unravel

TECHNICAL WHITEPAPER

AI-POWERED DATA OPERATIONS FOR MODERN DATA APPLICATIONS

Operationalizing big data to optimize and automate modern applications and infrastructure.

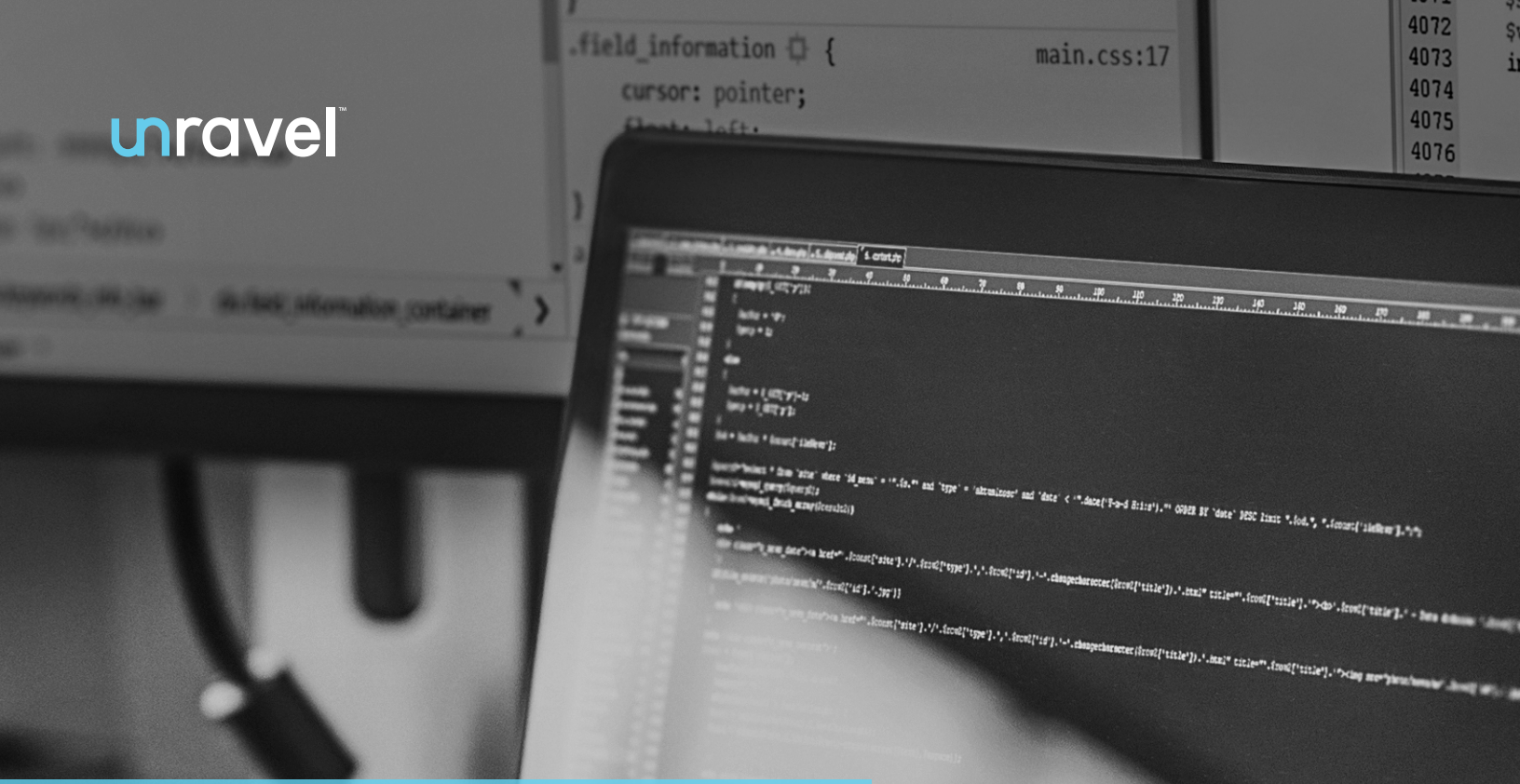


TABLE OF CONTENTS

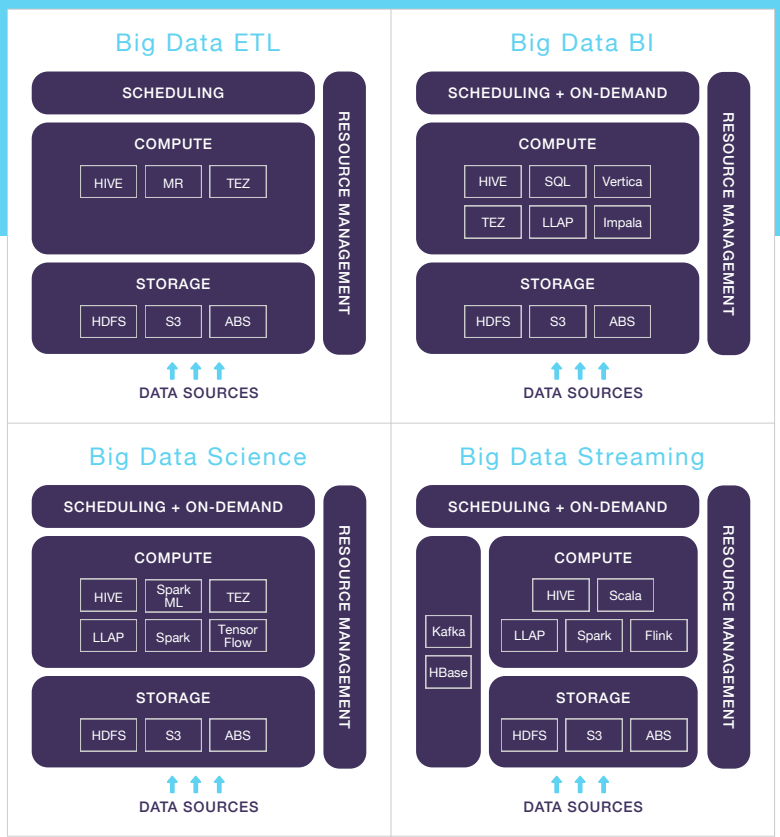
OPERATIONALIZING BIG DATA TO OPTIMIZE AND AUTOMATE MODERN APPLICATIONS AND INFRASTRUCTURE

Today, more than 10,000 enterprise businesses worldwide use a complex stack composed of a combination of distributed systems like Spark, Kafka, Hadoop, NoSQL databases, and SQL access technologies. At Unravel, we have worked with many of these businesses across all major industries. These customers are deploying modern data applications in their data centers, in private cloud deployments, in public cloud deployments, and in hybrid combinations of these.

This paper addresses the requirements that arise in driving reliable performance in these complex environments. We provide an overview of these requirements both at the level of individual applications as well as in holistic clusters and workloads. We also present a platform that can deliver automated solutions to address these requirements as well as taking a deeper dive into a few of these solutions.

THE MODERN DATA STACK

Many applications – in fields as diverse as healthcare, genomics, financial services, self-driving technology, government, and media – are being built on what is popularly known today as the modern data stack. What is unique about the modern data stack is that it is composed of multiple distributed systems. The typical evolution of this stack usually includes the following stages:



Application Performance Requirements

The nature of distributed applications is such that they interact with many different components, both independent and interdependent. In popular literature, this is often referred to as "having many moving parts." In such an environment, questions like these become important to answer:

Failure Detection

What caused this application to fail, and how can I fix It?

Application Non-Responsiveness

This application seems to have made little progress in the past hour. Where is it stuck?

Runaway Processes

Will this application ever finish? Will this application finish in a reasonable time?

Service-Level Violation

Will this application meet its SLA?

Changing Runtime Behavior

Is the behavior (e.g., performance, resource usage) of this application very different from past behavior? If so, in what way, and why?

Rogue and Victim Applications

Is this application throttling the performance of other applications on my cluster? Or, just the opposite, is the performance of this application being affected by one or more of my other applications?

Keep in mind that almost every modern data application interacts with multiple distributed systems. For example, an SQL query may interact with Spark for its computational aspects, with YARN for its resource allocation and scheduling, and with HDFS or S3 for its data access and IO. Or a streaming application may interact with Kafka, Flink, and HBase.

Big Data ETL

ETL in the big data context is the process of organizing and providing structure to incoming data to run queries and analytics against it. Storage systems like HDFS, S3, and Azure Blob Storage (ABS) are used to store the large volumes of structured, semi-structured, and unstructured data in the enterprise. Distributed processing engines like MapReduce, Tez, and Pig/Hive (usually running on MapReduce or Tez) are used for data extraction, cleaning, and data transformation.

Big Data BI

Big data programs often stem from the needs and activities of BI teams and activities in the enterprise. For big data BI, massively parallel processing (MPP) SQL systems like Impala, Presto, LLAP, Drill, BigQuery, RedShift, or Azure SQL DW are added to the stack; sometimes alongside incumbent MPP SQL systems like Teradata and Vertica. Compared to traditional MPP systems, newer solutions have been created to deal with data stored in different distributed storage systems like HDFS, Amazon S3, and Microsoft Azure Blob Storage (ABS). These systems power the interactive SQL queries common to business intelligence workloads.

Big Data Science

As enterprise businesses mature in their use of the modern data stack, they often introduce new data-science workloads leveraging machine learning (ML) and artificial intelligence (AI). This is usually the point when the Spark distributed system starts to be used more often.

Operational Performance Requirements

Many performance requirements also arise at the "macro" level compared to the level of individual applications. Examples of such requirements are:

Resource Allocation

Configuring resource allocation policies to meet SLAs in multi-tenant clusters

Rogue Application Detection

Detecting rogue applications that can affect the performance of SLA-bound applications through a variety of low-level resource interactions

Configuration

Configuring the hundreds of settings distributed systems are notoriously known for, to drive optimal performance

Partitioning and Storage

Tuning data partitioning and storage layout for optimal throughput

Cost Optimization

Optimizing dollar costs on the cloud. All types of resource usage on the cloud cost money. For example, picking the right node type for a cloud cluster can have a major impact on the overall cost of running a workload.

Capacity Planning and Forecasting

Capacity planning using predictive analysis to proactively address workload growth

Big Data Streaming

Over time, enterprises begin to understand the importance of making data-driven decisions in real time as well as how to overcome the challenges in implementing them. Usually at this point in the evolution, systems like Kafka, Cassandra, and HBase are added to the big data stack to support applications that ingest and process data in a continuous stream.

At Unravel, we have worked closely with more than 50 businesses at this stage in their evolution, and have had detailed conversations with more than 350 others, covering a wide array of industries and deployment models, with cluster size varying from tens to thousands of nodes. In some of these cloud-based production deployments, the size of an auto-scaling cluster can vary in size from 1 node to 1,000 nodes in under 10 minutes.

The goal of this paper is to define the data operations and performance requirements that arise in modern data stacks and how Unravel addresses these needs. We split these needs into two main categories: application performance requirements and operational performance requirements.

ARCHITECTURE OF A PERFORMANCE MANAGEMENT SOLUTION FOR BIG DATA

Addressing the challenges of big data performance management requires an architecture like the one shown below:

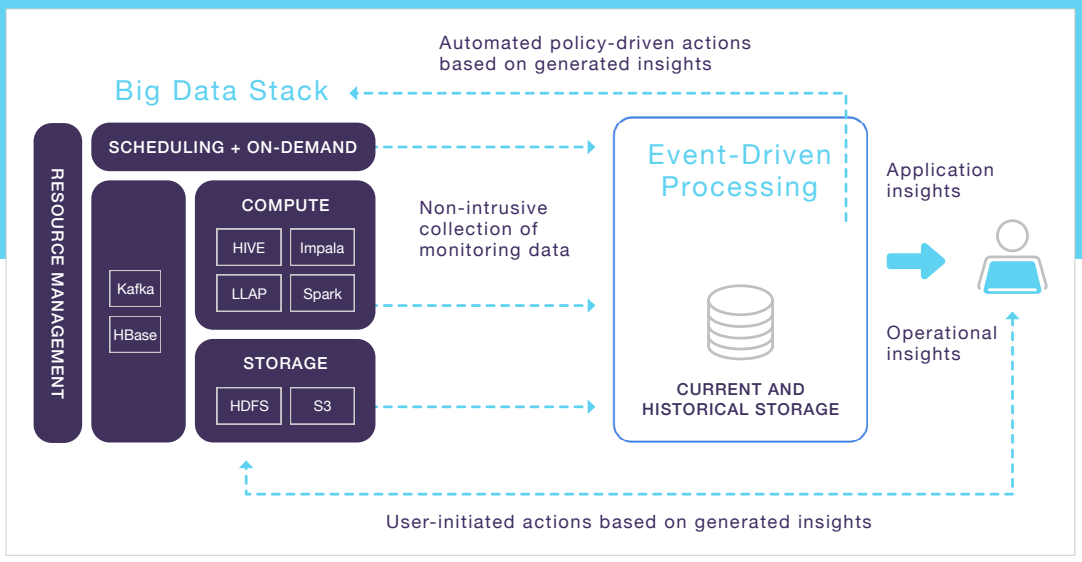


Figure 2: Architecture of a Data Operations Platform for Modern Data Applications and Pipelines

Full-Stack Data Collection

Addressing the challenges above requires a modern, data-centric architecture with data collection from all components in the data pipeline:

- SQL queries, execution plans, data pipeline dependency graphs, and application logs
- Resource allocation and wait-time metrics from resource managers and job schedulers
- Actual CPU, memory, and network usage metrics from the cloud infrastructure
- Data access and storage metrics from the file-system and storage

Collecting such data in non-intrusive and low-overhead ways from production clusters remains a major technical challenge, but Unravel has addressed this issue through the use of native platform APIs (Hadoop, Spark, Kafka, etc.) and lightweight micro-sensors instead of the use of more resource-intensive software agents.

Event-Driven Data Processing

Unravel develops and tests on clusters of more than 500 nodes in size, running hundreds or thousands of applications every day across ETL, BI, data science, and streaming. This presents the familiar big data challenges to event processing of volume, velocity, and variety, and adds the additional challenge of consistency

Volume: These deployments generate tens of terabytes of logs and metrics every day.

Velocity: real-time runtime data from thousands of nodes and hundreds of apps

Variety: full spectrum from unstructured logs to semi-structured data pipeline dependency DAGs and to structured time-series metrics.

Consistency: The distributed nature of data collection means no assumptions can be made about the timeliness or order in which the monitoring data arrives.

As a result, the data processing later has to be based on event-driven processing algorithms whose outputs converge to the same state irrespective of the timeliness and order in which the monitoring data arrives. From the user's perspective, they should get the same insights no matter what – however and whenever data is introduced.

ML-Driven Insights and Policy-Driven Actions

Enabling all the monitoring data to be collected and stored in a single place opens up interesting opportunities to apply statistical analysis and learning algorithms to this data. These algorithms can generate insights that can be applied manually or automatically based on configured policies to address the performance requirements identified above.

THE UNRAVEL SOLUTION

Unravel has created an AI/ML-driven solution based on the architecture described above. The Unravel platform addresses performance tuning, troubleshooting, and resource management at both the application and operations levels.

Unravel goes beyond the capabilities of systems monitoring tools of the big data stake to provide predictive capabilities around forecasting and capacity planning, as well as accounting and governance issues around chargeback/showback.

Addressing Application Failure

In distributed systems, applications can fail for many reasons. But when an application fails, users are required to address the cause of the failure to get the application back up and running quickly and successfully. Since applications in distributed systems interact with multiple components, a failed application can generate a large collection of raw log data. These logs typically contain thousands of messages, including errors and stacktraces.

Hunting down the root cause of an application failure in these messy, raw, distributed logs is hard enough for data experts – for the thousands of new users trying to find their way around the modern data stack, it's a nightmare. This is where Unravel can help.

Unravel automatically generates insights into what's causing application failures – even in multi-engine ecosystems – providing recommendations and even policy-driven autonomous actions to help get the application running successfully again.

Automatic Identification of the Root Cause of Application Failure

Unravel continually collects the large and growing data set of logs from millions of application failures from Spark, Impala, Tez, MapReduce, and Hive, with additional engines to be added in the future. This data set is a gold mine for applying state-of-the-art artificial intelligence (AI) and machine learning (ML) techniques.

Unravel has developed ways to automate the process of failure diagnosis by building predictive models that continuously learn from logs of past application failures for which the respective root causes have been identified. These models can then automatically predict the root cause when an application fails. Such actionable root-cause identification improves the productivity of big data developers significantly.

Root Cause Analysis for Spark Applications

Taking Spark as an example, a distributed Spark application consists of a Driver container and one or more Executor containers. A number of logs are available every time a Spark application fails. But the logs are extremely verbose and messy. They contain multiple types of messages, such as informational messages from every component of Spark, error messages in many different formats, stack traces from code running on the Java Virtual Machine (JVM), and more.

The complexity of Spark usage and internals makes things worse. Types of failures and error messages differ across Spark SQL, Spark Streaming, iterative machine learning and graph applications, and interactive applications from Spark shell and notebooks (e.g., Jupyter, Zeppelin).

Furthermore, failures in distributed systems routinely propagate from one component to another. Such propagation can cause a flood of error messages in the log and obscure the root cause.

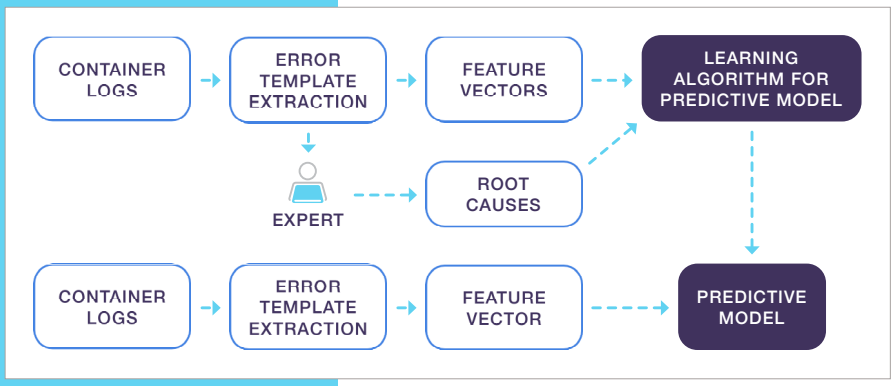


Figure 3: Approach for Automatic Root Cause Analysis (RCA) of Spark Applications

Figure 3 shows Unravel's unique approach for dealing with these challenges, automating Root Cause Analysis (RCA) for Spark application failures.

Overall, the solution involves:

- Continuously collecting logs across a variety of Spark application failures
- Converting these logs into feature vectors
- Developing a predictive model for RCA from these feature vectors
- Data collection for training. It is critical to train RCA models on representative input data.

In addition to relying on logs from real-life Spark application failures observed on customer sites, we have also invested in a lab framework where root causes can be artificially injected to collect even larger and more diverse training data.

Structured Versus Unstructured Data

Logs are mostly made up of unstructured data. To ensure a high level of accuracy in the model predictions used to perform automated RCA, it is important to combine this unstructured data with some structured data. Thus, whenever we collect logs, we are careful to collect trustworthy structured data in the form of key-value pairs that we also use as input in our predictive models. This includes Spark platform information and environmental details from Scala, Hadoop, OS, and so on.

Attaching Root-Cause Labels

Machine learning predictive techniques fall into two broad categories: supervised and unsupervised learning. At Unravel, we use both techniques. For supervised learning, we attach root-cause labels to the logs collected from application failures. These labels are derived from a taxonomy of root causes we've developed based on millions of Spark application failures seen both in the field and in our lab.

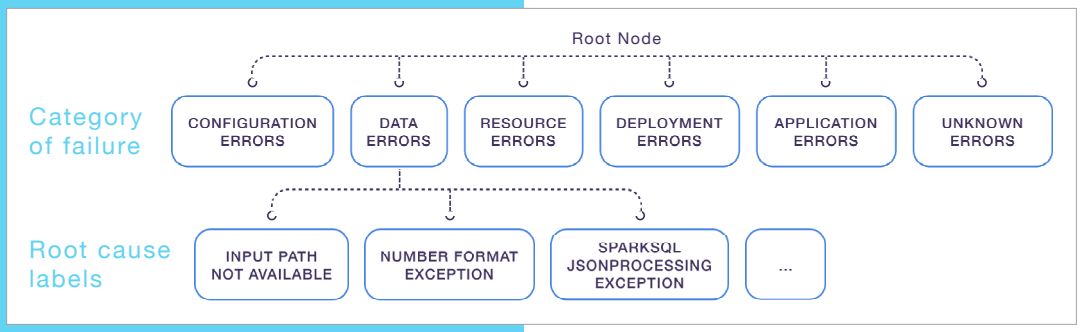


Figure 4: Taxonomy of Application Failures

Broadly speaking, the taxonomy (as shown in Figure 4) can be thought of as a tree data structure that categorizes the full space of root causes. For example, the first non-root level of this tree includes failures caused by:

- Configuration errors
- Deployment errors
- Resource errors
- Data errors
- Application errors
- Unknown factors

The leaves of the taxonomy tree form the basis of the labels used in our supervised learning techniques. In addition to a text label representing the root cause, each leaf also stores additional information, such as a description template that presents that root cause in such a way that users will easily understand the recommended fixes for that issue.

These labels are associated with logs in one of two ways: In the first, the root cause is already known when the logs are generated, because we introduced a specific root cause in our lab framework designed to produce an application failure. In the second, a label for an application failure is added to the logs by a Spark domain expert who manually diagnoses the root cause of the failure.

Input Features

Once the logs are available, there are various ways in which the feature vector can be extracted from these logs. (Recall Unravel's overall approach in Figure 3.) One way is to transform the logs into a bit vector (e.g., 1001100001). Each bit in this vector represents whether a specific message template is present in the respective logs. A prerequisite to this approach is to extract all possible message templates from the logs. A more traditional approach for deriving feature vectors is to represent the logs for a failure as a bag of words. This method is similar to the bit-vector approach except for two key differences:

- Each bit in the vector now corresponds to a word instead of a message template.
- Instead of 0s and 1s, it is more common to use numeric values generated with techniques like TF-IDF.

More recent advances in machine learning have popularized vector embeddings – in particular, at Unravel, we use the Doc2Vec technique. At a high level, these embeddings map words (or paragraphs, or entire documents) to multidimensional vectors by evaluating the order and placement of words with respect to their neighboring words. Similar words map to nearby vectors in the feature vector space. The Doc2Vec technique uses a three-layer neural network to gauge the context of the document and relate similar content together.

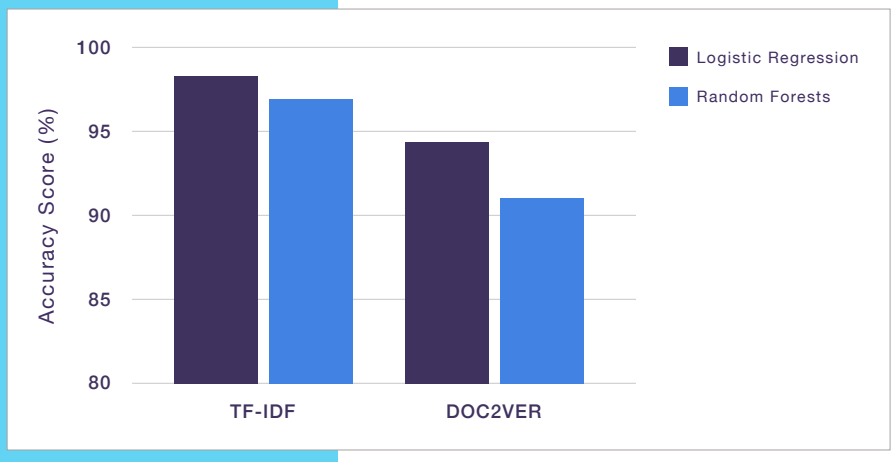


Figure 5: Feature Vector Generation

Once the feature vectors are generated along with a corresponding label, automatic RCA can be performed using a variety of supervised learning techniques. We have evaluated both shallow and deep learning techniques, including random forests, support vector machines, Bayesian classifiers, and neural networks. The overall results produced by the Unravel platform are promising – as shown in Figure 5.

In Figure 5, 14 different types of root causes were introduced into runs of various Spark applications to collect a large set of logs. The figure shows the accuracy of Unravel's unique approach in Figure 3 to predicting the correct root cause based on a 75/25 split of training and test data. The accuracy of prediction is fairly high.

At the moment, we are enhancing our solution in some key ways. One of these is to quantify the degree of confidence in the predicted root cause in a way that users will easily understand. Another key enhancement is to speed our ability to incorporate new types of application failures.

Automatic Fixes for Failed Applications

We also performed a deeper analysis of the Spark application failure logs available to us from more than 20 large-scale production clusters. The key finding from our analysis is that there is a 90/10 rule in the root cause of app failures – that is, in all of these clusters, more than 90% of failures were caused by less than 10 unique root causes.

The 2 most common causes were:

- The application failing due to running out of memory (OOM) in some component
- The application failing due to timeout while waiting for some resource

For application failures caused by OOM, we designed algorithms that – in addition to leveraging historical examples of failed and successful applications runs – can intelligently try a limited number of memory configurations to get the application quickly back to a running state, followed by a resource-efficient running state.

As mentioned earlier, a Spark application runs one driver container and one or more executor containers. The application has multiple configuration parameters that control the allocation and usage of memory at the container level, as well as at the level of the JVMs running within the container. If overall usage at the container level exceeds the allocation, then the application will be killed by the resource management layer. If the overall usage at the Java heap level exceeds the allocation, then the application will be killed by the JVM.

Unravel has developed an algorithm to automatically find fixes to OOM failures by analyzing both successful and failed application runs. The algorithm takes into account configuration parameters correlated to OOM failures across the Spark driver, executor, container, and JVM – as well as a few other parameters that affect Spark memory usage. Space constraints prevent us from going into further details, but Unravel has also adapted algorithms from a related problem to address OOM issues.

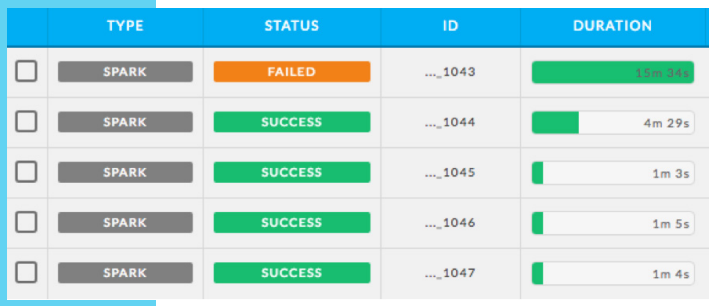


Figure 6: Automated Tuning of a Failed Spark Application

Figure 6 shows the algorithm in action for 5 successive runs of an application. Note that the first run is a failure due to OOM. The second run, leveraging a configuration setting produced by the algorithm, managed to get the app running successfully. The third run was able to run the application successfully – and faster than the second run – because the algorithm provided a more memory-efficient configuration.

Overallocation of memory can slow an application due to a long wait to allocate that much memory. Note how the algorithm is able to automatically find configurations that run the application successfully while being optimally efficient with resources. In this way, we can remove the burden of manually troubleshooting failed applications, enabling users to entirely focus their efforts on solving business problems with the modern data stack.

We can remove the burden of manually troubleshooting failed applications from the user, enabling them to focus on solving business problems with the big data stack.

CLUSTER OPTIMIZATION

Performing cluster-level workload analysis and optimization across a variety of distributed systems is complicated. Meeting key objectives – including performance management, autoscaling, and cost optimization – is imperative for both on-premises and cloud deployments and for both operations and developer teams.

Operational Insights

To help teams meet these objectives, Unravel analyzes metrics collected across pipelines and provides a rich set of actionable insights, including:

- Insights into application performance issues – e.g., whether an issue is due to code inefficiencies, contention with cloud and cluster resources, or hardware failure or inefficiencies like slow nodes
- Insights into cluster tuning, based on the aggregation of application data – e.g., whether a compute cluster is properly tuned at both the cluster and application level
- Insights into cluster utilization, cloud usage, and autoscaling

We also provide users with tools to help them understand how they are using their compute resources – for example, comparing cluster activity between two time periods, aggregating cluster workload, and providing summary reports for cluster usage and chargeback reports.

Unravel is different from other monitoring tools (e.g., Ambari, Cloudera Manager, Vertica) in that we offer a "single pane of glass" for understanding the entire modern data stack, and not just individual systems. We also harness AI, machine learning, and advanced analytics to uncover issues and inefficiencies, while recommending actionable solutions to these problems.

Cluster Resource Management

One of the biggest challenges in managing multi-tenant clusters is understanding how resources are being used by applications running in those clusters. Unravel provides a forensic view of each cluster's KPIs over time – and how they relate to those applications running in the cluster. For example, we can pinpoint the applications causing sudden spikes in the total CPU, VCore, and memory usage. We then enable users to drill down into these applications to understand their behavior, and, whenever possible, we provide intelligence and recommendations to help improve how these applications run.

In addition, Unravel also provides recommendations for fine-tuning cluster-wide parameters, helping maximize a cluster's efficiency based upon its typical workload. To do so, we:

- Collect performance data of prior completed applications
- Analyze these applications with respect to the cluster's current configuration
- Generate recommended cluster parameter changes
- Predict and quantify the impact these changes will have on applications executing in the future

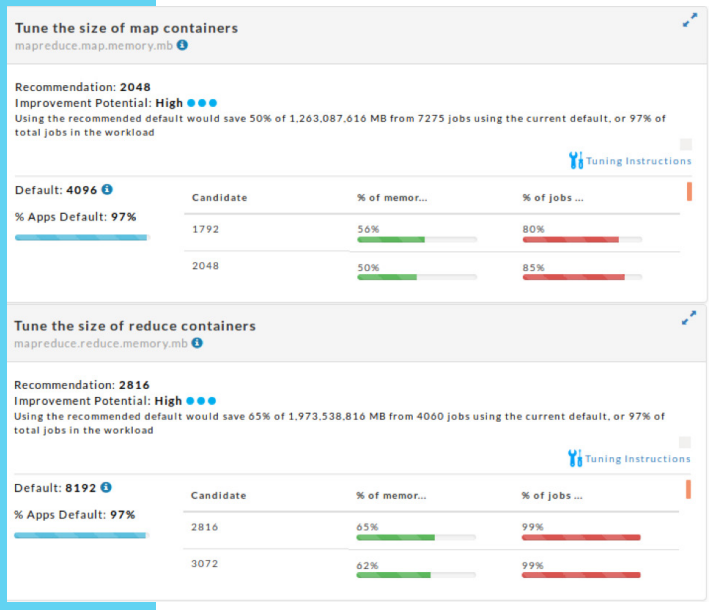


Figure 7: An Example of the Cluster-Wide Recommendations Unravel Provides

Figure 7 shows a sample of the cluster-wide recommendations Unravel can provide – in this case, for tuning the size of map containers (top) and reducing containers on a production cluster (bottom), in particular the allocated memory in MB.

In this example, at the cluster level, the default value of memory allocated for map tasks was set to 4096 MB. By analyzing historical data, we've identified a number of optimal alternative memory sizes for map containers. The figure shows a histogram of the distribution of applications across different memory sizes, along with potential rewards (the predicted amount of memory savings) and calculated risks (the percentage of jobs predicted to run if the candidate value is applied). Based on this data, we make a recommendation to set the memory size to 2048MB and calculate the potential for improvement: The recommended value could halve memory usage for 97% of the expected workload. Similar recommendations are made at the bottom of Figure 7 for the reduce containers.

Before applying our cluster-level recommendations (8 day interval)

- Total 7275 jobs = > 902 jobs/day
- Total vCore-Seconds = 18454324 (5126 vCore-Hours) = > 641 vCore-Hours/day
- Total #containers = 395096 = > 49387 containers/day
- Total #containers for MAP = 308371 (78%)
- Total #containers for REDUCE = 86725 (22%)

After applying our cluster-level recommendations (7 day interval)

- Total 13538 jobs = > 1934 jobs/day
- Total vCore-Seconds = 8582425 (2384 vCore-Hours) = > 341 vCore-Hours/day
- Total #containers = 292346 = > 41764 containers/day
- Total #containers for MAP = 211375 (72%)
- Total #containers for REDUCE = 80971 (28%)

Figure 8: Example Improvements of Applying Unravel's Cluster-Level Recommendations

Figure 8 shows the potential improvements that applying Unravel's cluster-level recommendations can have on the production cloud deployment of a financial institution: Our cluster tuning enabled 200% more applications (i.e., from 902 to 1,934 applications per day) to be run at 50% lower cost (i.e., from 641 to 341 VCore-hours per day), thereby increasing the organization's confidence and benefit in using the cloud.

Workload Analysis

Typically, how a particular application performs depends on what other applications are also running in the stack, altogether forming an application workload. This workload may contain heterogeneous applications in Hive, Spark SQL, Spark ML, and more.

Understanding how these applications run and impact one another is critical. Unravel analyzes queue usage across a set of clusters, identifying queue usage trends, suboptimal queue designs, workloads that run suboptimally on queues, convoys, "problem applications" (i.e., applications with excessive wait times), "problem users" (i.e., users who frequently run apps at max capacity for a long period), and queue usage per application type, user, and project.

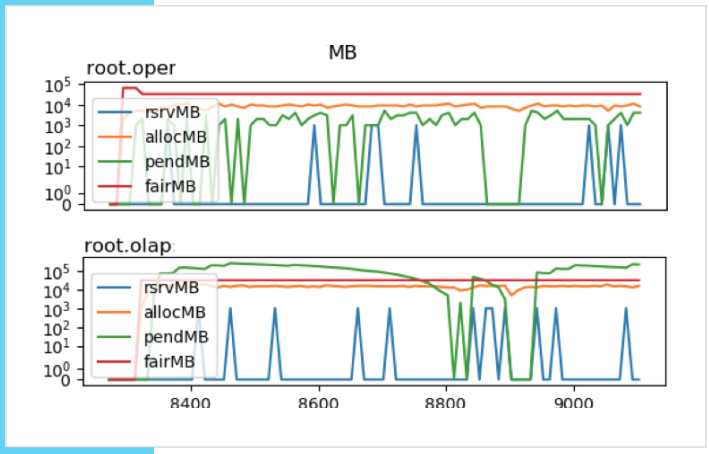


Figure 9: Example Memory Utilization for 2 Queues

Figure 9 shows the example memory utilization for two queues over time. In this example, the workload running in the *root.oper* queue does not use all of its allocated resources, while the workload in the *root.olap* queue needs additional resources. Pending resources (*pendMB* in green) that extend beyond the maximum capacity of resources (*fairMB*) can be assigned to this queue by Fair Scheduler. Similar analyses can be performed for other metrics like CPU, disk, scheduling, and more.

Based on these findings, Unravel generates queue-level insights and recommendations, including modifications for queue design and settings (e.g., changing resource budgets for a queue or maximum and minimum limits), reassigning workloads to different queues (i.e., moving an application or workload from one queue to another), and forecasting queue usage. Any of these recommendations could be applied to the situation shown in Figure 9.

A typical large-scale data deployment involves hundreds of queues, which can be a tedious process. Unravel can implement many of these recommendations using Auto-Actions, defined by complex actionable rules based on a wide variety of cluster metrics.

Each rule consists of a logical expression and a corresponding action:

- A logical expression aggregates cluster metrics and evaluates if the rule is being violated – it consists of two primary conditions: a prerequisite condition that causes a violation (e.g., the number of applications running or the amount of memory used) and a defining condition (e.g., who and what cause a violation, and when a violation can be caused).
- An action is a concrete, executable task, such as killing an application, moving an application to a different queue, sending an HTTP post, notifying a user, and so on.

Forecasting

Beside understanding the current status of their applications, stack, and systems, enterprise businesses need to be able to provision for future resources, usage, costs, job scheduling, and more. One of the advantages of the Unravel platform is that we collect a plethora of historical operational and application metrics that can be used for capability planning through predictive time-series models.

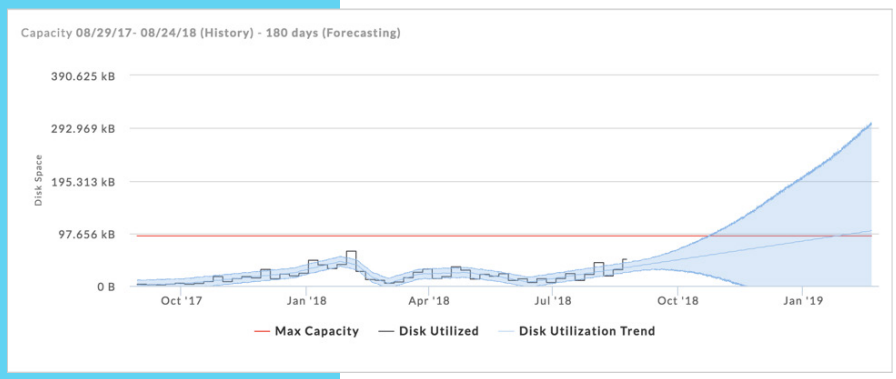
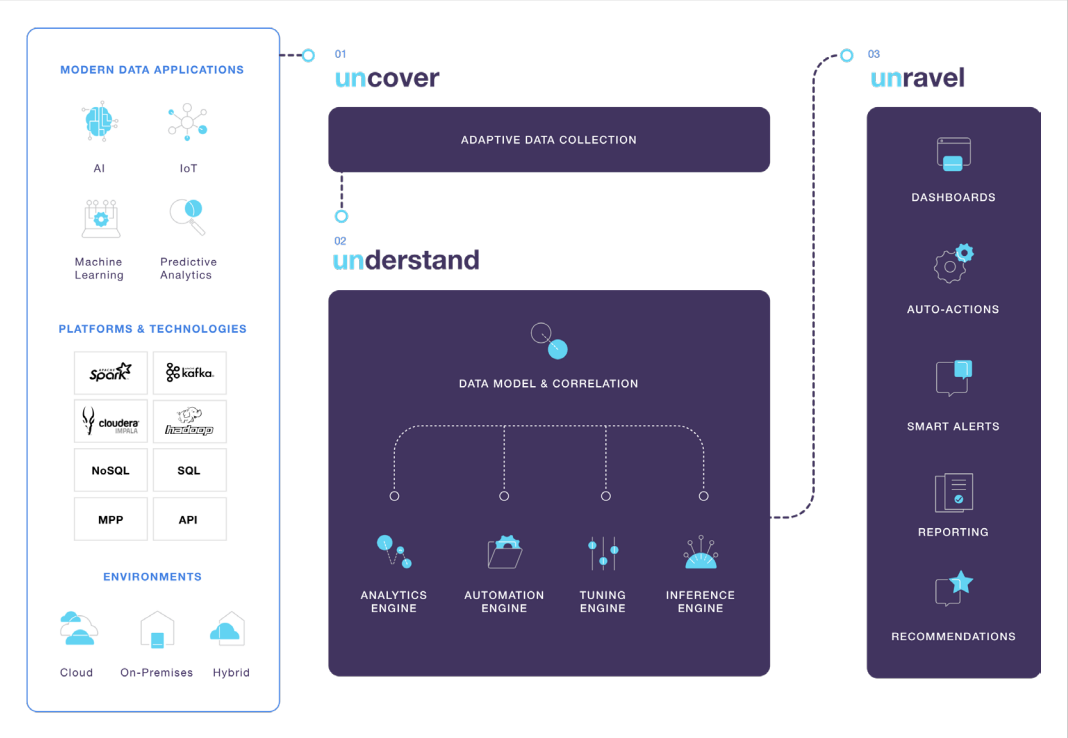


Figure 10: Disk Capacity Forecasting

Figure 10 shows an example disk-capacity forecasting chart; the black line shows actual utilization, and the light blue line shows a forecast of future needs within an error bound.

THE UNRAVEL PRODUCT ARCHITECTURE



Uncover: Adaptive Data Collection with Micro-Sensors

Unravel uses native platform APIs and lightweight micro-sensors to perform real-time, non-intrusive ingestion and analysis of runtime data. This *operational metadata* is collected from applications and from underlying components, including Spark, Hadoop, Kafka, MPP, NoSQL, and many others. Maximum coverage of the stack means maximum observability of your environment.

Unravel collects metadata about the full range of data-driven apps, analytics, machine learning, AI, IoT, and business intelligence workloads. Unravel also collects runtime metrics about modern data infrastructure – whether it is on-premises, in the cloud (i.e., using AWS, Azure, or GCP), or in hybrid and multi-cloud environments.

Understand: The Unravel Data Model and Processing Engines

Unravel uses AI, machine learning, and other analytical approaches to optimize modern data application performance. Unravel correlates the collected operational metadata mentioned above to create a dynamic data model that reflects the current state of the stack and running applications. This data is used as the basis for all of our recommendations, automation, alerts, and reporting.

Unravel is unlike other monitoring systems or log analysis tools that simply aggregate runtime metrics and display them in graphs and charts – instead, Unravel uses this data model to correlate metrics about apps, resources, data sets, and users to deliver a complete understanding of performance.

Establishing enterprise-grade SLAs for modern data applications requires the most accurate intelligence with a multivariate approach to tuning and troubleshooting. Unravel provides an "MRI view" of applications and infrastructure, using four main intelligence engines and application metrics that can be used for capability planning through predictive time-series models.

ANALYTICS ENGINE

Unravel uses a variety of analytics technologies – including dependency maps, rules engines, and AI and machine learning algorithms like random forest, Bayesian classifiers, and neural networks – choosing the optimal approach for optimizing performance and troubleshooting issues and failures.

INFERENCE ENGINE

Unravel's data model makes recommendations based on more than 70 different events, correlated across an unlimited number of permutations. These recommendations include providing the root cause analysis of failures and slowdowns, improving resource utilization and throughput, speeding application performance, and fixing application failures.

TUNING ENGINE

Unravel can dramatically increase the speed of application performance, using AI and machine learning to optimize configuration settings, analyzing application code for optimal performance, identifying and adjusting poorly sized containers, mitigating poorly utilized memory and CPU, applying caching where best suited, optimizing load balancing across executors, and more.

AUTOMATION ENGINE

Unravel goes beyond monitoring performance to automatically taking action to improve performance, using the tuning and troubleshooting recommendations provided by the intelligence engines described above. These Auto-Actions include:

- Auto-tuning slow apps and stack components
- Killing rogue processes and applications
- Moving jobs from one queue to another based on priority or service-level violation
- Executing custom scripts, processes, and applications via HTTP callouts
- Smart alerting delivered through email, Jira, Slack, and PagerDuty

UNRAVEL: OUTPUTS AND ACTIONS FROM THE UNRAVEL PLATFORM

Dashboards

Unravel provides real-time visibility into data operations and applications with an inside-out view on the health and performance of clusters and apps. Unravel's dashboards offer real-time intelligence on recent failures, slowdowns, and inefficiencies. Users can also use these dashboards to drill down to fine-grained views into each application, resource, and service. Dashboards are also customizable based on roles and responsibilities.

Auto-Actions

Unravel's Auto-Actions offers policy-driven automation of our tuning and troubleshooting recommendations. Our platform goes beyond passive monitoring to take corrective or preventive action by leveraging user-defined rules for application performance and behavior. Unravel enables users to define specific actions to take for specific rule violations, including:

- Emailing or Slacking ops teams and developers
- Killing a process or job
- Moving applications to another queue

Smart Alerts

Smart Alerts are part of Unravel Auto-Action capabilities. Not threshold-based or passive, these alerts provide complete and contextualized evidence and the root cause of issues, including:

- Applications taking too long to complete
- Missed SLAs
- Jobs or applications using too much memory or too many vCores
- Spark SQL, Hive, or Impala queries taking too long
- Inefficient apps and users on the cluster
- Application and component failures

Reporting

Unravel provides rich reporting on both the operational and resource level, including resource consumption, cluster operations, application workloads, and more. For operational reporting, Unravel provides:

- Chargeback and showback by department, user, app type, and more
- Cluster usage summary and comparison by user, app type, department, and project
- Detailed reporting on application workloads on the cluster

Unravel also provides detailed data insights for:

- Usage and data access by app and user
- Hot and cold analysis of data sets, to drive best storage strategy

Recommendations and Insights

Unravel provides deep intelligence and actionable recommendations for everything from cluster operations to modern data applications and pipelines. Recommendations cover a wide range of technologies and conditions, including:

- Rightsizing resource allocation and utilization for containers, CPU, memory, and partitions
- Remediating underperforming jobs, applications, and stack components
- Increasing the speed of applications with improved caching, garbage collection, parallelism, and load balancing
- Remediating application failures, including out-of-memory conditions, missing files and data, and timeouts

CONCLUSION

Unravel harnesses the latest in AI, machine learning, and predictive analytics to drive optimal performance in today's large-scale, data-driven applications and pipelines. Unravel also addresses the challenges of predicting and avoiding application failures across multiple processing engines. We have developed a unique platform that provides intelligent, automated solutions to address the complexity of modern data programs.

At Unravel, we are creating the next generation of data operations platforms, working with today's enterprise businesses to help them understand and solve their real-world challenges while keeping operational costs, staffing levels, and MTTR low.

About Unravel

Unravel radically simplifies the way businesses understand and optimize the performance of their modern data applications – and the complex pipelines that power those applications. Providing a unified view across the entire stack, Unravel's AI-powered data operations platform leverages AI, machine learning, and advanced analytics to offer actionable recommendations and automation for tuning, troubleshooting, and improving performance – both today and tomorrow.

By operationalizing how you do data, Unravel's solutions support modern big data leaders, including Kaiser Permanente, TIAA, Adobe, Deutsche Bank, Wayfair, and Neustar. The company is headquartered in Palo Alto, California, and is backed by Menlo Ventures, GGV Capital, M12, Data Elite Ventures, and Jyoti Bansal. To learn more, visit unraveldata.com.

unravel

unraveldata.com | hello@unraveldata.com