Current Challenges

Netflix faced several challenges with their previous rule-based auto-remediation system:

1. The rule-based classifier required continuous maintenance and manual updates to handle new failure patterns
2. It couldn't effectively handle complex failures with multiple symptoms
3. The system had high false-positive rates leading to incorrect remediation actions
4. Rule creation was labor-intensive and required domain expertise
5. As data pipeline complexity increased, the rule-based approach couldn't scale
6. Many remediation opportunities were missed due to the limitations of static rules
7. Engineers spent significant time manually investigating and resolving failures that could potentially be automated.

Proposed Solution

Netflix developed a machine learning-powered auto-remediation system with these key components:

1. Automatically learns from historical failures and their resolutions
2. Uses a multi-class classification model to identify failure types
3. Leverages rich contextual information from data pipeline executions
4. Continuously improves as it processes more failures
5. Integrates with existing remediation action execution systems
6. Maintains human oversight while increasing automation coverage

Solution Design

To address the above-mentioned challenges, our basic methodology is to integrate the rule-based classifier with an ML service to generate recommendations, and use a configuration service to apply the recommendations automatically:

- Generating recommendations. We use the rule-based classifier as the first pass to classify all errors based on predefined rules, and the ML service as the second pass to provide recommendations for memory configuration errors and unclassified errors.
- Applying recommendations. We use an online configuration service to store and apply the recommended configurations. The pipeline is fully automated, and the services used to generate and apply recommendations are decoupled.

Summary of Major Services

1. Nightingale – ML-based retry recommendation service

- Determines whether a failed job is restartable.
- Suggests optimized configurations for restarting the job.

2. ConfigService – Online configuration management

- Stores recommended configurations as JSON patches.
- When the Scheduler requests configurations, it applies patches to the original settings and returns updated configurations.

3. Pensive – Error classification service

- Uses rule-based classification to analyze job failures.
- Calls Nightingale for recommendations and stores them in ConfigService for job restarts.

4. Scheduler (Netflix Maestro) – Job execution manager

- When a job fails, it calls Pensive for error classification.
- Fetches recommended configurations from ConfigService and restarts the job accordingly.

the sequence of service calls with Auto Remediation:

1. Upon a job failure, Scheduler calls Pensive to get the error classification.
2. Pensive classifies the error based on the rule-based classifier. If the error is identified to be a memory configuration error or an unclassified error, it calls Nightingale to get recommendations.
3. With the obtained recommendations, Pensive updates the error classification result and saves the recommended configurations to ConfigService; and then returns the error classification result to Scheduler.
4. Based on the error classification result received from Pensive, Scheduler determines whether to restart the job.
5. Before restarting the job, Scheduler calls ConfigService to get the recommended configuration and retries the job with the new configuration.

New version of Nightingle service

The ML service, i.e., Nightingale, aims to generate a retry policy for a failed job that trades off between retry success probability and job running costs. It consists of two major components:

- A prediction model that jointly estimates a) probability of retry success, and b) retry cost in dollars, conditional on properties of the retry.

- An optimizer which explores the Spark configuration parameter space to recommend a configuration which minimizes a linear combination of retry failure probability and cost.

The prediction model is retrained offline daily, and is called by the optimizer to evaluate each candidate set of configuration parameter values. The optimizer runs in a RESTful service which is called upon job failure. If there is a feasible configuration solution from the optimization, the response includes this recommendation, which ConfigService uses to mutate the configuration for the retry. If there is no feasible solution–in other words, it is unlikely the retry will succeed by changing Spark configuration paramPrediction Model (Multi-Output Supervised Learning)

- Goal: Predict retry success and retry cost based on job metadata.
- Architecture:
  - Uses a Feedforward Multilayer Perceptron (MLP) with two heads (one for each prediction).
  - Shared feature set includes structured numeric values (e.g., Spark memory config) and unstructured metadata (e.g., user name).
  - Feature hashing is applied to high-cardinality categorical values, converting them into lower-dimensional embeddings.
- Training Data:
  - Labels: (a) Did retry fail? (b) Retry cost.
  - Includes failed retries due to memory configuration errors or unclassified errors.
- Deployment:
  - Validated models are stored in Metaflow Hosting for online inference.
  - The model is queried for every configuration recommendation request.

Optimizer (Bayesian Optimization with Meta's Ax Library)

- Goal: Optimize Spark configuration parameters to minimize retry failure probability and cost.
- Process:
  - A job failure triggers a request to Nightingale.
  - Feature vector is built based on Spark config parameters (e.g., spark.executor.memory).
  - Bayesian Optimization generates candidate configurations iteratively.
  - Each candidate is evaluated using the prediction model, estimating retry failure probability & cost.
  - The best configuration (minimizing failure & cost) is selected for ConfigService. If no feasible solution is found, retries are disabled.
- Performance Optimization:
  - Initial design faced timeouts due to bottlenecks in candidate generation.
  - Solution: Leveraged GPU acceleration in Ax library, reducing timeout rates significantly.

Production Deployment of Auto Remediation

- Handles Memory Configuration & Unclassified Errors in Spark jobs to enhance user experience and reduce costs.
- For Memory Configuration Errors:
    - Successful retries with recommended configurations reduce operational load and job costs.
    - Failed retries do not worsen user experience.
- For Unclassified Errors:
    - ML model predicts if a retry is likely to fail and disables unnecessary retries.
    - Users can override ML recommendations by adding rules to the rule-based classifier for critical jobs.
- Results:
    - Successfully remediated 56% of memory configuration errors without human intervention.
    - Reduced compute costs by ~50% by optimizing retries.

Optimizing for User Experience vs. Cost

- Currently, a conservative policy is in place to avoid excessive retry disabling.
- Users who prefer always retrying can add new rules to the rule-based classifier.
- Future iterations aim to gradually increase cost savings by fine-tuning the ML model's objective function.

Beyond Error Handling: Right Sizing Initiative

- Objective: Optimize Spark job resource allocation to minimize overprovisioning.
- Findings: Average requested executor memory is 4× higher than actual usage, leading to unnecessary costs.
- Approach:
    - Use heuristic- and ML-based methods to infer optimal resource configurations.
    - Automate configuration updates via ConfigService without human intervention.
- Impact: Expected to save millions of dollars annually by preventing overprovisioning while maintaining performance.