Resource Management for Advanced Data Analytics at Large Scale

Final Public Oral

Haoyu Zhang

Committee: Mike Freedman (advisor), Kyle Jamieson, Kai Li, Wyatt Lloyd, Jennifer Rexford

PRINCETON UNIVERSITY
Advanced data analytics: making sense of complex data

- Unstructured, multimodal
  *numerical, text, images, videos, …*
- High-dimensional, interconnected
  *medical, linked social graphs, …*
- Growing very fast in volume
- Discover interpretable patterns
- Understand causal relationships
- Make informed predictions and decisions
Data

Complexity

Resource
Challenge 1: the growth of data volume

Batch processing

- **10s PB** new data per day for Spark jobs
- **100s TB** new data per day for a single job

Video stream analytics

- NYPD expands surveillance net to fight crime as well as terrorism
- Cameras and IoT: Going from smart to intelligent
- Microsoft looks to stop bike crashes before they happen, testing Minority Report-style predictive intelligence

Machine learning

- **100+M** user ratings of 17,770 movies
- **14+M** images of 1,000 categories
Challenge 2: the complexity of analytics

Batch processing

- >50% batch jobs have multiple stages
- 10x larger than available memory

Video stream analytics

- 1Fps object tracking on 8-core node [1]
- 30GFlops to recognize objects in image [2]

Machine learning

- 600K training steps to converge [3]
- 10K hyperparameter combinations to explore [4]

---

[1] VOT Challenge 2015 Results  
[3] He et al. 2015  
Challenge 3: limited cluster resources

- Our rapidly improving hardware technology is coming to a “grinding halt” [1]
  
  - DRAM and disk capacity: double once in next decade [2]
  
  - CPU performance: double in two decades [2]
  
  - Moore’s Law is ending…

Datacenter resource scheduling

- Treat tasks as black boxes
- Based on general principles
  - fairness, locality, load balancing, …

allocate executors
assign tasks

Worker Machine 1
Worker Machine n

Task Executor
Task Executor
Task Executor
Task Executor
New opportunities to optimize scheduling

• Batch processing
  • large amount of fragmented I/O in multi-stage jobs
  • largest Spark deployment known has 8,000 nodes

• Video stream analytics
  • quality-resource-delay tradeoffs between queries
  • live analytics deployed on public & private cloud

• Machine learning
  • iterative training process with diminishing returns
  • TPU, Facebook Big Basin in datacenters for ML jobs

Occupying the cloud!
In this talk

- VideoStorm: Live Video Analytics [NSDI ’17]
- SLAQ: Quality-Driven ML Scheduling [SoCC ’17 🏆]
- Riffle: Optimized Shuffle Service [EuroSys ’18]
Riffle: Optimized Shuffle Service for Large-Scale Data Analytics

Haoyu Zhang, Brian Cho, Ergin Seyfe, Avery Ching, Michael J. Freedman

European Conference on Computer Systems (EuroSys ’18)

PRINCETON UNIVERSITY

facebook
Batch analytics systems are widely used

- Large-scale SQL queries
- Custom batch jobs
- Pre-/Post-processing for ML

At **Facebook**

- 10s of PB new data is generated every day for batch processing
- 100s of TB data is added to be processed by a single job
Batch analytics jobs: logical graph

narrow dependency

wide dependency
Batch analytics jobs: DAG execution plan

- Shuffle: all-to-all communication between stages
- >10x larger than available memory, strong fault tolerance requirements
  → on-disk shuffle files
The case for tiny tasks

• Benefits of slicing jobs into small tasks
  • Improve parallelism [Tinytasks HotOS 13] [Subsampling IC2E 14] [Monotask SOSP 17]
  • Improve load balancing [Sparrow SOSP 13]
  • Reduce straggler effect [Dolly NSDI 13] [SparkPerf NSDI 15]
The case against tiny tasks

Although we were able to run the Spark job with such a high number of tasks, we found that there is significant performance degradation when the number of tasks is too high.

- Engineering experience often argues against running too many tasks
  - Medium scale → very large scale (10x larger than memory space)
  - Single-stage jobs → multi-stage jobs (> 50%)

[*] Apache Spark @Scale: A 60 TB+ Production Use Case. [https://tinyurl.com/yadx29gl](https://tinyurl.com/yadx29gl)
Shuffle I/O grows *quadratically* with data

- Large amount of fragmented I/O requests
  - Adversarial workload for hard drives!
Strawman: fix number of tasks in a job

- Tasks spill intermediate data to disk if data splits exceed memory capacity
- Larger task execution reduces shuffle I/O, but increases spill I/O
Strawman: tune number of tasks in a job

- Need to retune when input data volume changes for each individual job
- Bulky tasks can be detrimental [Dolly NSDI 13] [SparkPerf NSDI 15] [Monotask SOSP 17]
  - straggler problems, imbalanced workload, garbage collection overhead
Small Tasks

Large Amount of Fragmented Shuffle I/O

Bulky Tasks

Fewer, Sequential Shuffle I/O
Riffle: optimized shuffle service

- Riffle shuffle service: a long running instance on each physical node
- Riffle scheduler: keeps track of shuffle files and issues merge requests
Riffle: optimized shuffle service

- When receiving a merge request
  1. Combines small shuffle files into larger ones
  2. Keeps original file layout
- Reducers fetch fewer, large blocks instead of many, small blocks
Results with merge operations on synthetic workload

- Riffle reduces number of fetch requests by 10x
- Reduce stage -393s, map stage +169s $\rightarrow$ job completes 35% faster
Best-effort merge

- Observation: slowdown in map stage is mostly due to stragglers

- Best-effort merge: mixing merged and unmerged shuffle files
  - When number of finished merge requests is larger than a user specified percentage threshold, stop waiting for more merge results
Results with best-effort merge

- Reduce stage -393s, map stage +52s → job completes 53% faster
  - Riffle finishes job with only ~50% of cluster resources!
Additional enhancements

• Handling merge operation failures
• Efficient memory management
• Balance merge requests in clusters
Experiment setup

- **Testbed**: Spark on a 100-node cluster
  - Each node has 56 CPU cores, 256GB RAM, 10Gbps Ethernet links
  - Each node runs 14 executors, each with 4 cores, 14GB RAM

- **Workload**: 4 representative production jobs at Facebook

<table>
<thead>
<tr>
<th></th>
<th>Data</th>
<th>Map</th>
<th>Reduce</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>167.6 GB</td>
<td>915</td>
<td>200</td>
<td>983 K</td>
</tr>
<tr>
<td>2</td>
<td>1.15 TB</td>
<td>7,040</td>
<td>1,438</td>
<td>120 K</td>
</tr>
<tr>
<td>3</td>
<td>2.7 TB</td>
<td>8,064</td>
<td>2,500</td>
<td>147 K</td>
</tr>
<tr>
<td>4</td>
<td>267 TB</td>
<td>36,145</td>
<td>20,011</td>
<td>360 K</td>
</tr>
</tbody>
</table>
Reduction in shuffle I/O requests

- Riffle reduces # of I/O requests by 5--10x for medium / large scale jobs
Savings in end-to-end job completion time

- Map stage time is almost not affected (with best-effort merge)
- Reduces job completion time by 20--40% for medium / large jobs
Part I Conclusion

• Shuffle I/O becomes scaling bottleneck for multi-stage jobs

• Efficiently schedule merge operations, mitigate merge stragglers

• Riffle is deployed for Facebook’s production jobs processing PBs of data
Live Video Analytics at Scale with Approximation and Delay-Tolerance

Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, Michael J. Freedman

USENIX Symposium on Networked Systems Design and Implementation (NSDI ’17)
Video analytics queries

Intelligent Traffic System

AMBER Alert

Electronic Toll Collection

Video Doorbell
Video query: a pipeline of *transforms*

- Example: traffic counter pipeline
Video queries are expensive in resource usage

- Example: traffic counter pipeline

- When processing *thousands* of video streams in multi-tenant clusters
  - How to reduce processing cost of a query?
  - How to manage resources efficiently across queries?
Vision algorithms are intrinsically *approximate*

- **Knobs**: parameters / implementation choices for transforms

- License plate reader → window size
- Car tracker → mapping metric
- Object classifier → DNN model

- **Query configuration**: a combination of knob values
Knobs impact quality and resource usage

Frame Rate: 3
Resolution: 720p
Quality=0.93, CPU=0.54

Frame Rate: 1
Resolution: 480p
Quality=0.57, CPU=0.09
Tuning the knobs all together

- Orders of magnitude cheaper resource demand for little quality drop
- No analytical models to predict resource-quality tradeoff
  - Different from approximate SQL queries
Diverse quality and lag requirements

Lag: time difference between frame arrival and frame processing

<table>
<thead>
<tr>
<th>Service</th>
<th>Quality</th>
<th>Lag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toll Collection</td>
<td>High</td>
<td>Hours</td>
</tr>
<tr>
<td>Intelligent Traffic</td>
<td>Moderate</td>
<td>Few Seconds</td>
</tr>
<tr>
<td>AMBER Alert</td>
<td>High</td>
<td>Few Seconds</td>
</tr>
</tbody>
</table>
Decide configuration and resource allocation to maximize quality and minimize lag within the resource capacity.
Video analytics framework: Challenges

1. Many knobs → large configuration space
   • No known analytical models to predict quality and resource impact
2. Diverse requirements on quality and lag
   • Hard to configure and allocate resources jointly across queries
VideoStorm: Solution Overview

- Builds model
- Reduces config space

Trades off quality and lag across queries
Offline: query profiling

- Profile: configuration $\Rightarrow$ resource, quality
  - Ground-truth: labeled dataset or results from golden configuration
  - Explore configuration space, compute average resource and quality

![scatter plot showing resource demand vs. quality, with more efficient configurations indicated]

\( \times \) is strictly better than \( \bigotimes \) in quality and resource efficiency

more efficient
Offline: Pareto boundary of configuration space

- **Pareto boundary**: optimal configurations in resource efficiency and quality
  - Cannot further increase one without reducing the other
  - Orders of magnitude reduction in config. search space for scheduling

![Pareto optimal](image)
VideoStorm: Solution Overview

query → Profiler → resource-quality profile → Scheduler

utility function

Workers

offline

online
Online: utility function and scheduling

- Utility function: encode **goals** and **sensitivities** of quality and lag
  - Users set required quality and tolerable lag
  - Reward additional quality, penalize higher lag

- Schedule for two natural goals
  - **Maximize the minimum utility** – (max-min) fairness
  - **Maximize the total utility** – overall performance

- Allow lag accumulation during resource shortage, then catch up
VideoStorm Evaluation Setup

• **Platform:**
  - Microsoft Azure cluster
  - Each worker contains 4 cores of the 2.4GHz Intel Xeon processor and 14GB RAM

• **Four types of vision queries:**
  - license plate reader
  - car counter
  - DNN classifier
  - object tracker
Experiment Video Datasets

- Operational traffic cameras in Bellevue and Seattle
- 14–30 frames per second, 240P–1080P resolution
Resource allocation during burst of queries

• Start with 300 queries:
  ① Lag Goal=300s, Low-Quality 60%
  ② Lag Goal=20s, Low-Quality 40%

• Burst of 150 seconds (50 – 200):
  ③ 200 LPR queries (AMBER Alert)
     Lag Goal=20s, High-Quality

• VideoStorm scheduler:
  ③ dominate resource allocation
     run ② with lower quality
     significantly delay ①

All meet quality and lag goals
Resource allocation during burst of queries

- Start with 300 queries:
  ① Lag Goal=300s, low-quality ~60%
  ② Lag Goal=20s, low-quality ~40%

- Compare to a fair scheduler with varying burst duration:
  - Quality improvement: up to 80%
  - Lag reduction: up to 7x

- VideoStorm scheduler:
  ③ dominate resource allocation
  significantly delay ①
  run ② with lower quality
  All meet quality and lag goals
VideoStorm Scalability

• Frequently reschedule and reconfigure in reaction to changes of queries

• Even with thousands of queries, VideoStorm makes rescheduling decisions in just a few seconds
Related Work

• Video query optimization
  • Optasia [SoCC ’16], NoScope [VLDB ’17], EVA [SysML ’18]
  • Share common operators and reuse results from different queries

• Video systems on cloud-edge architecture
  • Vigil [MobiCom ’15], Firework [TPDS ’18], Chameleon [SIGCOMM ’18]
  • Placing tasks / operators of a processing pipeline to different locations
Part II Conclusion

- VideoStorm explores quality-resource-lag tradeoff in video queries
- Offline profiler: efficient estimates resource-quality profiles
- Online scheduler: optimizes jointly for quality and lag of queries

- Significant improvement in achieved quality and lag
Deployment at Bellevue Traffic Department

https://vavz.azurewebsites.net
SLAQ: Quality-Driven Scheduling for Distributed Machine Learning

Haoyu Zhang*, Logan Stafman*, Andrew Or, Michael J. Freedman

ACM Symposium on Cloud Computing (SoCC ’17)

🏆 Best Paper Award
ML algorithms are *approximate*

- ML model: a parametric transformation

\[ f_\theta \]
ML algorithms are *approximate*

- ML model: a parametric transformation

\[
X \rightarrow f_\theta \rightarrow Y
\]

- maps input variables \(X\) to output variables \(Y\)
- typically contains a set of *parameters* \(\theta\)
- **Loss function**: discrepancy of model output and ground truth

- **Quality**: how well model maps input to the correct output
Training ML models: an iterative process

- Training algorithms iteratively minimize a loss function
  - E.g., stochastic gradient descent (SGD), L-BFGS
Training ML models: an *iterative* process

- Quality improvement is subject to **diminishing returns**
- More than **80% of work done in 20% of time**
Exploratory ML training: not a one-time effort

- Train model multiple times for exploratory purposes
- Provide early feedback, direct model search to high quality models
How to schedule multiple training jobs on shared cluster?

• Problems with resource fairness scheduling
  • Jobs in early stage: could benefit a lot from additional resources
  • Jobs almost converged: make only marginal improvement
SLAQ: quality-aware scheduling

- Intuition: in exploratory ML training, more resources should be allocated to jobs that have the most potential for quality improvement
Solution Overview

- Normalize quality metrics
- Predict quality improvement
- Quality-driven scheduling
Universal quality measurement metric

• Accuracy?
  • Precision, F1 Score, Area Under Curve, …
    ✗ Not applicable to non-classification models

• Loss function values?
  • Square loss, smoothed hinge loss, logistic loss, cross entropy loss, …
    ✗ Do not have comparable magnitudes or known ranges

• Reduction of loss values (ΔLoss)
  ✓ Always decrease to 0 as the loss function value converges
Normalizing quality metrics

- Quality: normalized change of loss values \(w.r.t\.) largest change so far

![Graph showing normalized \(\Delta\) Loss over iterations for different algorithms]

- Currently does not support some non-convex optimization algorithms
Training iterations: loss prediction

• Previous work: offline profiling / analysis [Ernest NSDI 16] [CherryPick NSDI 17]
  • Overhead for frequent offline analysis is huge
• Strawman: use last $\Delta$Loss as prediction for future $\Delta$Loss
• SLAQ: online prediction using weighted curve fitting
Scheduling approximate ML training jobs

• Predict how much quality can be improved when assign X workers to jobs
• Reallocate workers to maximize quality improvement
Experiment setup

- Representative mix of training jobs with Spark MLlib
- Compare against a work-conserving fair scheduler

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Acronym</th>
<th>Type</th>
<th>Optimization Algorithm</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-Means</td>
<td>K-Means</td>
<td>Clustering</td>
<td>Lloyd Algorithm</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>LogReg</td>
<td>Classification</td>
<td>Gradient Descent</td>
<td>Epsilon [33]</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>SVM</td>
<td>Classification</td>
<td>Gradient Descent</td>
<td>Epsilon</td>
</tr>
<tr>
<td>SVM (polynomial kernel)</td>
<td>SVMPol</td>
<td>Classification</td>
<td>Gradient Descent</td>
<td>MNIST [34]</td>
</tr>
<tr>
<td>Gradient Boosted Tree</td>
<td>GBT</td>
<td>Classification</td>
<td>Gradient Boosting</td>
<td>Epsilon</td>
</tr>
<tr>
<td>GBT Regression</td>
<td>GBTR</td>
<td>Regression</td>
<td>Gradient Boosting</td>
<td>YearPredictionMSD [35]</td>
</tr>
<tr>
<td>Multi-Layer Perceptron Classifier</td>
<td>MLPC</td>
<td>Classification</td>
<td>L-BFGS</td>
<td>Epsilon</td>
</tr>
<tr>
<td>Latent Dirichlet Allocation</td>
<td>LDA</td>
<td>Clustering</td>
<td>EM / Online Algorithm</td>
<td>Associated Press Corpus [36]</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>LinReg</td>
<td>Regression</td>
<td>L-BFGS</td>
<td>YearPredictionMSD</td>
</tr>
</tbody>
</table>
Evaluation: cluster-wide quality and time

**Quality**

- SLAQ’s average loss is 73% lower than that of the fair scheduler

**Time**

- SLAQ reduces time to reach 90% (95%) loss reduction by 45% (30%)
Part III Conclusion

• SLAQ leverages the approximate and iterative ML training process

• Highly tailored prediction for iterative job quality

• Allocate resources to maximize quality improvement

• SLAQ achieves better overall quality and end-to-end training time
Conclusion
Research Summary

• Resource management for advanced data analytics
  • *Live Video Analytics at Scale with Approximation and Delay-Tolerance* [NSDI ’17]
  • *SLAQ: Quality-Driven Scheduling in Distributed Machine Learning* [SoCC ’17][SysML ’18]
  • *Riffle: Optimized Shuffle Service for Large-Scale Data Analytics* [EuroSys ’18]

• Network-assisted system acceleration
  • *NetCache: Balancing Key-Value Stores with Fast In-Network Caching* [SOSP ’17]
  • *NetChain: Scale-Free Sub-RTT Coordination* [NSDI ’18]

• SDN fault tolerance
  • *Ravana: Controller Fault-Tolerance in Software-Defined Networks* [SOSR ’15]
Thanks!

Haoyu Zhang

http://www.haoyuzhang.org