Conference on Innovative Data Systems Research (CIDR)

2019 Asilomar, California

Automated Performance Management for the Big Data Stack

Shivnath Babu et al
Unravel Data

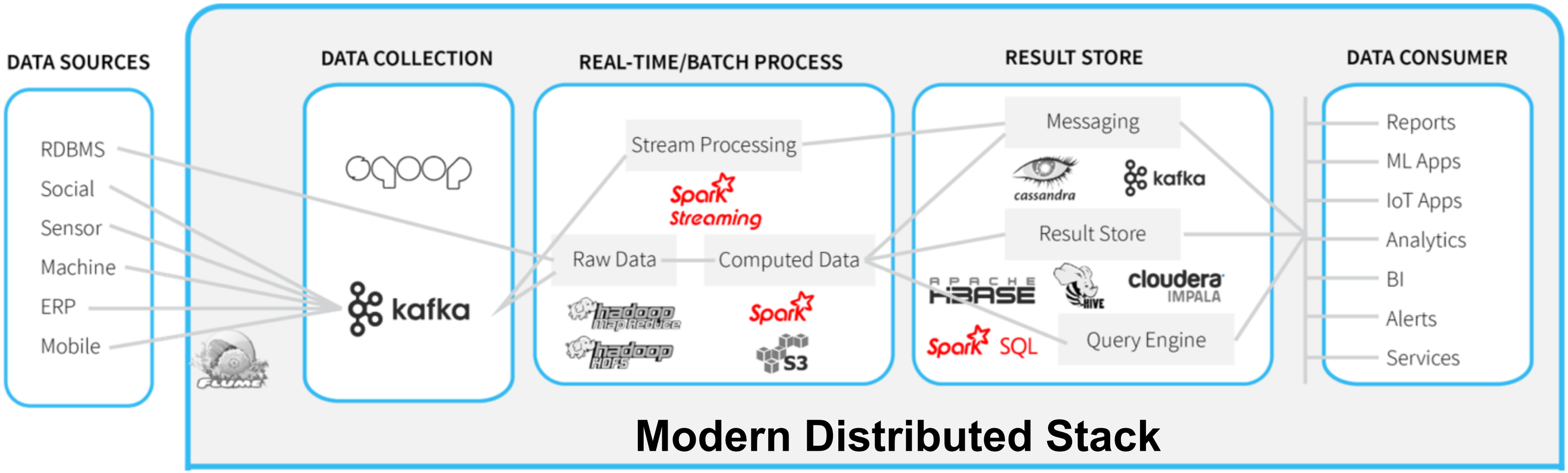
http://cidrdb.org/cidr2019/index.html

Automated Performance Management for the Big Data Stack

Anastasios Arvanitis, Shivnath Babu, Eric Chu,
Adrian Popescu, Alkis Simitsis, Kevin Wilkinson

unravel

Modern applications are being built on a collection of distributed systems



Modern Distributed Stack

But:
Running distributed applications reliably & efficiently is hard

My app failed

My data pipeline is missing SLA

Our cloud costs are out of control

There are many challenges

What enterprises are facing: Monitoring Data is Silo'ed

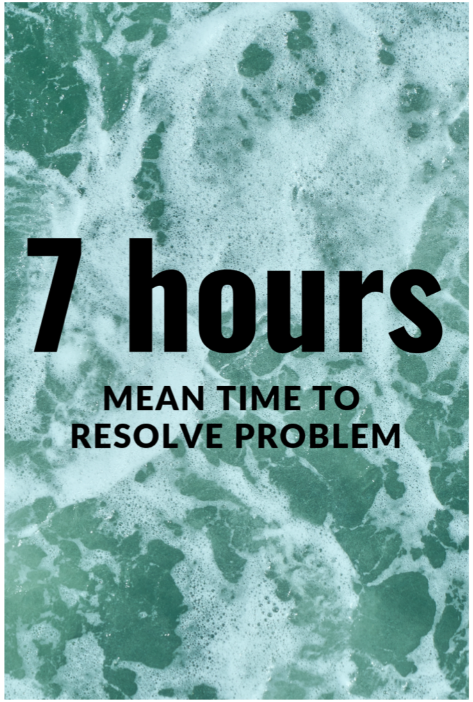
- A survey of 6000+ enterprise IT professionals from Australia, Canada, France, Germany, UK, & USA
  - 91% are struggling with silo'ed monitoring data

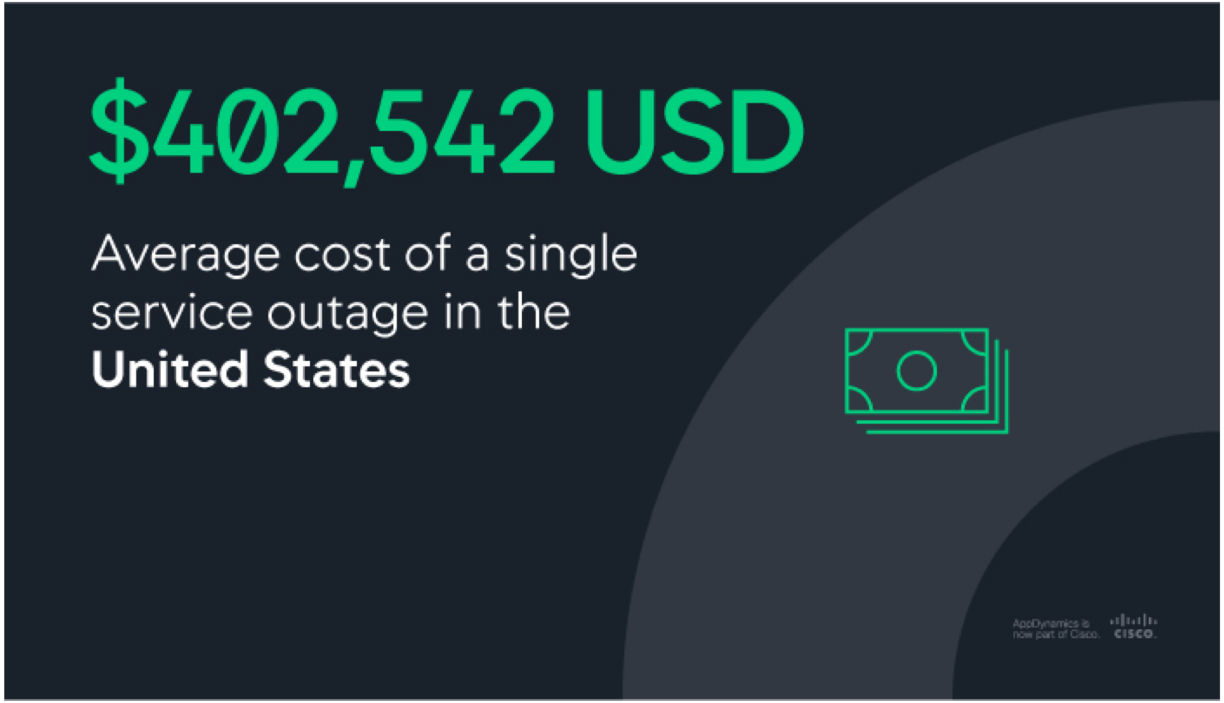
Published: https://blog.appdynamics.com/aiops/aiops-platforms-transform-performance-monitoring/

What enterprises are facing: Reactive Approach

Published: https://blog.appdynamics.com/aiops/aiops-platforms-transform-performance-monitoring/

What enterprises are facing: High MTTR





Published: https://blog.appdynamics.com/aiops/aiops-platforms-transform-performance-monitoring/

We can solve this problem
as a Data and AI/ML problem

First: Bring all monitoring data to a single platform

Resource Manager API

History Server API

Container Metrics

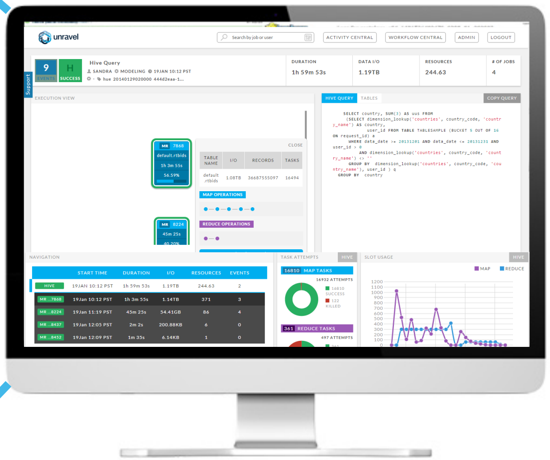
Data Statistics

SQL Query Plans

Logs

Metadata

Configuration



One complete correlated view.

Then: Apply algorithms to analyze the data & (whenever possible) take actions automatically

Resource Manager API

History Server API

Container Metrics

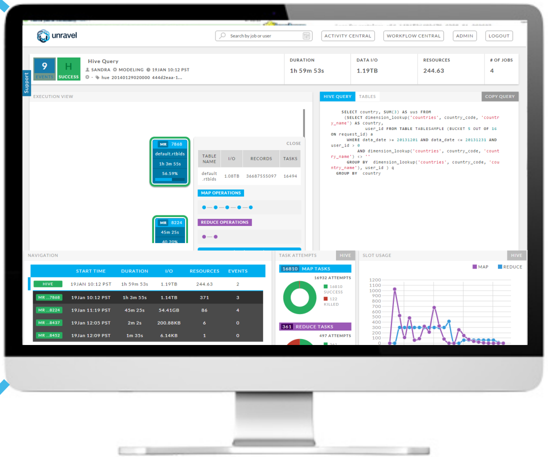
Data Statistics

SQL Query Plans

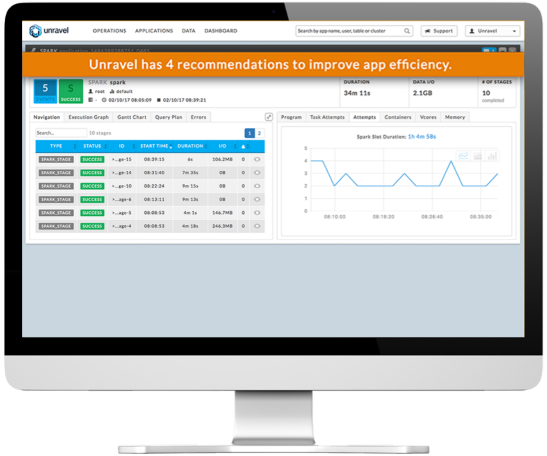
Logs

Metadata

Configuration



One complete correlated view.



Built-in intelligence & automation.

Building this platform requires innovation

- In data collection & transport
  - Non-intrusive, low overhead, transient/elastic clusters
- In data storage
  - Variety, scale, asynchronous arrival

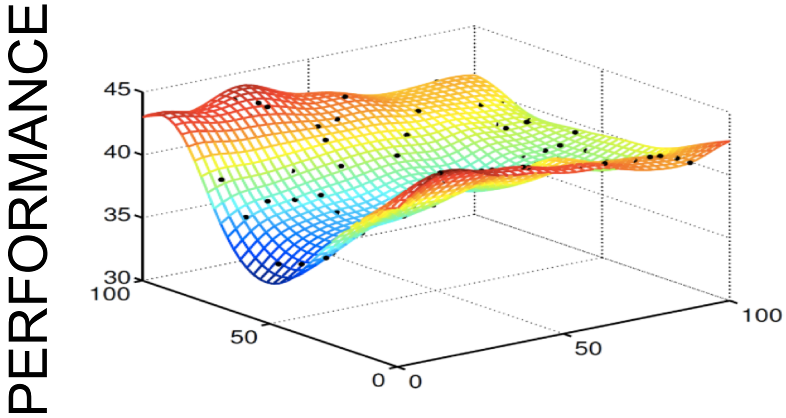
Building this platform requires innovation

- In data collection & transport
  - Non-intrusive, low overhead, transient/elastic clusters
- In data storage
  - Variety, scale, asynchronous arrival
- In algorithms to provide insights
  - Real-time, combine expert knowledge with ML
- In algorithms to take actions
  - Reliable, predictable

Example problems for which our solutions are running in production enterprise environments

- Application autotuning
- Failures of distributed applications
- SLA management for streaming data pipelines
- Holistic cluster optimization

spark.driver.cores	2
spark.executor.cores **…**	10
spark.sql.shuffle.partitions	300
spark.sql.autoBroadcastJoinThreshold	20MB
…	
SKEW('orders', 'o_custId')	true
spark.catalog.cacheTable(“orders")	true
…	



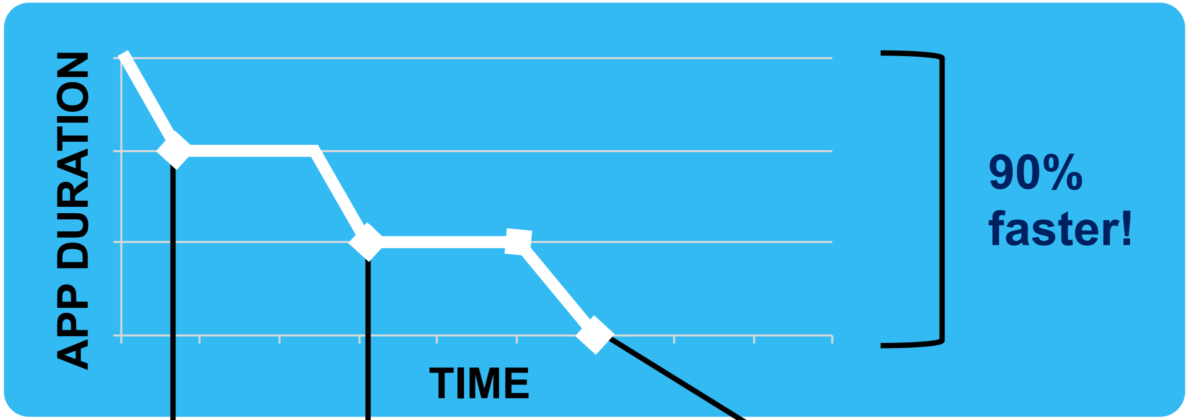
Today, tuning is often by trial-and-error

A new world

INPUTS
1. **App = Spark Query**
2. **Goal = Speedup**

"I need to make this app faster"

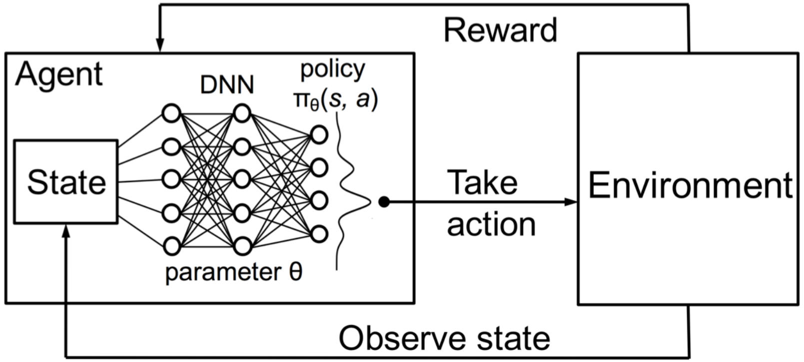
A new world



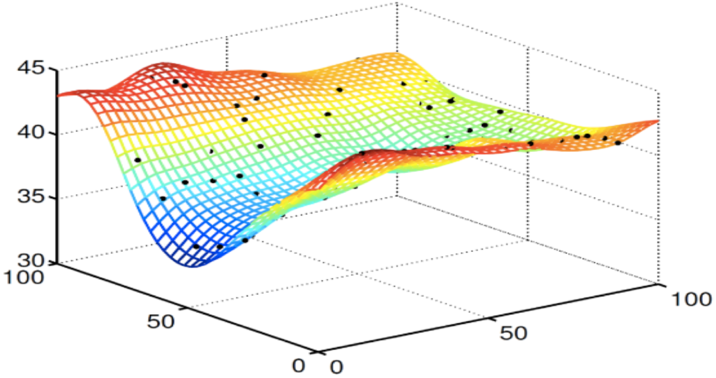
In blink of an eye, user gets recommendations to make the app **30% faster**

As user finishes checking email, she has a verified run that is **60% faster**

User comes back from lunch. A verified run that is **90% faster**



Reinforcement Learning



Response Surface Methodology

Tuning Database Configuration Parameters with iTuned

Songyun Duan, Vamsidhar Thummala, Shivnath Babu*
Department of Computer Science
Duke University
Durham, North Carolina, USA
{syduan,vamsi,shivnath}@cs.duke.edu

ABSTRACT

Database systems have a large number of configuration parameters that control memory distribution, I/O optimization, costing of query plans, parallelism, many aspects of logging, recovery, and

Amy recalls that the database has *configuration parameters*. For lack of better understanding, she had set them to default values during installation. The parameters may need tuning, so Amy pulls out the 1000+ page database tuning manual. She finds many dozens

Xplus: A SQL-Tuning-Aware Query Optimizer

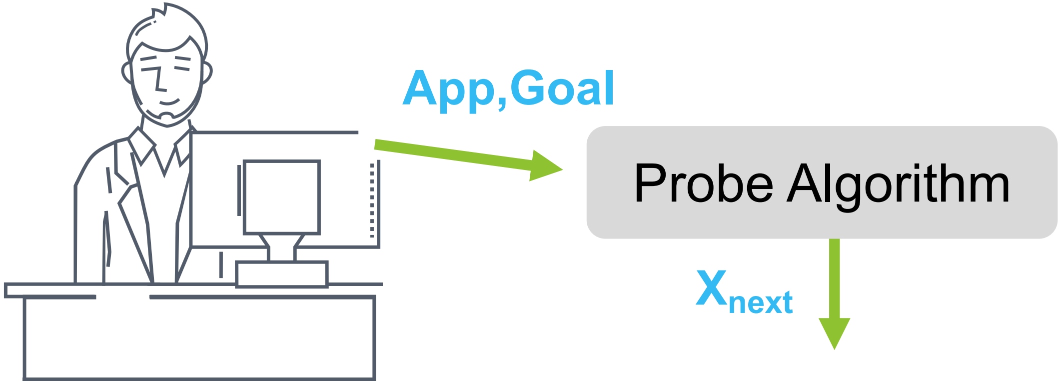
Herodotos Herodotou and Shivnath Babu*
Department of Computer Science
Duke University
{hero,shivnath}@cs.duke.edu

ABSTRACT

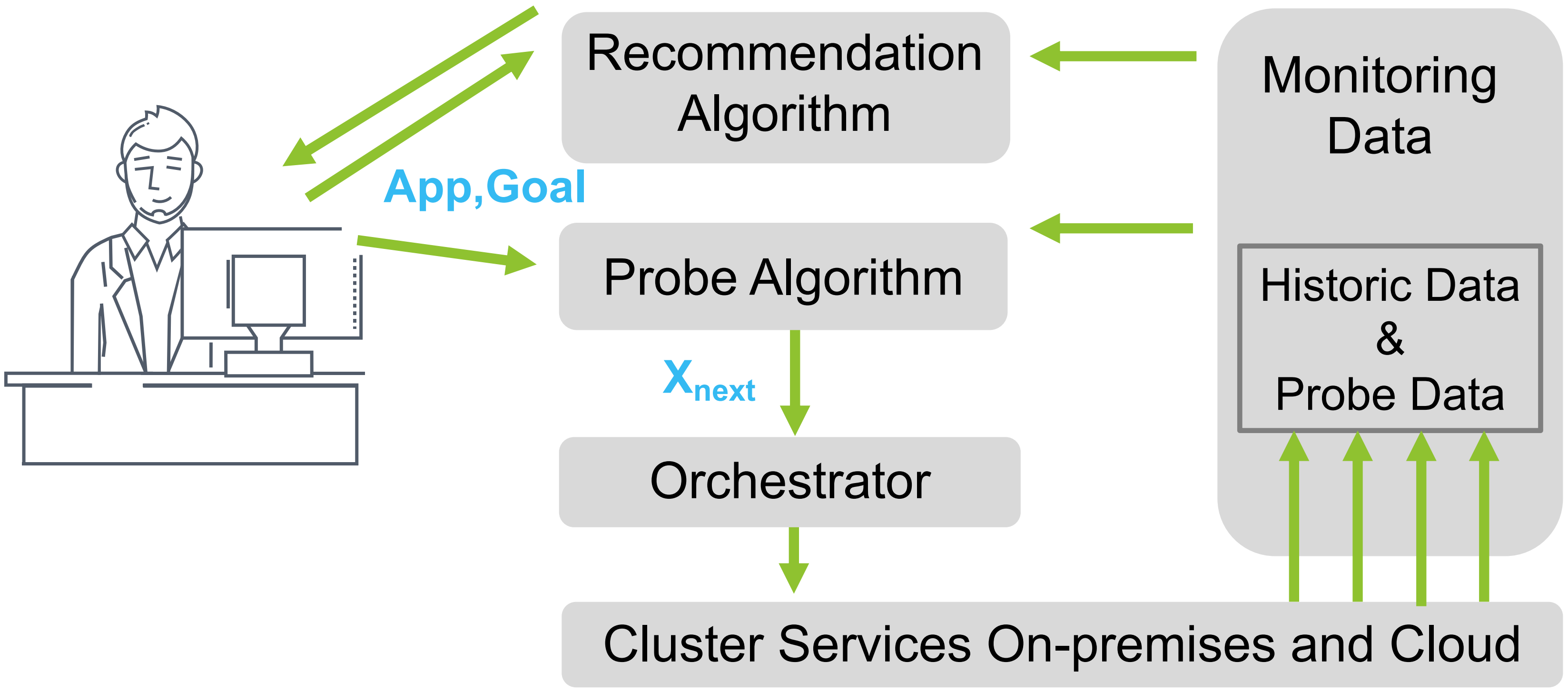
The need to improve a suboptimal execution plan picked by the query optimizer for a repeatedly run SQL query arises routinely. Complex expressions, skewed or correlated data, and changing con-

step in to lead the optimizer towards a good plan [6]. This process of improving the performance of a "problem query" is referred to in the database industry as *SQL tuning*. Tuning a problem query is critical in two settings:

Autotuning workflow



Autotuning workflow



Example problems for which solutions are running in production enterprise environments

- Application autotuning
- **Failures of distributed applications**
- SLA management for streaming data pipelines
- Holistic cluster optimization

Manual Root Cause Analysis of App Failures

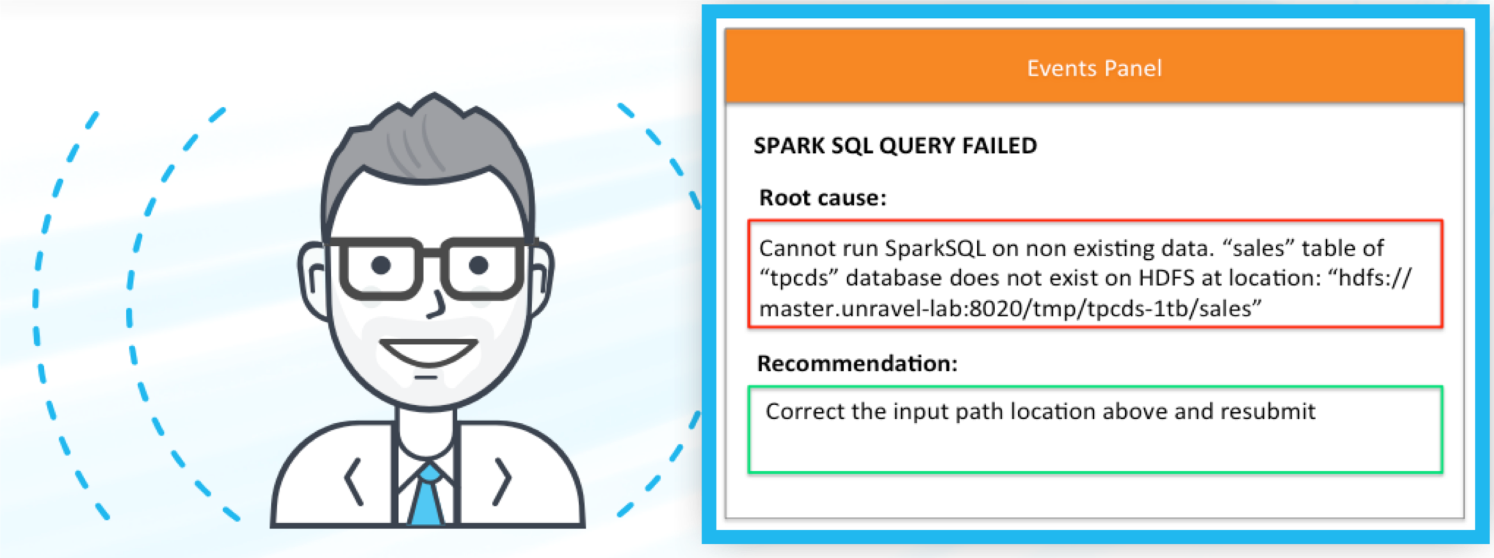
```
e.spark.sql.execution.ConvertToSafe.doExecute(rowFormatConverters.scala:56)
e.spark.sql.execution.SparkPlan$$anonfun$execute$5.apply(SparkPlan.scala:132)
e.spark.sql.execution.SparkPlan$$anonfun$execute$5.apply(SparkPlan.scala:130)
e.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:150)
e.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:130)
e.spark.sql.execution.Exchange.prepareShuffleDependency(Exchange.scala:164)
e.spark.sql.execution.Exchange$$anonfun$doExecute$1.apply(Exchange.scala:254)
e.spark.sql.execution.Exchange$$anonfun$doExecute$1.apply(Exchange.scala:248)
e.spark.sql.catalyst.errors.package$.attachTree(package.scala:48)

e.spark.sql.catalyst.errors.package$TreeNodeException: execute, tree:
(key=[nation#71,o_year#72], functions=[(sum(amount#73),mode=Partial,isDistinct=false)], output=[nation#71,o_year#72,sum#78])
e#50 AS nation#71,year(cast(o_orderdate#20 as date)) AS o_year#72,((l_extendedprice#5 * (1.0 - l_discount#6)) - (ps_supplycost#28 * l_quantity#4)) AS

in [o_orderkey#16L], [l_orderkey#0L]
orderkey#16L ASC], false, 0
tenExchange hashpartitioning(o_orderkey#16L,200), None
oject [o_orderkey#16L,o_orderdate#20]
 Scan ExistingRDD[o_orderkey#16L,o_custkey#17L,o_orderstatus#18,o_totalprice#19,o_orderdate#20,o_orderpriority#21,o_clerk#22,o_shippriority#23L,o_comm
orderkey#0L ASC], false, 0
tenExchange hashpartitioning(l_orderkey#0L,200), None
oject [l_extendedprice#5,l_discount#6,l_quantity#4,l_orderkey#0L,n_name#50,ps_supplycost#28]
 SortMergeJoin [p_partkey#30L], [l_partkey#1L]
 :- Sort [p_partkey#30L ASC], false, 0
 :  +- TungstenExchange hashpartitioning(p_partkey#30L,200), None
 :     +- Project [p_partkey#30L]
 :        +- Filter Contains(p_name#31, ghost)
 :           +- Scan ExistingRDD[p_partkey#30L,p_name#31,p_mfgr#32,p_brand#33,p_type#34,p_size#35L,p_container#36,p_retailprice#37,p_comment#38]
 +- Sort [l_partkey#1L ASC], false, 0
    +- TungstenExchange hashpartitioning(l_partkey#1L,200), None
       +- Project [l_extendedprice#5,l_discount#6,l_quantity#4,l_partkey#1L,l_orderkey#0L,n_name#50,ps_supplycost#28]
          +- SortMergeJoin [ps_suppkey#26L,ps_partkey#25L], [l_suppkey#2L,l_partkey#1L]
             :- Sort [ps_suppkey#26L ASC,ps_partkey#25L ASC], false, 0
             :  +- TungstenExchange hashpartitioning(ps_suppkey#26L,ps_partkey#25L,200), None
```

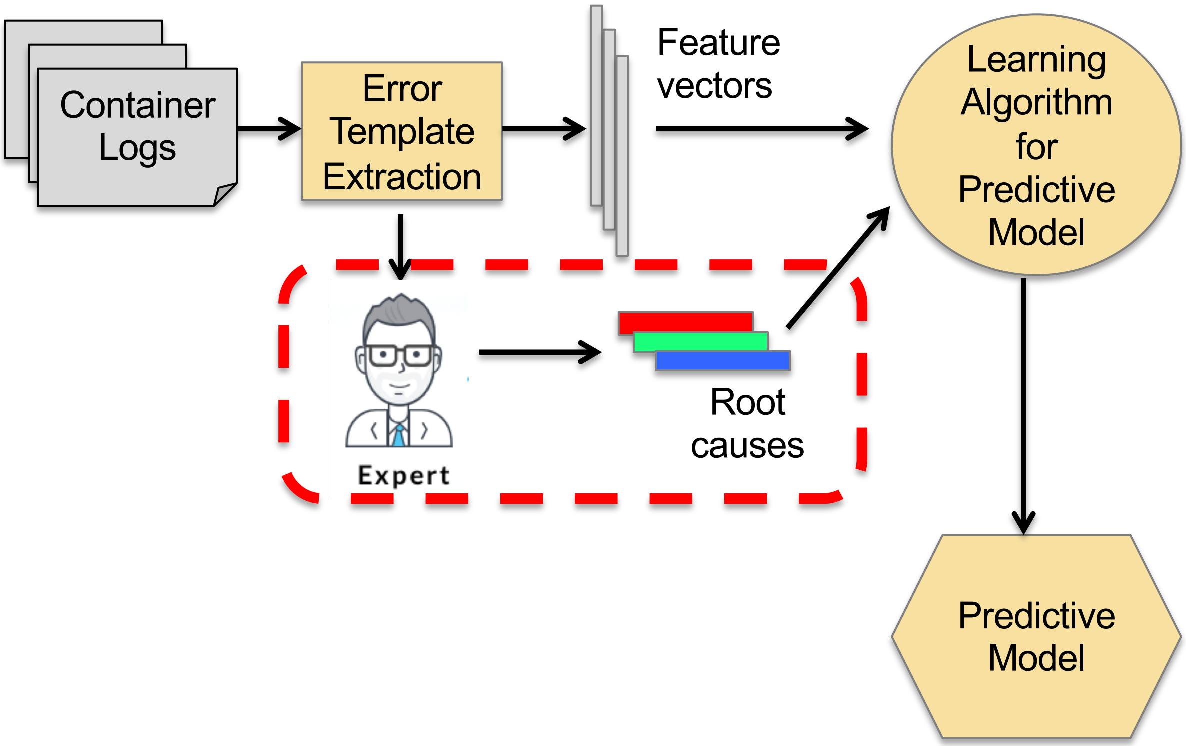

- Many levels of correlated stack traces
- Identifying the root cause is hard and time consuming

Automated Root Cause Analysis of Failures



- **Reduce troubleshooting time from days to seconds**
- Improve productivity of data scientists and analysts

Automated Root Cause Analysis of Failures



Two Ways to get Root-Cause Labels

- Manual diagnosis by a domain expert
- Automatic injection of the root cause
    - Invalid input
    - Invalid memory configuration
    - OOME: Java heap space
    - OOME: GC overhead limit
    - Container killed by YARN
    - Runtime incompatibility
    - No space left on device
    - Transformations inside other transformations
    - Runtime error
    - Arithmetic error
    - Invalid configuration settings

Large-scale Lab Framework for Automatic Root Cause Analysis

Environment:
- Lab created on demand on cloud or on-premises
- Workloads are run and failures are injected

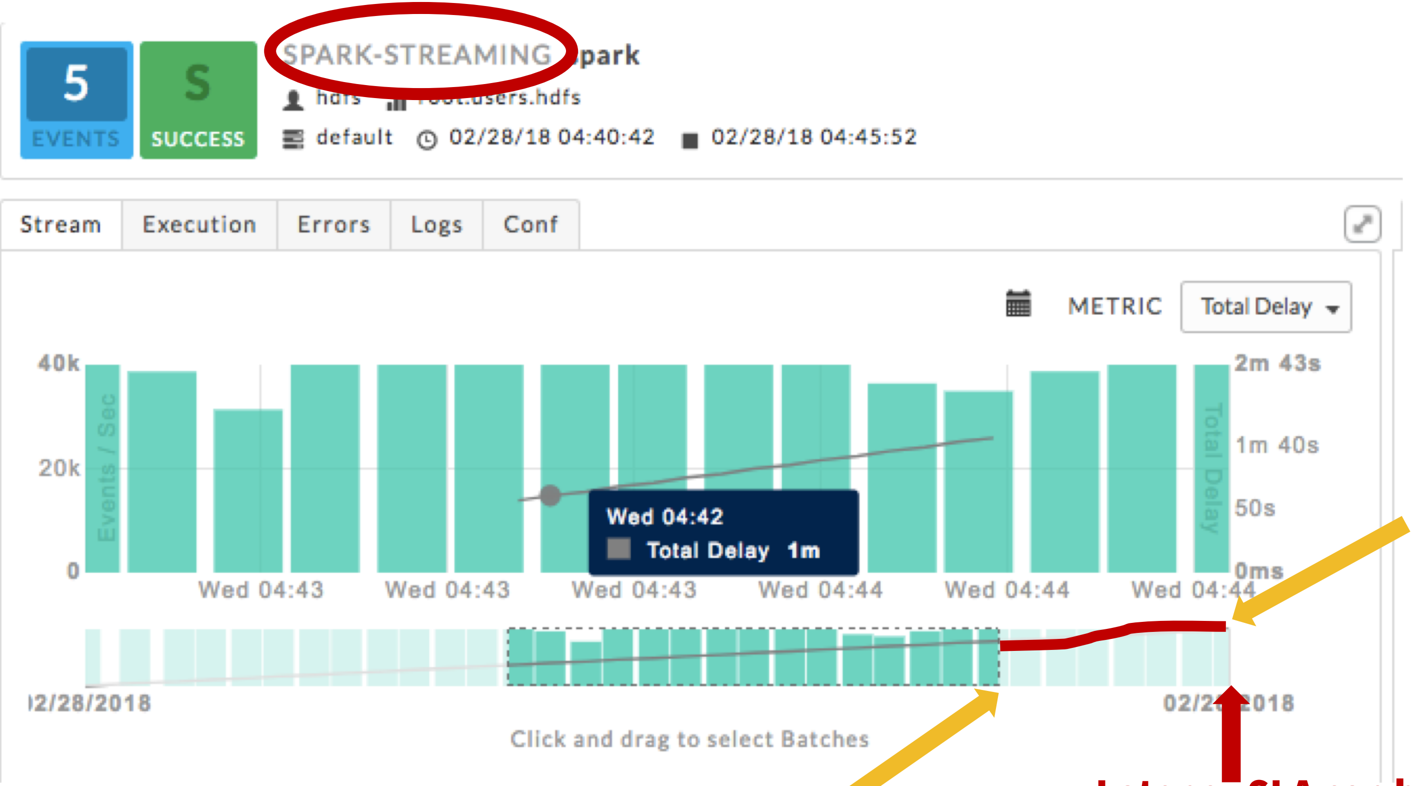
Multi-tenant Workloads:
- Variety of workloads: Batch, ML, SQL, Streaming, etc.

Failures:
- Large set of root causes learned from customers & partners. Constantly updated
- Continuously inject these root causes to train & test models for root-cause prediction

Example problems for which solutions are running in production enterprise environments

- Application autotuning
- Failures of distributed applications
- **SLA management for streaming data pipelines**
- Holistic cluster optimization

Predicting when SLAs are in danger of being missed



Latency SLA is 3 minutes

Current time is here

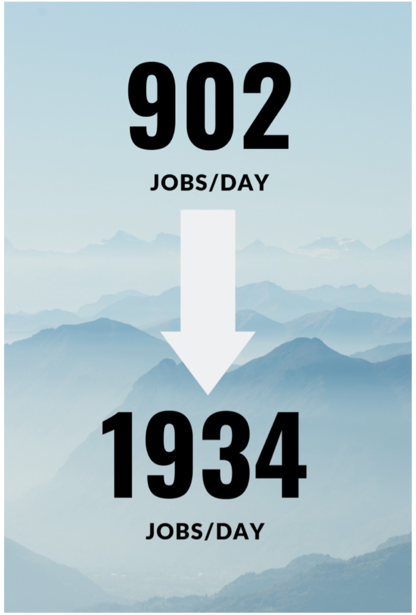
Latency SLA can be missed by this time

Forecasting & Anomaly Detection are very useful to manage streaming data pipelines

Example problems for which solutions are running in production enterprise environments

- Application autotuning
- Failures of distributed applications
- SLA management for real-time data pipelines
- **Holistic cluster optimization**

Holistic Cluster Optimization

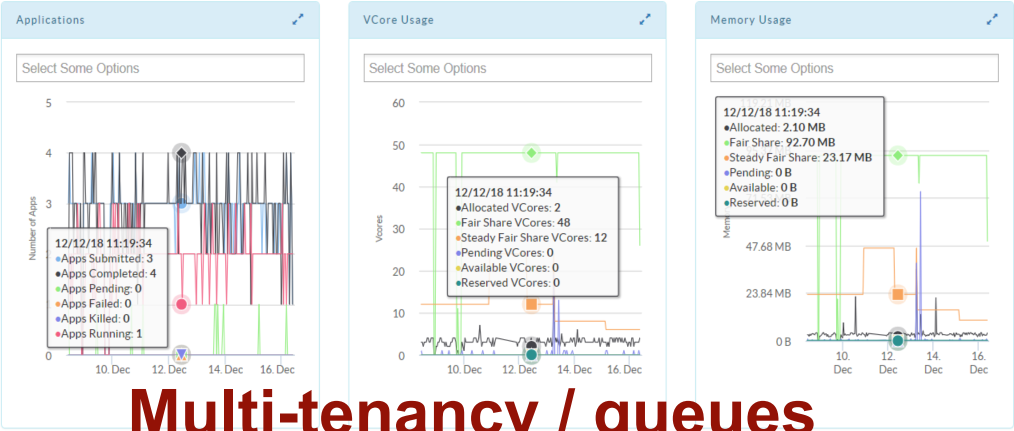


2x throughput increase and 2x reduction in cost!

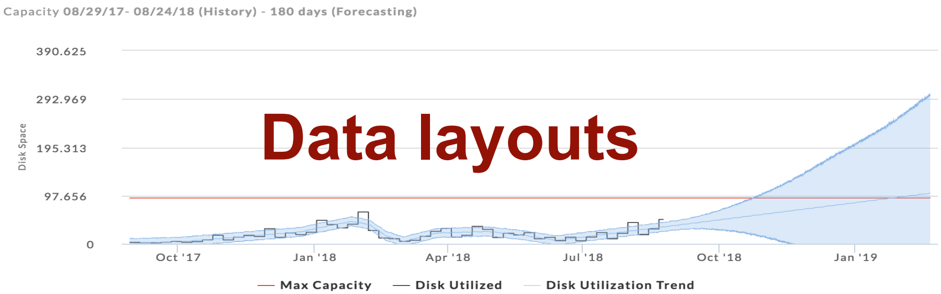
Holistic Cluster Optimization

Multi-tenancy / queues

Holistic Cluster Optimization

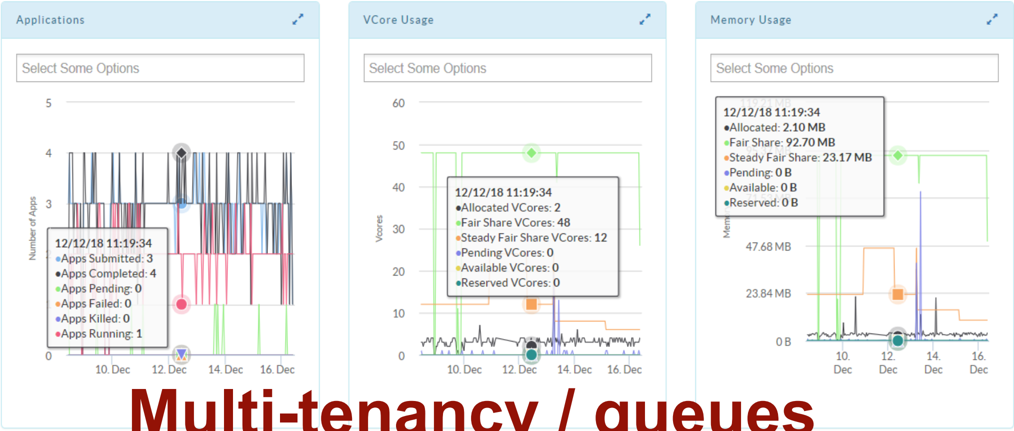


Multi-tenancy / queues

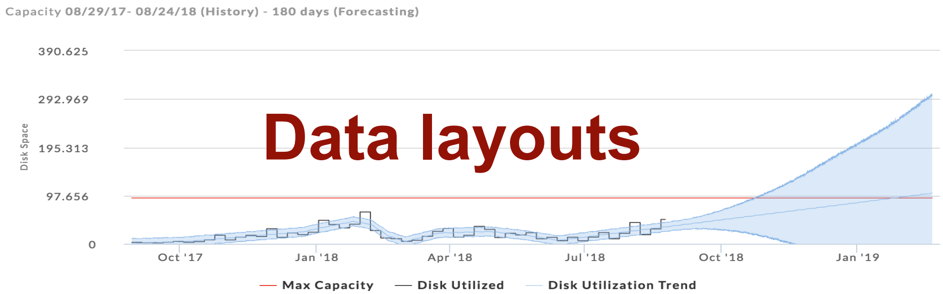


Data layouts

Holistic Cluster Optimization



Multi-tenancy / queues



Data layouts

Tune the size of map containers

mapreduce.map.memory.mb

Recommendation: **2048**
Improvement Potential: **High**
Using the recommended default would save 50% of 1,263,087,616 MB from 7275 jobs using the current default, or 97% of total jobs in the workload

Tuning Instructions

Default: **4096**

% Apps Default: **97%**

Candidate	% of memor...	% of jobs ...
1792	56%	80%
2048	50%	85%

Tune the size of reduce containers

mapreduce.reduce.memory.mb

Configuration

Recommendation: **2816**
Improvement Potential: **High**
Using the recommended default would save 65% of 1,973,538,816 MB from 4060 jobs using the current default, or 97% of total jobs in the workload

Tuning Instructions

Default: **8192**

% Apps Default: **97%**

Candidate	% of memor...	% of jobs ...
2816	65%	99%
3072	62%	99%

In Summary

AIOps: Rich opportunities to address distributed application performance management as AI/ML problems

We welcome your collaboration!

Thank You

Stay informed
https://unraveldata.com/blog/

Free Fully Featured Trial on Amazon EMR, Microsoft Azure, On-premises

https://unraveldata.com/free-trial/