5, 5, 8, 9, 8, 6, 7, 6, 2, 6, 4, 6, 1, 6]
99        [3, 5, 5, 8, 9, 8, 6, 7, 6, 2, 6, 4, 6, 1, 6]
[100 rows x 1 columns]
```

Test dataset

```
                                      span_list
trace_id
0         [3, 5, 5, 1, 9, 8, 6, 7, 6, 2, 6, 6, 1, 6]
1         [3, 5, 5, 8, 9, 8, 6, 7, 6, 2, 6, 4, 6, 6]
```

2 Encoding traces sequences

- The encoding scheme pads each sequence of inputs with zeros, up to a pre-defined maximum length.
- This allows to pre-allocate a chain of LSTM units of a specific length
- We also pass information about the sequence lengths. This is important for not treating the zero padding as actual inputs, and from injecting the error signal at the right unit in the sequence during back propagation.

Hot encoding

- A one hot encoding enables to represent categorical variables as binary vectors.
- Categorical values are mapped to integer values.
- Each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

4

```
# Padding is done using using keras.preprocessing.sequence.pad_sequences(sequences, ...)
# maxlen: Int, maximum length of all sequences.
# truncating='pre' remove values at the beginning from sequences larger than maxlen
# padding='pre' pads each trace at the beginning with a special integer (e.g., 0)
X = pad_sequences(X, maxlen=self.trace_size, dtype=np.int16,
                  truncating='pre', padding='pre', value=PADDING_SYMBOL_INT)
```

Padding with symbol 0

```
[[0 0 3 ... 6 1 6]
 [0 0 3 ... 6 1 6]
 [0 3 5 ... 6 1 6]
 ...
 [0 0 3 ... 6 1 6]
 [0 3 5 ... 6 1 6]
 [0 3 5 ... 6 1 6]]
```

Hot encoding not shown

Tensorflow tutorial: https://github.com/campdav/text-rnn-tensorflow
Keras tutorial: https://github.com/campdav/text-rnn-keras

Code available at: https://github.com/jorge-cardoso/aiops_practice
Subdirectory: dta_lstm

5 Shift traces left

```
x [[0 0 3 ... 6 1 6]
   [0 0 3 ... 6 1 6]
   [0 3 5 ... 6 1 6]
   ...
   [0 0 3 ... 6 1 6]
   [0 3 5 ... 6 1 6]
   [0 3 5 ... 6 1 6]]
```

```
# X.shape and y.shape are (n_traces, self.trace_size)
y = np.roll(X, -1, axis=1)

# Pad j array where X array has zeros
y[X == 0] = DEFAULTS['padding_symbol']

# Write the DEFAULTS['padding_symbol'] at the last position of the shifted array
y[:, -1] = DEFAULTS['padding_symbol']
```

```
y [[0 0 5 ... 1 6 0]
   [0 0 5 ... 1 6 0]
   [0 5 5 ... 1 6 0]
   ...
   [0 0 5 ... 1 6 0]
   [0 5 5 ... 1 6 0]
   [0 5 5 ... 1 6 0]]
```

6 Create the DL model

```
def _build_model(self, input_dim=0):
    """ Creates an LSTM model (for sequence to sequence mapping)
    """
    model = Sequential()
    model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2, return_sequences=True,
                   input_shape=(self.trace_size, input_dim)))
    model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))
    model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))

    # A Dense layer is used as the output for the network.
    model.add(TimeDistributed(Dense(input_dim, activation='softmax')))
    if self.gpus > 1:
        model = keras.utils.multi_gpu_model(model, gpus=self.gpus)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model
```

Keras and **Tensorflow**

- **LSTM** network
- **dropout layer of 0.2** (high, but necessary to avoid quick divergence)
- **Softmax** activation to generate probabilities over the different categories
- Because it is a classification problem, loss calculation via **categorical cross-entropy** compares the output probabilities against one-one encoding
- **ADAM** optimizer (instead of the classical stochastic gradient descent) to update weights

Distributed Trace Analysis Using LSTMs

7 Train the model

```
# The X_train.shape is (n_traces, self.trace_size, self.max_n_span_types)
# e.g. (21251, 20, 33)
self.model = self._build_model(input_dim=X.shape[2])

self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size, shuffle=True,
               validation_split=self.validation_split)
```

```
50/85 [================>.............] - ETA: 0s - loss: 2.2797 - acc: 0.3150
85/85 [==============================] - 1s 9ms/step - loss: 2.2738 - acc: 0.3140 - val_loss: 2.2466 - val_acc: 0.3125
Epoch 3/100

50/85 [================>.............] - ETA: 0s - loss: 2.2468 - acc: 0.3125
85/85 [==============================] - 1s 6ms/step - loss: 2.2379 - acc: 0.3125 - val_loss: 2.1908 - val_acc: 0.3125
Epoch 4/100

50/85 [================>.............] - ETA: 0s - loss: 2.1910 - acc: 0.3125
85/85 [==============================] - 1s 6ms/step - loss: 2.1747 - acc: 0.3125 - val_loss: 2.0961 - val_acc: 0.3125
Epoch 5/100

50/85 [================>.............] - ETA: 0s - loss: 2.1000 - acc: 0.3125
85/85 [==============================] - 1s 8ms/step - loss: 2.0711 - acc: 0.3125 - val_loss: 1.9812 - val_acc: 0.3125
Epoch 6/100
```

8 Test traces

```
X_test = self._pad_traces(X_test)
X_test_bin = self._traces_to_binary(X_test)
# e.g., X_test_bin[0][0] = array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0....], dtype=int32)

yhat = self.model.predict(X_test_bin)
# e.g.,
# yhat.shape = (n_traces, self.trace_size, self.max_n_span_types)
# yhat[0][0] = [9.9969006e-01, 2.2644450e-05, 3.9419938e-06, 2.8681773e-09, ... ]

self._identify_anomalies(X_test, yhat, prob=self.threshold)  # idx -> True/False
```

```
Test Traces
X [[0 0 3 5 5 1 9 8 6 7 6 2 6 6 1 6]
   [0 0 3 5 5 8 9 8 6 7 6 2 6 4 6 6]]

X[0]: [0 0 3 5 5 1 9 8 6 7 6 2 6 6 1 6] =
[[0, 0.2625601],  [1, 0.38167596],   [0, 0.75843567],
 [0, 0.8912414],  [9, 2.394792e-05], [0, 0.94694203],
 [0, 0.9656691],  [0, 0.9686248],    [0, 0.9666971],
 [0, 0.95213455], [0, 0.94981205],   [0, 0.93016535],
 [0, 0.60129535], [0, 0.5417027],    [0, 0.6876041],
 [0, 0.82936114]]

X[1]: [0 0 3 5 5 8 9 8 6 7 6 2 6 4 6 6] =
[[0, 0.2625601], [1, 0.38167596], [0, 0.75843567],
 [0, 0.8912414], [0, 0.9302344],  [0, 0.94507533],
 [0, 0.9638574], [0, 0.9665326],  [0, 0.9627945],
 [0, 0.9435249], [0, 0.9405963],  [0, 0.92075515],
 [1, 0.3207201], [1, 0.46595544], [0, 0.64980185],
 [0, 0.8390282]]

trace_id 0 indices [4]
trace_id 1 indices []
```

Distributed Trace Analysis Using LSTMs

9 Identifying Anomalous Traces Sequences

```
def _identify_anomalies(self, X, yhat, prob):
    """
        mark anomalies based on the difference between X, y, and yhat

        i.e.,

        top_k = 5
        top_k_yhat = np.argsort(yhat, axis=2)[:, :, -top_k:]
        top_yhat = np.argmax(yhat, axis=2)

        y = _shift_traces_left(X)

        X[i]: Input:        [ 0  0  0  0  0  0  0  0  0  0  0 10 10 17  5  5  7  7  6  6]
        y[i]: Output:       [ 0  0  0  0  0  0  0  0  0  0  0 10 17  5  5  7  7  6  6  0]
        yhat[i]: Predicted: [ 0  0  0  0  0  0  0  0  0  0  0 10 17  5  5  7  7  6  6  0]
    """

    y = DTA_LSTM._shift_traces_left(X)

    for i in range(len(X)):

        rp = self._compare_traces(y[i], yhat[i])
        self.rank_prob.append(rp)

        if self.explain:
            print('X[{}]: {} = {}'.format(i, X[i], rp))

    return self.rank_prob
```

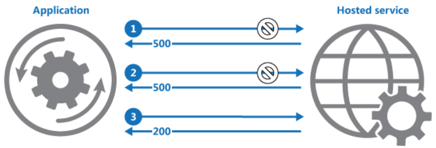
<u>Test Trace</u>

```
Input X    = [0 0 3 5 5 1 9 8 6 7 6 2 6 6 1 6]

Shifted_X = [0 0 5 5 1 9 8 6 7 6 2 6 6 1 6 0]


Prediction
yhat =       [0 0 5 5 12 9 8 6 7 6 2 6 6 1 6 0]


X[0]: [0 0 3 5 5 1 9 8 6 7 6 2 6 6 1 6] =
[[0, 0.2625601],  [1, 0.38167596],   [0, 0.75843567],
 [0, 0.8912414],  [9, 2.394792e-05], [0, 0.94694203],
 [0, 0.9656691],  [0, 0.9686248],    [0, 0.9666971],
 [0, 0.95213455], [0, 0.94981205],   [0, 0.93016535],
 [0, 0.60129535], [0, 0.5417027],    [0, 0.6876041],
 [0, 0.82936114]]

Error at index [4]
```

Distributed Trace Analysis
Why is trace analysis challenging?

Many "Hidden" Software "Patterns" which affect running systems…Circuit breaker, concurrency and parallelism, priority queues, publisher-subscriber, bulkhead, etc.

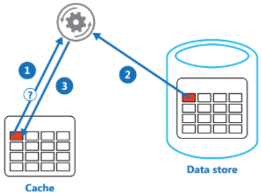
1. Retry pattern




```
n_tries = 3
while True:
  try:
     ExecuteOperation() # Success
     break
  except TimeoutException:
     if n_tries == 0:    # give up
       raise Failure
     else:
       sleep(1000) # Wait until retrying the call
```

T1: A, B, C, D, E, …
T2: A, B, C, C, D, E, ….
T3: A, B, C, C, C, D, E, …
T4: A, B, C, C, C, Z

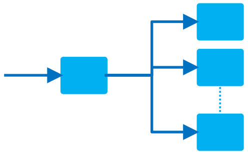
2. Cache-Aside pattern




```
v = cache.get(k)
if v == None:         # Check if the item is in the cache
   v = store.get(k) # If not in cache, read from data store
    cache.put(k, v) # Put a copy of the item into the cache
```

T1: A, B, C, D, E, …
T1: A, C, D, E, …

3. One-to-many subcalls



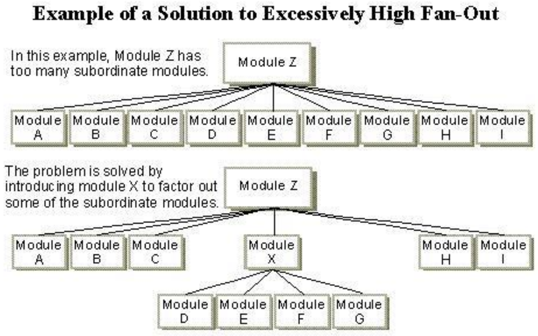
```
@app.route('/servers/<int:user_id>/<int:number>')
def create(user_id, number):
  for i in range(number):
     authentication(user_id)
     # Call remote microservice to create a server
     rpc.create_server()
```

T1: A, B, C, D, E, …
T2: A, B, C, C, D, E, ….
T3: A, B, C, C, C, D, E, …
T4: A, B, C, C, C, C, D, E, …

Distributed Trace Analysis
Why is trace analysis challenging?

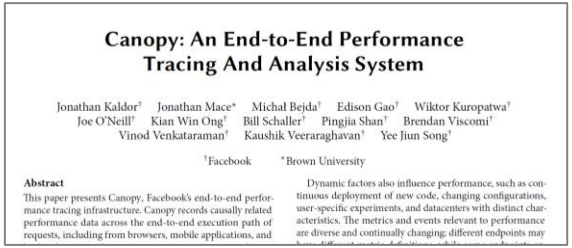
More design patterns

- Design Principles: Fan-In vs Fan-Out
- Lazy loading (also called on-demand loading) is an optimization technique for the online content, be it a website or a web app
- Chunking is a specific feature of the HTTP 1.1 protocol. Here, the meaning is the opposite of that used in memory management. It refers to a facility that allows inconveniently large messages to be broken into conveniently-sized smaller "chunks"



Distributed Trace Analysis
Why is trace analysis challenging?

Inspired by Facebook Canopy design
(http://cs.brown.edu/~jcmace/papers/kaldor2017canopy.pdf)



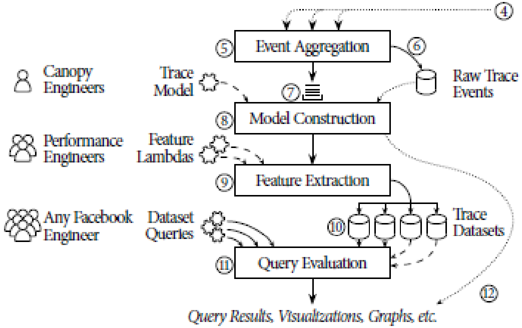
Canopy: An End-to-End Performance Tracing And Analysis System

Jonathan Kaldor† Jonathan Mace* Michał Bejda† Edison Gao† Wiktor Kuropatwa†
Joe O'Neill† Kian Win Ong† Bill Schaller† Pingjia Shan† Brendan Viscomi†
Vinod Venkataraman† Kaushik Veeraraghavan† Yee Jiun Song†

†Facebook *Brown University

Abstract
This paper presents Canopy, Facebook's end-to-end performance tracing infrastructure. Canopy records causally related performance data across the end-to-end execution path of requests, including from browsers, mobile applications, and

Dynamic factors also influence performance, such as continuous deployment of new code, changing configurations, user-specific experiments, and datacenters with distinct characteristics. The metrics and events relevant to performance are diverse and continually changing; different endpoints may



(b) Canopy's tailer aggregates events (5), constructs model-based traces (8), evaluates user-supplied feature extraction functions (9), and pipes output to user-defined datasets (10). Users subsequently run queries, view dashboards and explore datasets (11,12).

Facebook uses end-to-end tracing for all services, from data-center to mobile applications. As of 2017 the system processes 1.3 billion traces per day, each trace contains up to several thousands of events.

Span Model

- Introduced in Google Dapper in 2010, base model in Zipkin
- Suitable to describe synchronous REST operations
- Every span consists of 4 events: "client send", "server receive", "server send", "client receive"

Limitations of span model

- Non-synchronous execution models such as queues and asynchronous executions can not be described as span trees
- Fine-grained metrics are hard to express, e.g. RPC response handler can read the data from network then enqueue it and then process – queue time is not taken into account
- Multithreaded processing, like spawning of a group of threads, then joining them requires additional tags to differ from sequential processing.

Evolution of trace model in Facebook*

a) Started with Dapper span model
b) Idle (blocked) time is taken into account
c) Internal queue is taken into account
d) Client-queue metrics for AJAX processing are added
e) Request and response are completely decoupled, trace is represented as set of events with causal relations

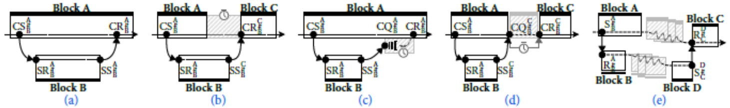


Figure 11: Evolution of Canopy's instrumentation and model for RPCs and network communication (cf. §5.3)

* http://cs.brown.edu/~jcmace/papers/kaldor2017canopy.pdf

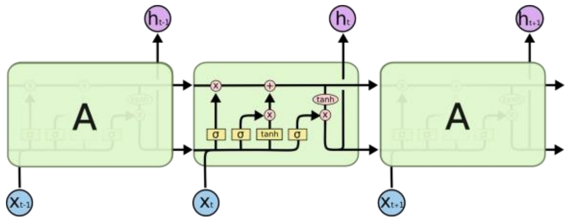
Distributed Trace Analysis
Using Multimodal Deep Learning

- **Monitoring and Incident Detection**
  - Anomaly detection in large-scale system is widely studied
  - IT monitoring systems typically use application logs and resource metrics to detect failures
    - Distributed tracing is becoming the third pillar of microservices observability
    - Logs and metrics were previously investigated (see [1-4])

- **Single and Bi-Models**
  - Use of single and bi-models to capture traces as sequences of events and their response time
  - Use sequential model representation by utilizing long-short-term memory (LSTM) networks

- **Results**
  - Detect anomalies in Huawei Cloud infrastructure
  - The novel approach outperforms other deep learning methods based on traditional architectures

Idea: represent distributed traces as sequences of events and their response time



Long Short Term Memory (LSTM)

LSTMs [1] are models which capture sequential data by using an internal memory. They are very good for analyzing sequences of values and predicting the next point of a given time series. This makes LSTMs adequate for Machine Learning problems that involve sequential data (see [3]) such **speech recognition**, **machine translation**, **visual recognition and description**, and **distributed traces analysis**.

[1] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
[2] Sequential processing in LSTM (from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[3] LSTM model description (from Andrej Karpathy. http://karpathy.github.io/2015/05/21/rnn-effectiveness/

S. Nedelkoski, J. Cardoso, O. Kao, **Anomaly Detection from System Tracing Data using Multimodal Deep Learning,** IEEE Cloud 2019, July 3-8, 2019, Milan, Italy.
S. Nedelkoski, J. Cardoso, O. Kao, **Anomaly Detection and Classification using Distributed Tracing and Deep Learning**, CCGrid 2019, 14-17.05, Cyprus.
J. Cardoso, **Mastering AIOps with Deep Learning**, Presentation at SRECon18, 29–31 August 2018, Dusseldorf, Germany.

Distributed Trace Analysis
Bi-modal Approach

- A span (event) is a vector of key-value pairs (k_i, v_i) describing a characteristic of a microservice at time t_i
- A trace T is an enumerated collection of events (i.e., spans) sorted by timestamps $\{e_0, e_1, \ldots, e_i\}$ [16]

- An event contains
  - *trace id, event id, parent id*
  - *protocol, host ip, status code, url*
  - *response time, timestamp*
  - *and much more*

- Trace can have different lengths
  - $T_p = \{e_0^p, e_1^p, e_2^p, \ldots, e_i^p\}$ and $T_q = \{e_0^q, e_2^q, e_1^q, \ldots, e_i^q\}$ are different (e_1 and e_2 are swapped) but originate from the same activity
  - Possibly caused by concurrent systems
- Label each span as
  - Label = *concat*(url, status code, host ip)
  - We have N_l labels
- Pad trace vector up to T_l or truncate traces

Trace Structure (sequence of events) and Response time (duration)

e.g., Service 11 → Service 21; duration = 12ms

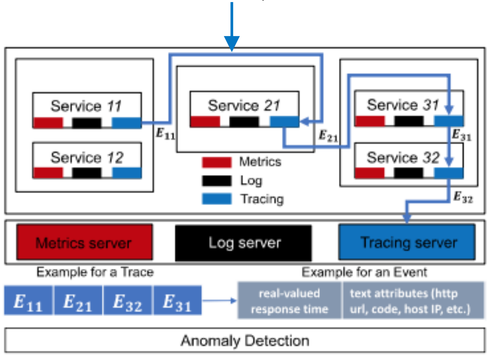


Fig. 1. Overall system architecture showing communication between services and the three system observability components. We combine two modalities of tracing data in a single model for anomaly detection in cloud infrastructures.

- **Trace structure**
  - Trace one-hot categorical encoding [17]
  - $D_1 = (N_t, T_l, N_l)$
  - N_t = # traces, T_l = max length, N_l = # labels
- **Response time**
  - Min-max scaling [0, 1]
  - $D_2 = (N_t, T_l, 1)$
  - Last dimension is the response time (duration)

Distributed Trace Analysis
Single-Modality LSTM (1)

- We model anomaly detection in traces as a sequence-to-sequence, multi-class, single-label problem
  - Use multiple possible labels for one trace that are not mutually exclusive.
  - The partial trace can have multiple subsequent events

- LSTM network architecture
  - SAD = Structural Anomaly Detection (D_1)
- Model input
  - $T_k = \{e_0, e_1, \dots, e_{T_l}\}$
- Model output
  - $\{e_0, e_1, \dots, e_{i-1}\}$
  - Probability distribution over the N_l unique labels from L, representing the probability for the next label (event) e_i in the sequence

- Compare the predicted output against the observed label
  - input $\{e_0, e_1, \dots, e_{T_l}\} \rightarrow$ output $\{e_1, \dots, e_{T_l}, '!0'\}$
  - The output is shifted by one event and padded
- Update network weights using categorical cross-entropy loss minimization via gradient descent

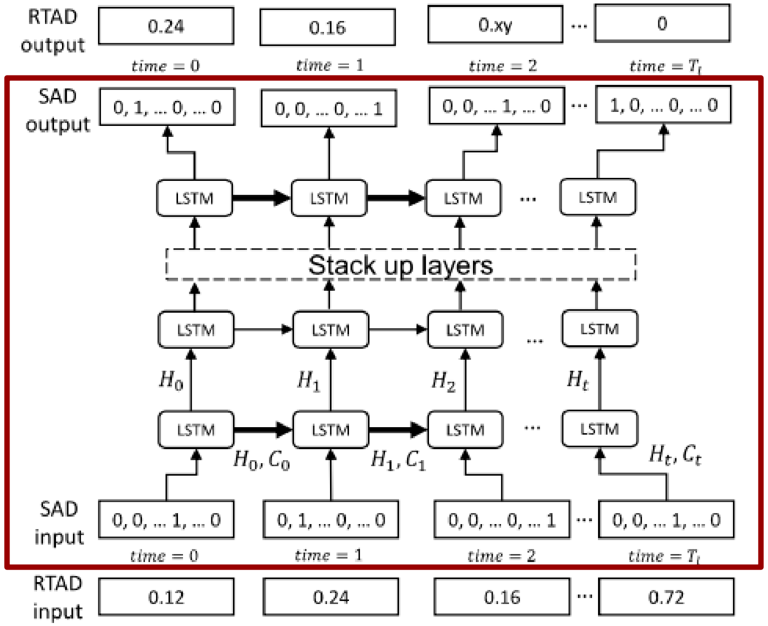


Fig. 2. Single-modality LSTM network architecture

SAD = Structural Anomaly Detection (D_1)

Distributed Trace Analysis
Single-Modality LSTM (2)

- Evaluate if trace T_{test} is anomalous
  - $T_{test} = \{e_0, e_1, \ldots, e_{T_l}\}$
- The network calculates
  - Probability distribution P
  - $P = \{l_0 : p_0, l_1 : p_1, \ldots, l_{N_t} : p_{N_t}\}$
  - Probability of a label of L to appear as the next label value in a trace
- The output layer has a ***softmax*** function
  - A generalization of the logistic function that "squashes" a *K*-dimensional vector *z* of arbitrary real values to a *K*-dimensional vector $\sigma(z)_i$ of real values in the range [0, 1] that add up to 1
- Distribute the probability over labels
  - $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
- Such that
  - $\sum_i^{N_l} p_i = 1$
- Classification
  - Accept **top-*k* predicted labels** as behaviorally **correct**
  - Otherwise, report an anomaly along with the events which contributed to the decision

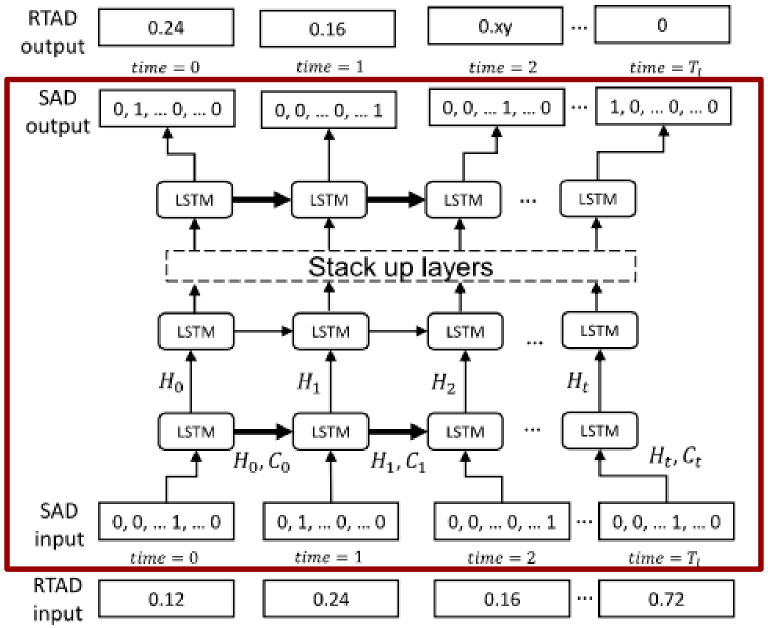


Fig. 2. Single-modality LSTM network architecture

SAD = Structural Anomaly Detection (D_1)

Distributed Trace Analysis
Single-Modality LSTM (3)

- We model response time anomaly detection in traces as a regression task
  - E.g., predict the duration of a span
- LSTM network architecture
  - RTAD = Response Time Anomaly Detection (D_2)
  - Linear (i.e. identity) activation function
- Approach
  - A each timestep $time = i$ with $\{rt_0, rt_1, \ldots, rt_{i-1}\}$
  - Predict the response time rt_i of the next event
- Update network weights using mean squared loss via gradient descent
- Detection
  - Compute the squared error distance
  - $error_i = \left(rt_i - rt_i^p\right)^2$, where rt_i^p is the predicted value at timestep i
  - errors are fitted by a Gaussian $N(0, \sigma^2)$
  - report trace as anomalous, if squared error between prediction and input at time i is out of 95% confidence interval

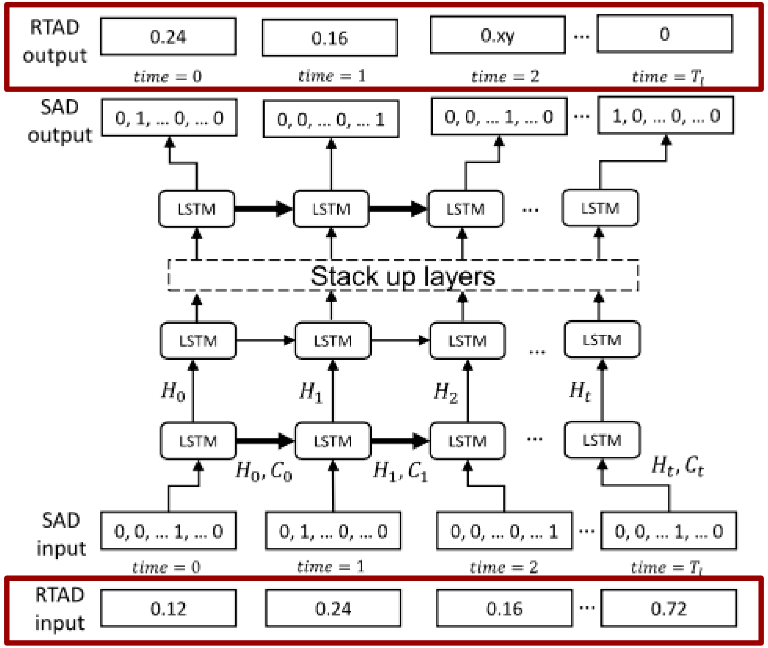


Fig. 2. Single-modality LSTM network architecture

RTAD = Response Time Anomaly Detection (D_2)

Distributed Trace Analysis
Multimodal LSTM (1)

- Explore the correlation between trace structure and response time
  - Improve accuracy/recall
- LSTM network architecture
  - Concatenation of both single-modality architectures in the second hidden layer
- Approach
  - *input* $[\{e_0, e_1, \dots, e_{T_l}\}, \{rt_0, rt_1, \dots, rt_{T_l}\}] \rightarrow$
  - *output* $[\{e_1, \dots, e_{T_l}, '!0'\}, \{rt_1, rt_2, \dots, rt_{T_l}, 0\}]$
- Update network weights using
  - Sum mean squared error
  - Categorical cross-entropy
- Detection
  - Performed by comparing the output element-wise with the input for both modalities using the strategy developed in the single-modality architectures
  - Anomaly source: 1) response time, 2) structural, or 3) both

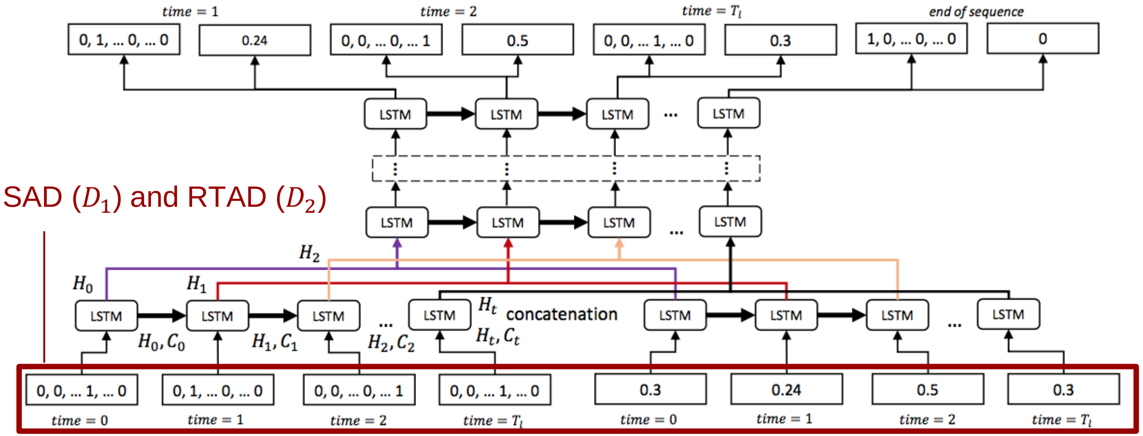


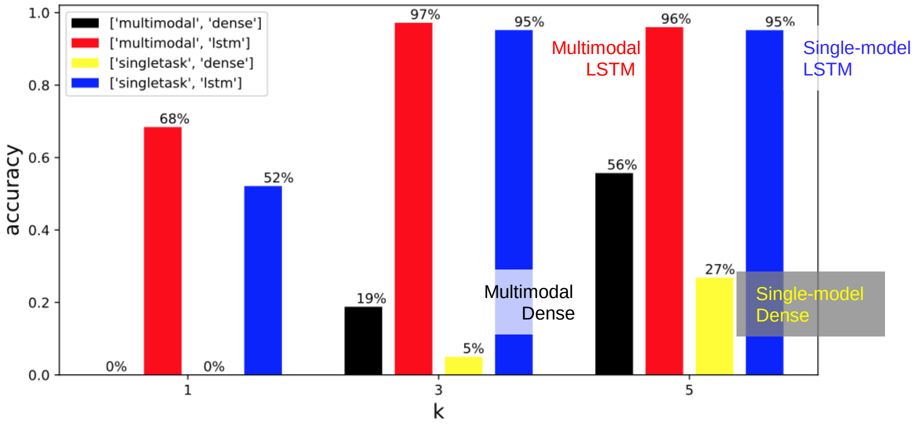
Fig. 3. Multimodal LSTM neural network architecture for anomaly detection from complete tracing data

Distributed Trace Analysis
Evaluation

- **Datasets**
  - System under study has 1000+ services running production Openstack-based cloud [19]
  - Traces collected using Zipkin [16] over 50 days: >4.5M events; 1M traces
- **Evaluation Platform**
  - Python using Keras, model with batch size = 512, learning rate of 0.001, and 400 epochs
  - PC using GPU-NVIDIA GTX 1060
- **Preprocessing**
  - To avoid outliers, select labels that appear more than 1000 times (105 unique labels)
  - Distribution of trace lengths is imbalanced
    - >90% have lengths <10 events
  - Select only 1000 samples of each trace length
    - Requires <1% of all the recorded data
    - Efficient and fast for training
  - For robustness, we also select the traces with lengths between 4 and 20
- **Performance**
  - Time to train multimodal LSTM on 1% traces (1M): approx. 30 min
  - Prediction time per trace: <50 ms

Distributed Trace Analysis
Evaluation

Best results in terms of accuracy of **structural anomaly detection** are achieved using the multimodal LSTM predictions



Single-and multimodal modality LSTM achieves a comparable accuracy, while single- and multimodal dense architectures have low accuracies

Multimodal LSTM achieves a better accuracy than the single-modality LSTM for k=1, otherwise it is similar

Accuracy when the anomaly is injected in traces with different sizes

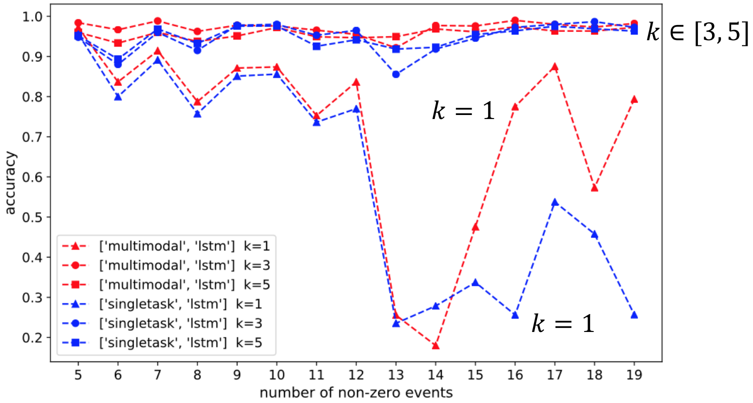


Fig. 6. Comparison of the accuracies of two best models, evaluated for each trace length $4 - 20$ and $k \in \{1, 3, 5\}$.

Multimodal slightly outperforms the single-modality approach in 9 out of 15 trace lengths for *k = 3 and k = 5*

Significantly better results are achieved for *k* = 1 for almost all of the trace lengths

Distributed Trace Analysis
Evaluation

For SAD, significantly better performance of the multimodal approach for k = 1

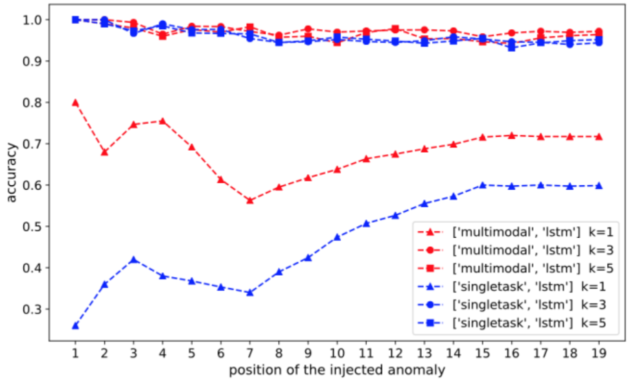


Fig. 7. Comparison of the accuracy of the two best models, evaluated for different positions $0, 20$ of the injected anomaly and $k \in \{1, 3, 5\}$.

Single-modality LSTM achieves a comparable accuracy, while single- and multimodal dense architectures have low accuracies

Multimodal LSTM achieves a better accuracy than the single-modality LSTM for k=1, otherwise it is similar

For RTAD, single-task models have low performance than those of both multimodal models

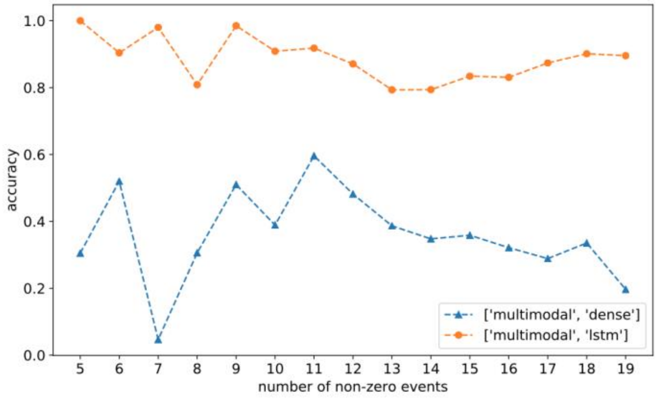


Fig. 8. Response time anomaly detection accuracy comparison of the multimodal LSTM and the baseline deep learning architecture evaluate for different trace lengths.

Multimodal approach achieves a higher accuracy for RTAD

Models have high accuracy even when the length of the trace increases. This is because the LSTMs are able to learn long-term dependencies in sequential tasks

Distributed Trace Analysis
References

- [1] F. Schmidt, A. Gulenko, M. Wallschlger, A. Acker, V. Hennig, F. Liu, and O. Kao, “Iftm - unsupervised anomaly detection for virtualized network function services,” in 2018 IEEE International Conference on Web Services (ICWS), July 2018, pp. 187–194.
- [2] A. Gulenko, F. Schmidt, A. Acker, M. Wallschlager, O. Kao, and F. Liu, “Detecting anomalous behavior of black-box services modeled with distance-based online clustering,” in 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), Jul 2018, pp. 912–915.
- [3] M. Du, F. Li, G. Zheng, and V. Srikumar, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communica- tions Security. ACM, 2017, pp. 1285–1298.
- [4] F. Schmidt, F. Suri-Payer, A. Gulenko, M. Wallschlger, A. Acker, and O. Kao, “Unsupervised anomaly event detection for cloud monitoring using online arima,” in 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Dec 2018, pp. 71–76.
- [16] OpenZipkin, “openzipkin/zipkin,” 2018. [Online]. Available: https://github.com/openzipkin/zipkin
- [19] A. Shrivastwa, S. Sarat, K. Jackson, C. Bunch, E. Sigler, and T. Campbell, OpenStack: Building a Cloud Environment. Packt Publishing, 2016.

Further reading

- S. Nedelkoski, J. Cardoso, O. Kao, Anomaly Detection from System Tracing Data using Multimodal Deep Learning, IEEE Cloud 2019, July 3-8, 2019, Milan, Italy.
- S. Nedelkoski, J. Cardoso, O. Kao, Anomaly Detection and Classification using Distributed Tracing and Deep Learning, CCGrid 2019, 14-17.05, Cyprus.
- J. Cardoso, Mastering AIOps with Deep Learning, Presentation at SRECon18, 29–31 August 2018, Dusseldorf, Germany.

Thank you.

Bring digital to every person, home and organization for a fully connected, intelligent world.