38th IEEE International Conference on Data Engineering

Automatic Performance Tuning for Distributed Data Stream Processing Systems

Herodotos Herodotou, Cyprus University of Technology
Lambros Odysseos, Cyprus University of Technology
Yuxing Chen, Tencent Inc.
Jiaheng Lu, University of Helsinki

Cyprus University of Technology

Tencent

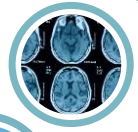
UNIVERSITY OF HELSINKI

Outline

- Introduction & Motivation
- Method Taxonomy
- Parameter Tuning Approaches
  - Cost modeling
  - Simulation-based
  - Experiment-driven
  - Machine learning
  - Adaptive tuning
- Comparison of Approaches
- Open Problems & Challenges

Our Data-driven World

Massive Data



Medicine

- MRI & CT scans
- Patient records

Entertainment

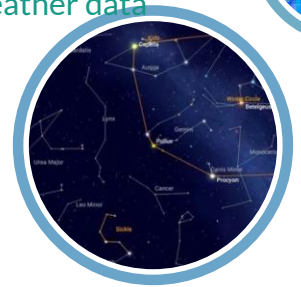
- Internet images
- Movie & music clips

Science

- Genomics
- Astronomy
- Weather data

Business

- Product sales
- Stock market
- Customer data



Humanities

- Social interactions
- Historical documents

Analysis

Insights

Ad placement

Scientific breakthroughs

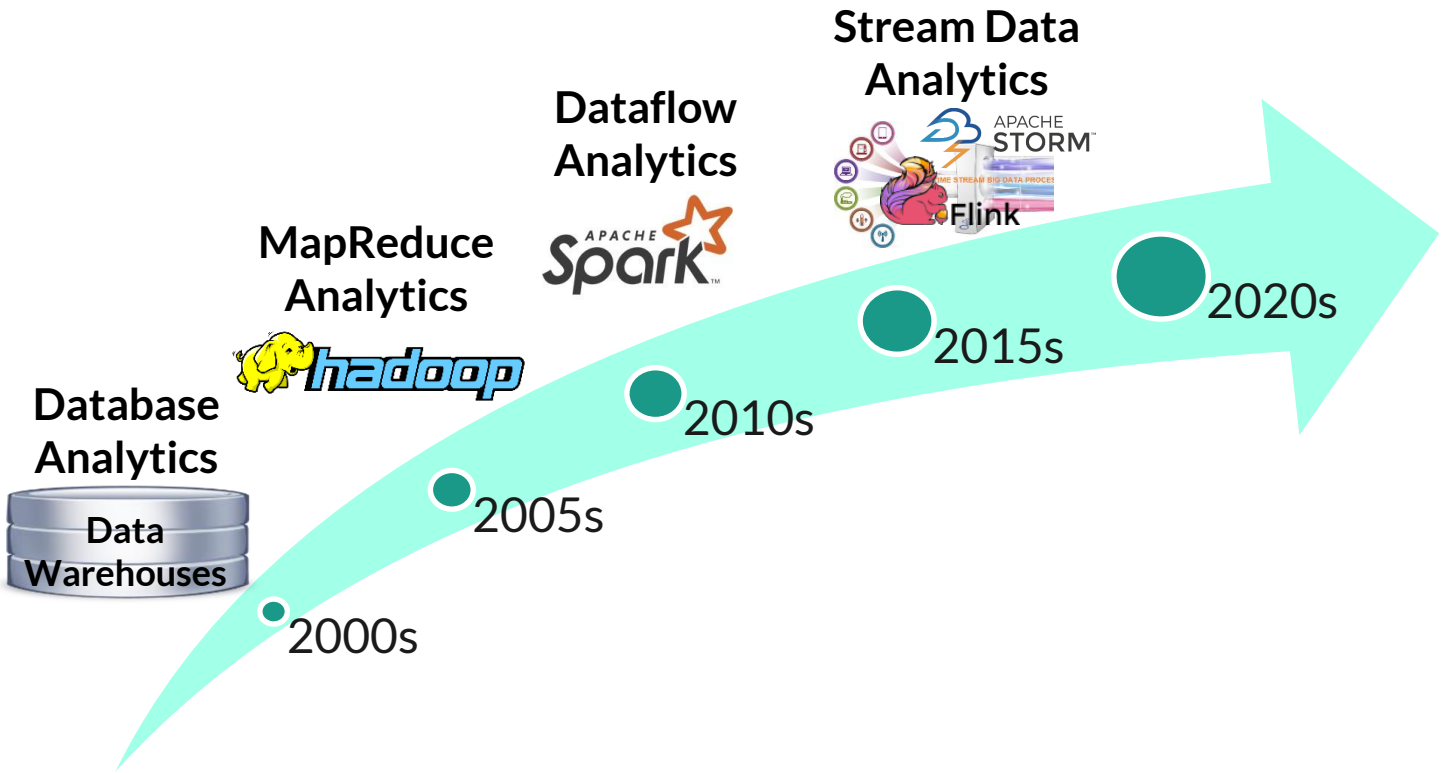
Business process efficiencies

Personalized recommendations

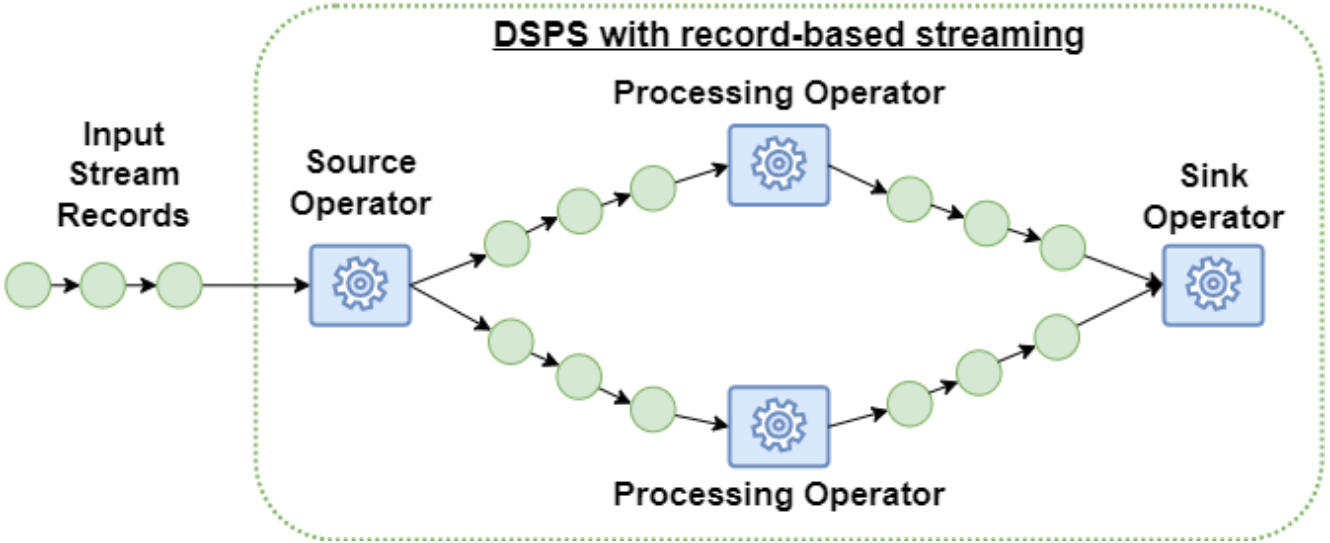
Improved healthcare

Fraud detection

Evolution of Big Data to Real Time Analytics Systems

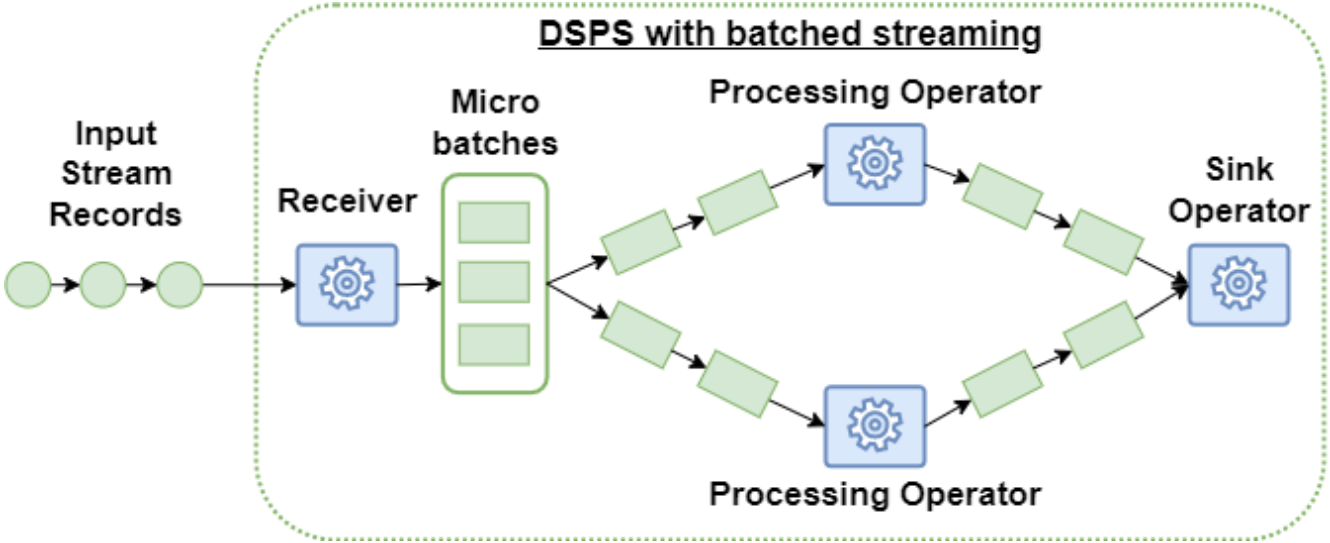


Record-based Stream Processing Systems



- Enables real-time processing, providing lower latencies but sacrifices throughput

Micro-batched Stream Processing Systems



TRIDENT STORM

Spark Streaming

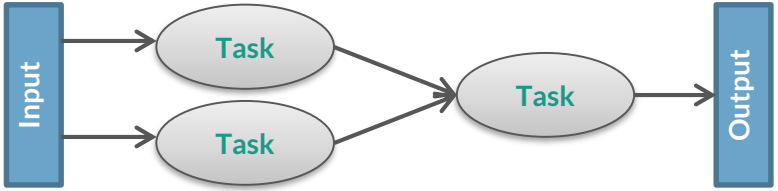
- Benefits from higher throughput but typically results in higher average latency

Decisions when Executing Streaming Workloads

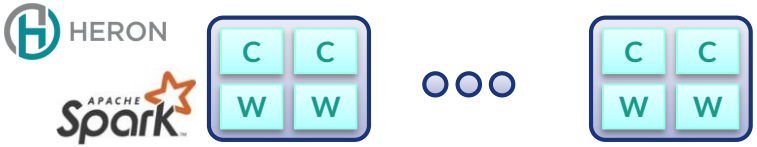
Goal:

Execute a streaming application with latency < 50ms

Application



Platform



Resources

Decisions:

- Task parallelism
- Micro-batch size
- …

- Number of cores
- Memory settings
- …

- Number of nodes
- Node/VM specs
- …

Selected Perf-Aware Parameters in Storm

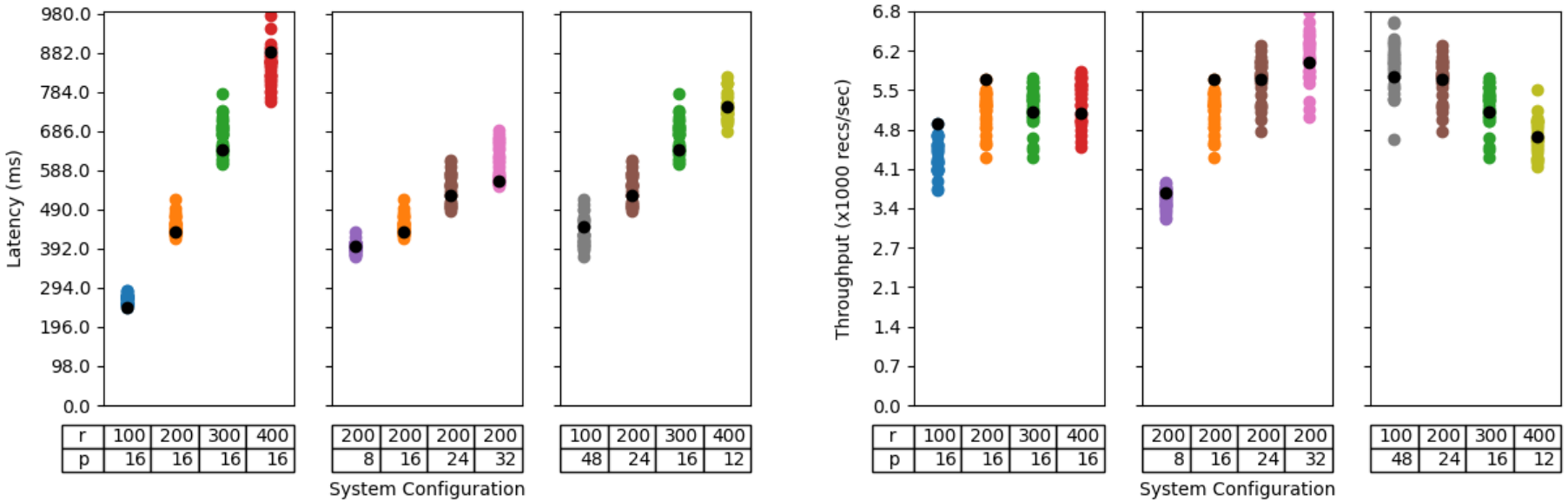
Name	Description	Default
supervisor.slots.ports	Number of Worker processes per machine	4
topology.workers	Number of Worker processes for the entire topology	1
parallelism hint (ph)	Number of Executor threads per spout/bolt in topology	1
topology.tasks	Number of tasks per spout or bolt in a topology	ph
topology.worker.receiver.thread.count	Number of tuple receiver threads per worker	1
topology.acker.executors	Number of Acker threads to spawn for the topology	Not set
topology.max.spout.pending	Max number of tuples to be pending on a spout task	Not set
topology.executor.receive.buffer.size	Size of receive queue per Executor	32768
topology.transfer.buffer.size	Size of outbound message (transfer) queue per Worker	1024

Selected Perf-Aware Parameters in Spark Streaming

Name	Description	Default
spark.driver.cores	Number of cores to use for the driver process	1
spark.driver.memory	Amount of memory to use for the driver process	1g
spark.executor.instances	Number of executors to lunch per cluster node	1
spark.executor.cores	Number of cores to use on each executor for running tasks	all
spark.executor.memory	Total amount of memory to use per executor process	1g
spark.shuffle.compress	Whether to compress map output files	false
spark.streaming.receiver.maxRate	Max rate (records/sec) for receiving data per receiver	not set
spark.streaming.blockInterval	Interval at which input data is partitioned into blocks	200ms
batchDuration	Time interval at which data will be divided into batches	1000ms

Impact of Parameter Configurations

r: Records/partition
p: Partitions/second



Scenario: Spark Streaming, 5-node cluster, streaming classification application, real-world data

*L. Odysseos and H. Herodotou. *Exploring System and Machine Learning Performance Interactions when Tuning Distributed Data Stream Applications*. ICDEW 2022

Performance Tuning Problem

- **Application Performance**

$$perf = F(g, d, r, p)$$

g = DAG of operators
d = input data properties
r = cluster resources
p = parameter settings

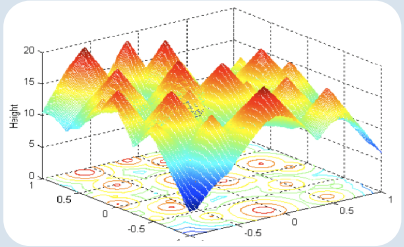
- **Performance Optimization**

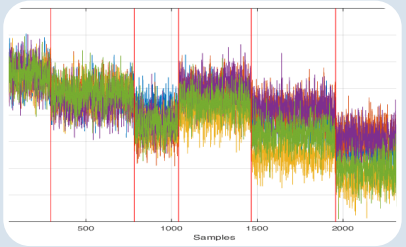
$$p_{opt} = \arg\max_{p \in S} F(g, d, r, p)$$

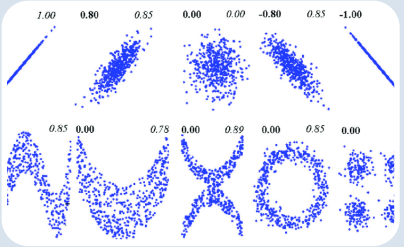
- NP-Hard problem

Goal: Automate the process of configuring and running streaming applications to meet service level objectives

Key Challenges







Large and complex parameter space

Dynamic changes in input data stream

Non-linear impact of applications & hardware

Outline

- Introduction & Motivation
- Method Taxonomy
- Parameter Tuning Approaches
  - Cost modeling
  - Simulation-based
  - Experiment-driven
  - Machine learning
  - Adaptive tuning
- Comparison of Approaches
- Open Problems & Challenges

Method Taxonomy of Tuning Approaches

Cost Modeling	Use cost models & statistics to find optimal settings
Simulation-based	Use simulator to estimate application performance
Experiment-driven	Execute application with different settings iteratively
Machine Learning	Use machine learning to model application performance
Adaptive	Change configurations while application is running

Tuning Method Taxonomy – Cost Modeling

- **Application Performance**

$$perf = F(g, d, r, p)$$

- Use statistical **cost models** to represent F
- Use **optimization algorithm** to find optimal parameter settings

$$B.CPUCost = \sum_{t=1}^{n} B.numInputTuples_t * B.inputCost_t + \sum_{t'=1}^{n'} (B.numOutputTuples_{t,t'} * B.outputCost_{t'})$$

with

$$B.inputCost_t = B.CPURead_t + B.CPUExecute_t + B.isExternalTask * CPUDeserialize_t$$

and

$$B.outputCost_{t'} = \sum_{i=1}^{B.numDest_{t'}} B_i.groupingCost_{t'} + B.hasExternalTask_{t'} * CPUSerialize_{t'}$$

where

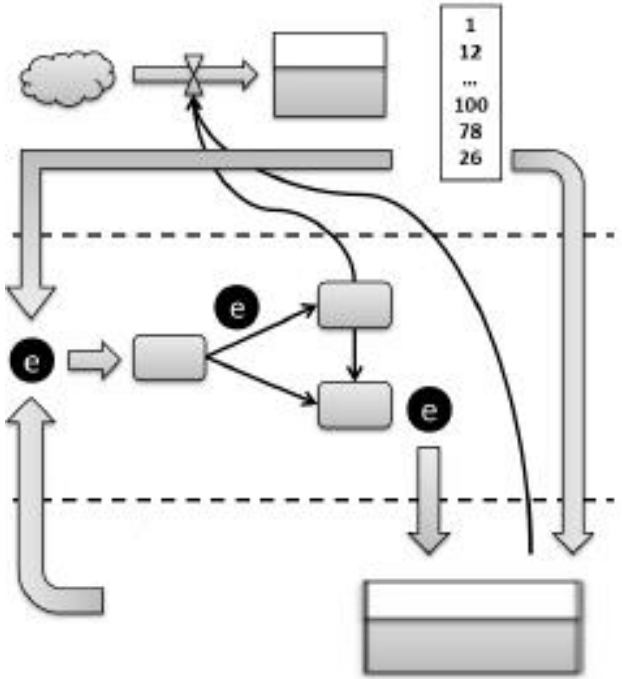
- $B.numInputTuples_t$ is the number of tuples of type t received by the bolt B, which, as was shown before, can be represented as:

$$B.numInputTuples_t = \sum_{d=1}^{m} E_d.numTuples_t$$

Tuning Method Taxonomy – Simulation based

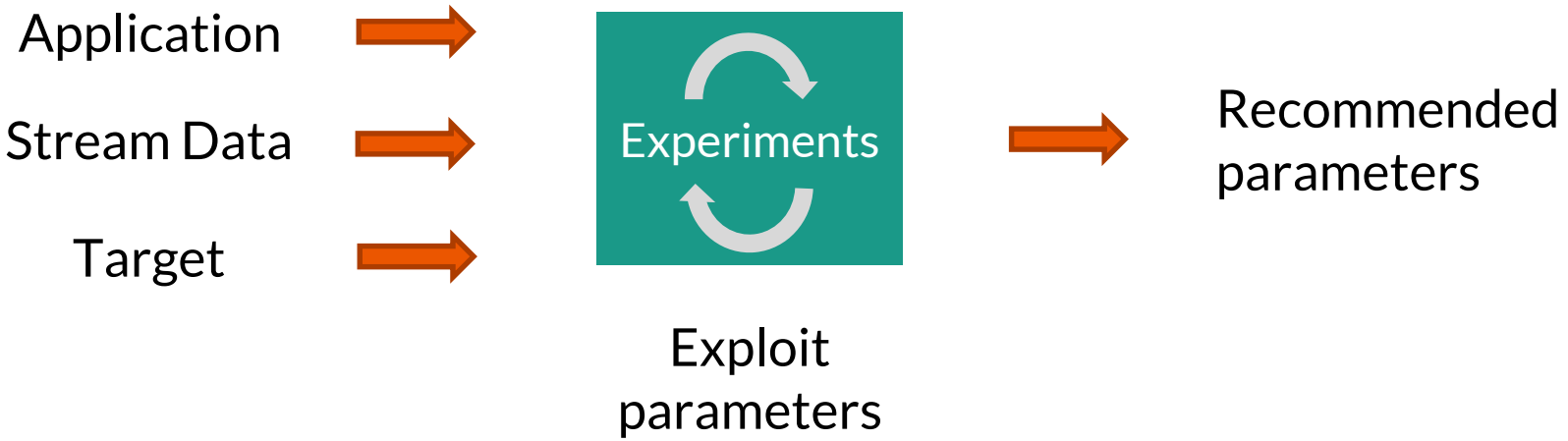
- **Application Performance**
  $perf = F(g, d, r, p)$

- Use modular or complete **simulator** to represent F
- Use **optimization algorithm** to find optimal parameter settings



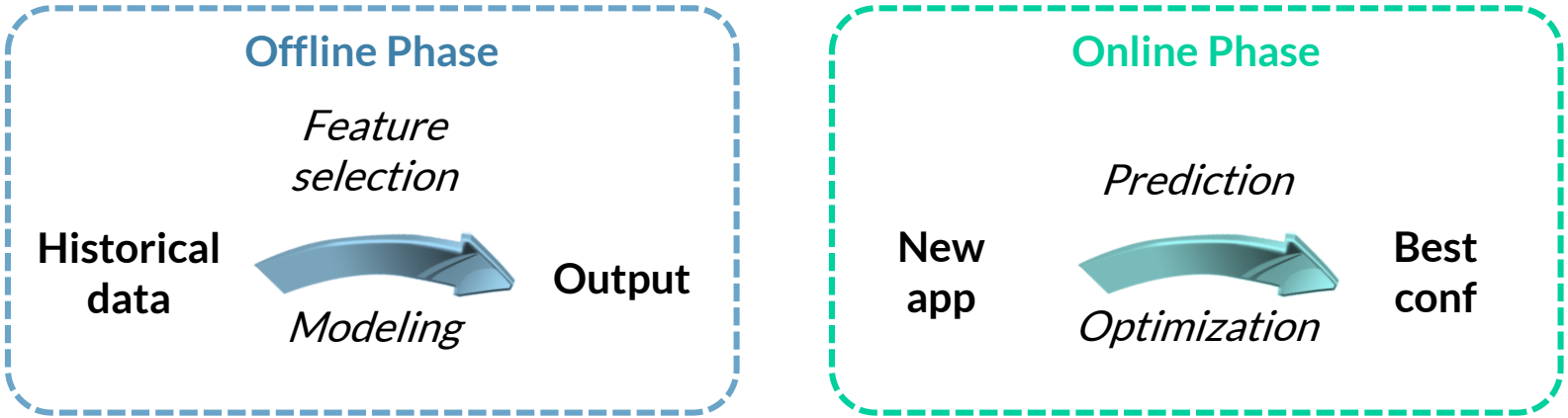
Tuning Method Taxonomy – Experiment-driven

- Execute the experiments **repeatedly** with different parameter settings, guided by a **search algorithm**



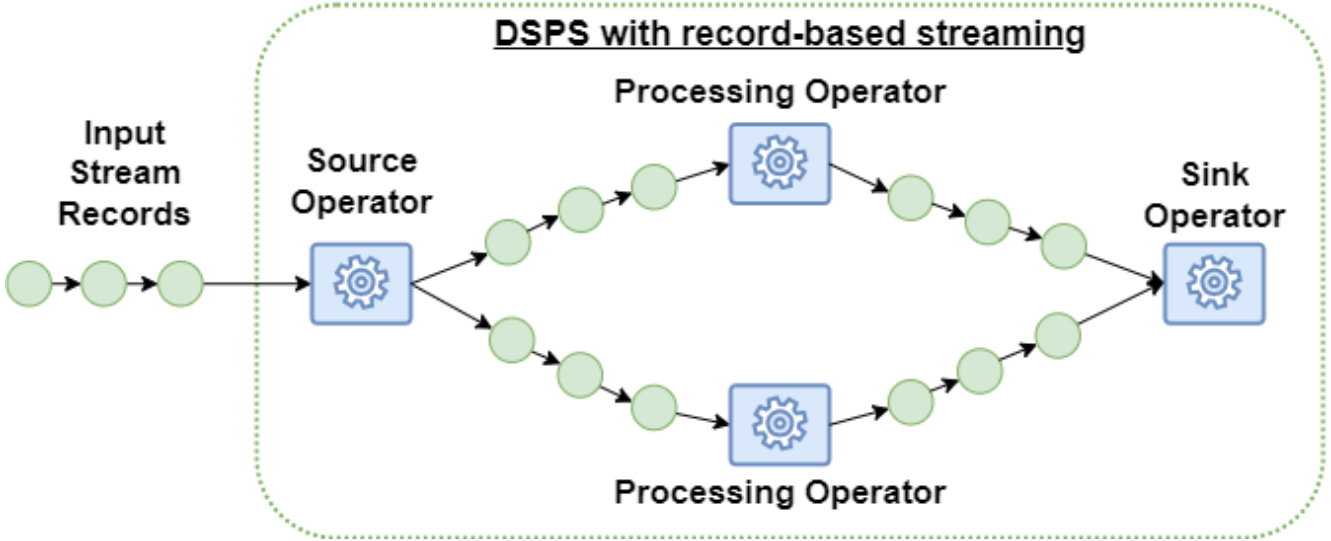
Tuning Method Taxonomy – Machine Learning

- Employ **machine learning** methods to establish performance models F



Tuning Method Taxonomy – Adaptive

- **Track execution** of an application and change its configuration in an **online fashion** in order to improve performance

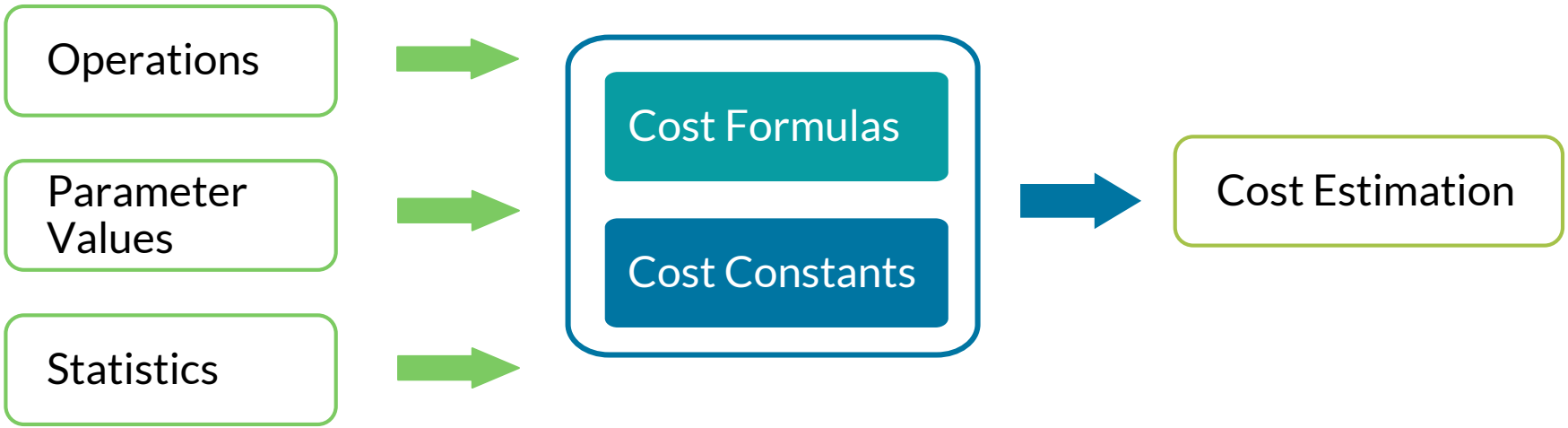


Outline

- Introduction & Motivation
- Method Taxonomy
- Parameter Tuning Approaches
  - Cost modeling
  - Simulation-based
  - Experiment-driven
  - Machine learning
  - Adaptive tuning
- Comparison of Approaches
- Open Problems & Challenges

Cost Modeling

Build performance prediction models by using statistical cost functions



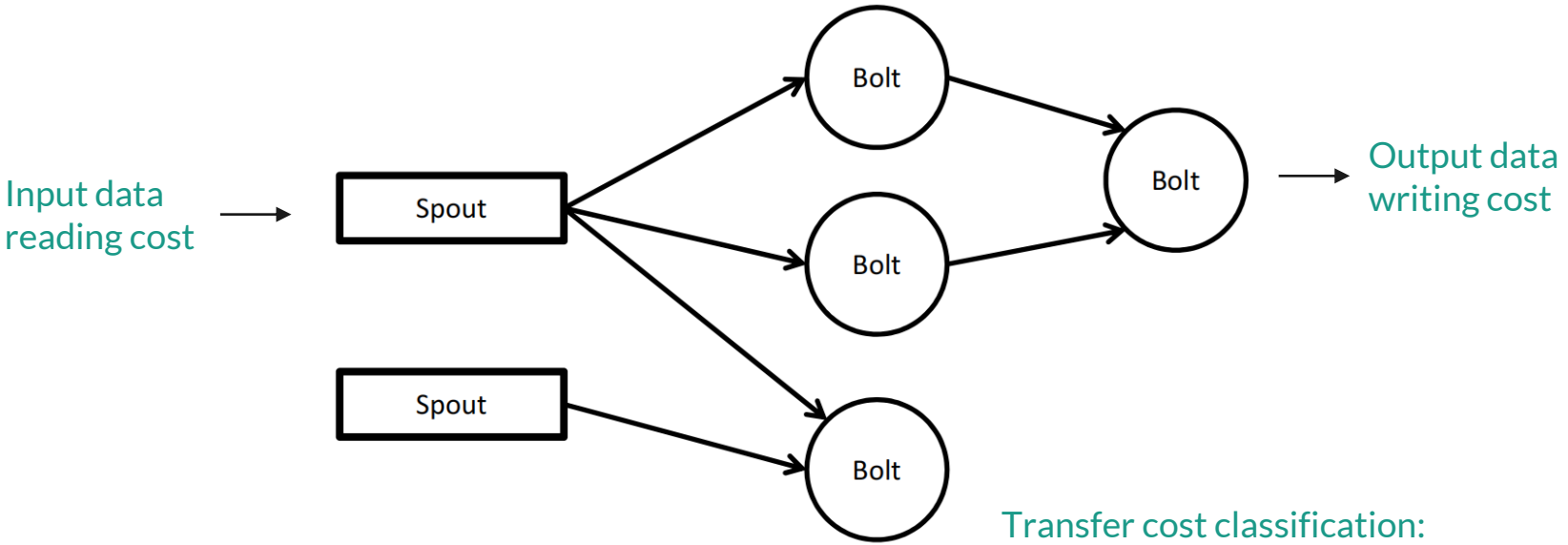
Profiling → Prediction → Optimization

Cost Modeling – Comparison

Platform	Paper	Profiling	Prediction	Optimization
Storm	1. Sax et al.	Performance metrics	Analytical model	Direct algorithm
Storm	2. Bedini et al.	Performance metrics	Fine-grained cost model	no
Heron	3. Trevor	Performance metrics	Linear models	no
Heron	4. Caladrius	Performance metrics	Cost models	Topological sorting

1. ICDEW'13 - Performance Optimization for Distributed Intra-Node-Parallel Streaming Systems
2. ACM/SPEC'13 - Modeling Performance of a Parallel Streaming Engine: Bridging Theory and Costs
3. CoRR'18 - Trevor: Automatic Configuration and Scaling of Stream Processing Pipelines
4. ICDE'19 - Caladrius: A Performance Modelling Service for Distributed Stream Processing Systems

Details of a Paper – Data Flow Cost



Transfer cost classification:

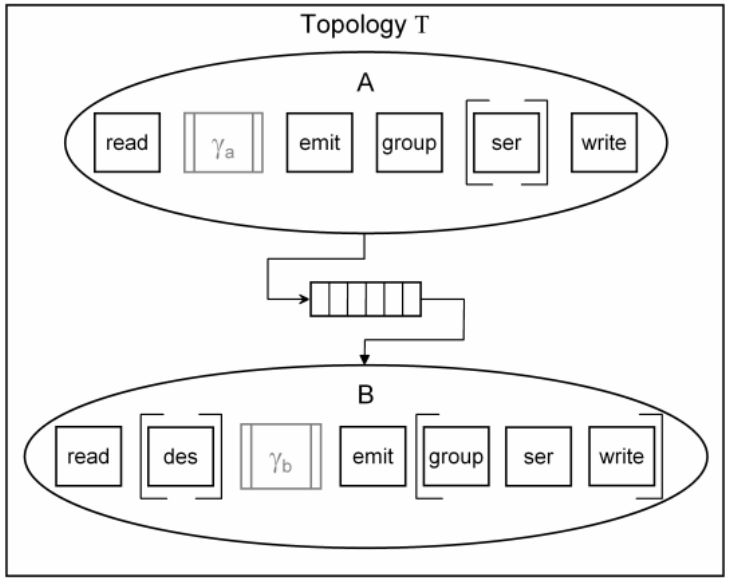
- Within same JVM
- Within same node but diff JVM
- With different nodes

ICPE'13 Modeling Performance of a Parallel Streaming Engine: Bridging Theory and Costs

Details of a Paper – Processing Cost

Spout and Bolt Cost:

- Read: read a source
- Destination: de-serialize
- Transform(γ): row data => storm tuple
- Emit: transfer cost
- Group: group by (to all, to some, to one)
- Serialize: for efficient transmit
- Write: output result



ICPE'13 Modeling Performance of a Parallel Streaming Engine: Bridging Theory and Costs

Cost Modeling Approach: Pros & Cons

- Very efficient for predicting performance
- Good accuracy in many (not complex) scenarios

Pros

- Hard to capture complexity of system internals & pluggable components (e.g., schedulers)
- Models often based on simplified assumptions
- Not effective on heterogeneous clusters

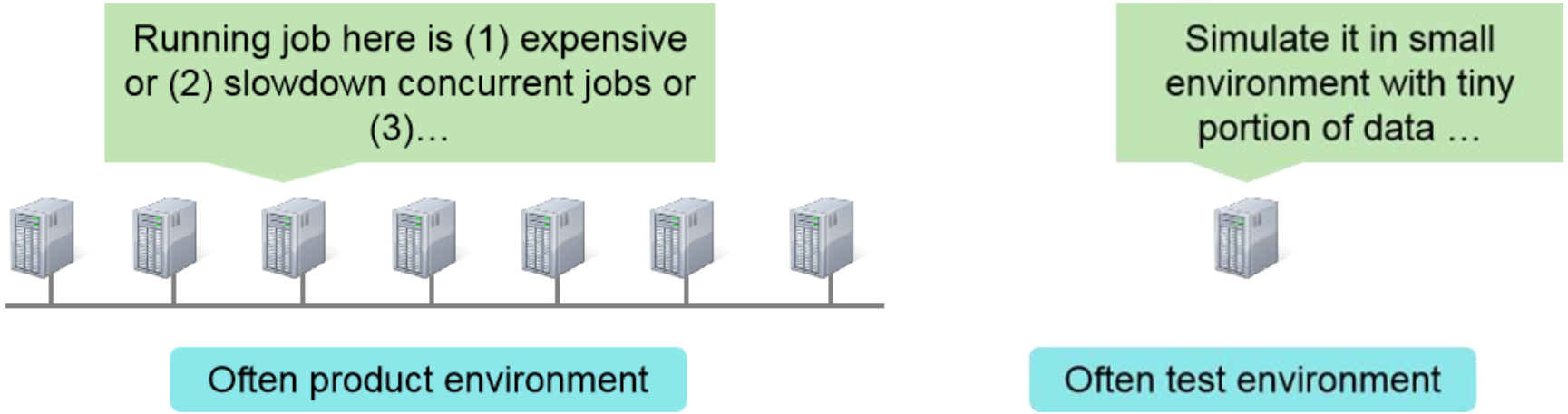
Cons

Outline

- Introduction & Motivation
- Method Taxonomy
- Parameter Tuning Approaches
  - Cost modeling
  - Simulation-based
  - Experiment-driven
  - Machine learning
  - Adaptive tuning
- Comparison of Approaches
- Open Problems & Challenges

Simulation-based

Build performance models based on modular or complete simulation



Simulation-based – Comparison

Platform	Simulator	System execution	System schedule	Configuration parameters
Storm	1. CEPSim	Task sub level	yes	Only basic ones
Storm	2. Requeno et al.	Task level	yes	Many
Spark	3. Kroß et al.	Task level	yes	Only basic ones
Spark	4. SSP	Task level	FIFO	Only basic ones

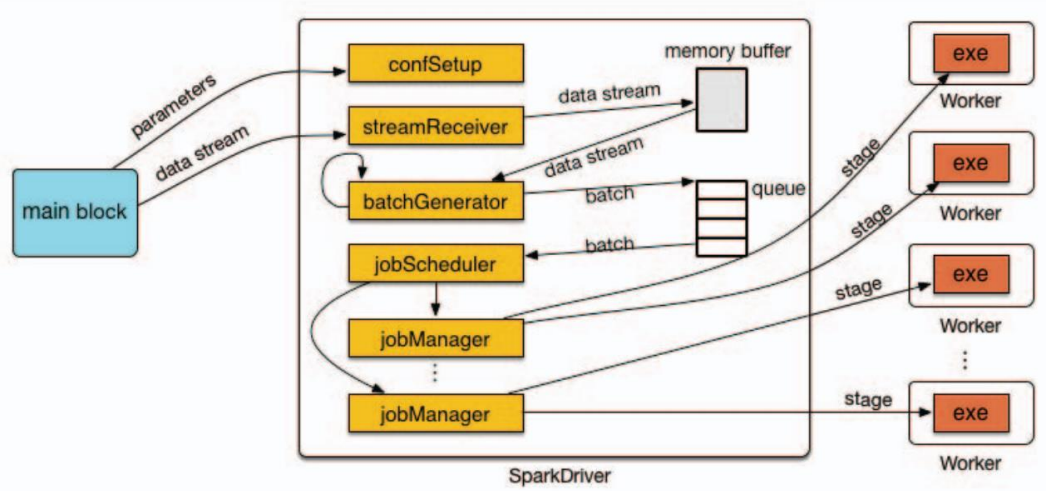
1. FCUS'16 - CEPSim: Modelling and simulation of Complex Event Processing systems in cloud environments
2. IRI'17 - Performance Analysis of Apache Storm Applications Using Stochastic Petri Nets
3. MASCOTS'17 - Model-based Performance Evaluation of Batch and Stream Applications for Big Data
4. AINA'18 - Modeling and Simulation of Spark Streaming

Details of a Paper

Simulation of a real run

- Workload
- Cost of a stage
- #worker nodes
- Resource of each node
- Batch interval
- Data pattern
- Concurrency

Not a real run



Simulation-based Approach: Pros & Cons

- High accuracy in simulating dynamic system behaviors
- Efficient for predicting fine-grained performance

Pros

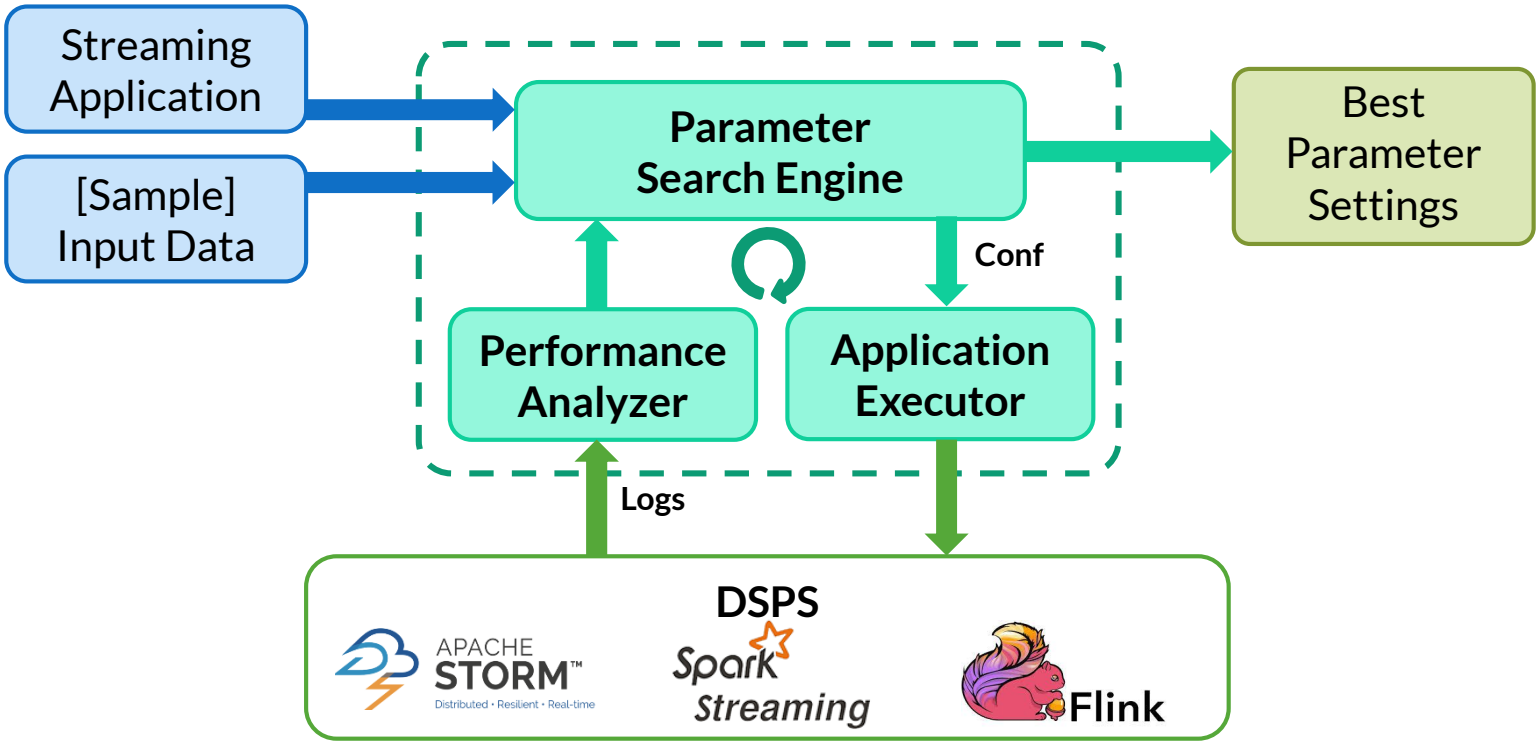
- Hard to comprehensively simulate complex internal dynamics
- Unable to capture dynamic cluster utilization
- Not very efficient for finding optimal settings

Cons

Outline

- Introduction & Motivation
- Method Taxonomy
- Parameter Tuning Approaches
  - Cost modeling
  - Simulation-based
  - Experiment-driven
  - Machine learning
  - Adaptive tuning
- Comparison of Approaches
- Open Problems & Challenges

Experiment-driven



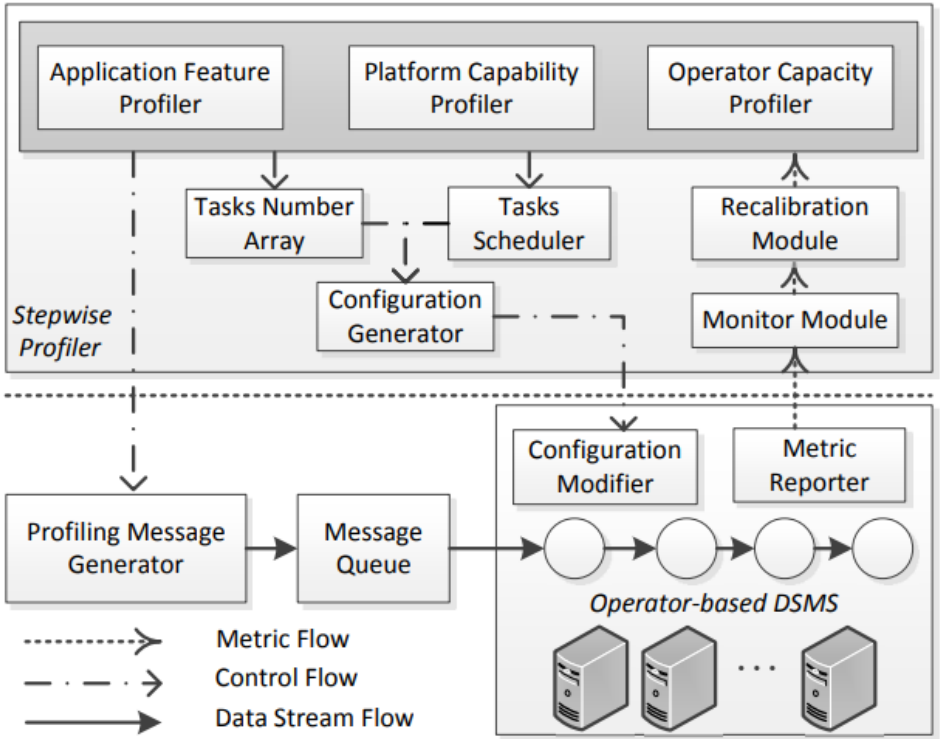
Experiment-driven – Comparison

Platform	Papers	Algorithm	Optimization
Storm	1. BO4CO	Gaussian Processes	Latin Hypercube Sampling (LHS)
Storm	2. Fischer et al.	Bayesian	/
Storm	3. Bilal et al.	Hill Climbing	Latin Hypercube Sampling (LHS)
Storm	4. Liu et al.	Trial-and-error	Stepwise profiling

1. MASCOTS'16 - An Uncertainty-aware Approach to Optimal Configuration of Stream Processing Systems
2. CLUSTER'15 - Machines Tuning Machines: Configuring Distributed Stream Processors with Bayesian Optimization
3. SoCC'17 - Towards Automatic Parameter Tuning of Stream Processing Systems
4. TAAS'18 - A Stepwise Auto-Profiling Method for Performance Optimization of Streaming Applications

Details of a Paper

- Profiling
  - Application feature
  - Platform capability
  - Operator capacity



TAAS'18 A Stepwise Auto-Profiling Method for Performance Optimization of Streaming Applications

Experiment-driven Approach: Pros & Cons

Pros

- Finds good settings based on real test runs on real systems
- Works across different system versions and hardware

Cons

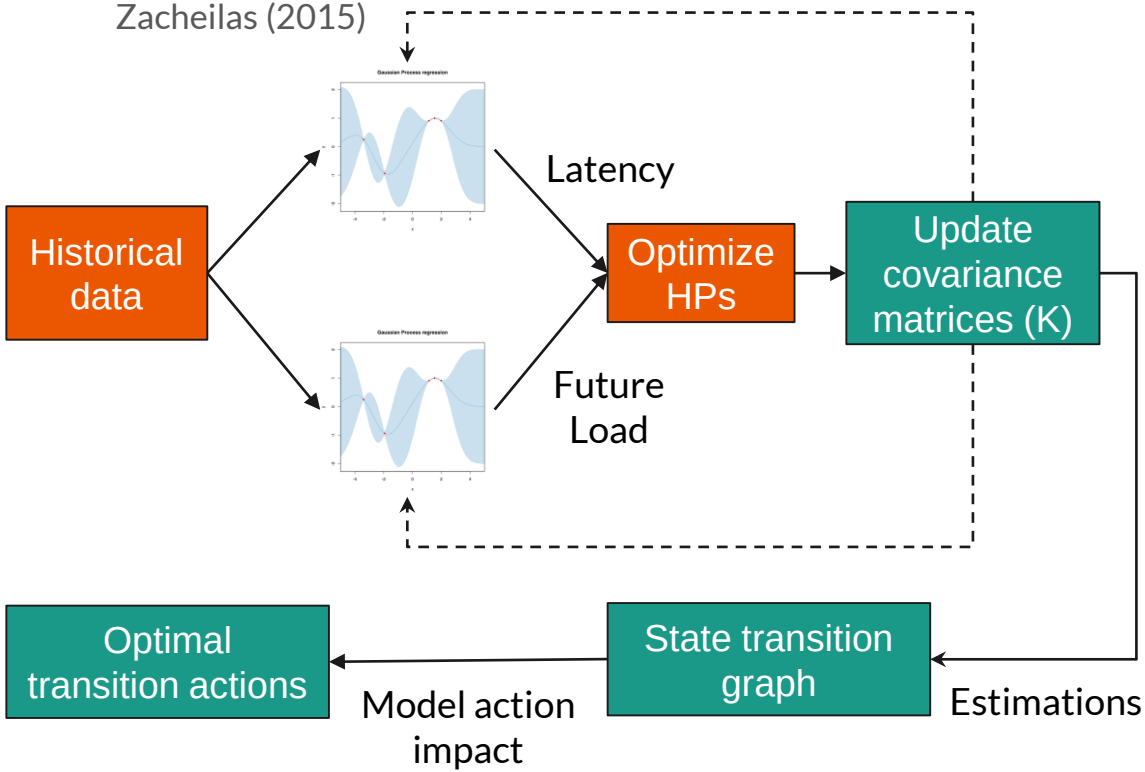
- Very time consuming as it requires multiple actual runs
- Not cost effective for ad-hoc analytics applications

Outline

- Introduction & Motivation
- Method Taxonomy
- Parameter Tuning Approaches
  - Cost modeling
  - Simulation-based
  - Experiment-driven
  - Machine learning
  - Adaptive tuning
- Comparison of Approaches
- Open Problems & Challenges

Machine learning

Zacheilas (2015)



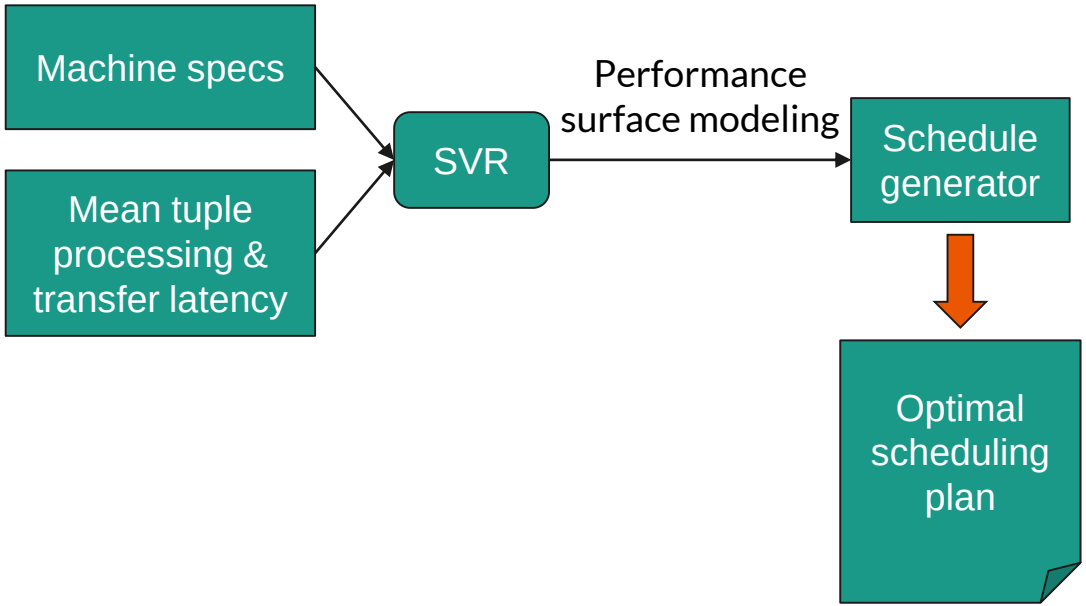
Offline phase

Online phase

- Use historical data to fit 2 GPs:
  - Latency
  - Future Load
- Optimize HPs
- Update covariance matrix of each model
- Construct state transition graph from estimations that models impact of actions (i.e. degree of parallelism)
- Utilize graph to select transition actions that:
  - Minimize cost
  - Maximize performance

Machine learning

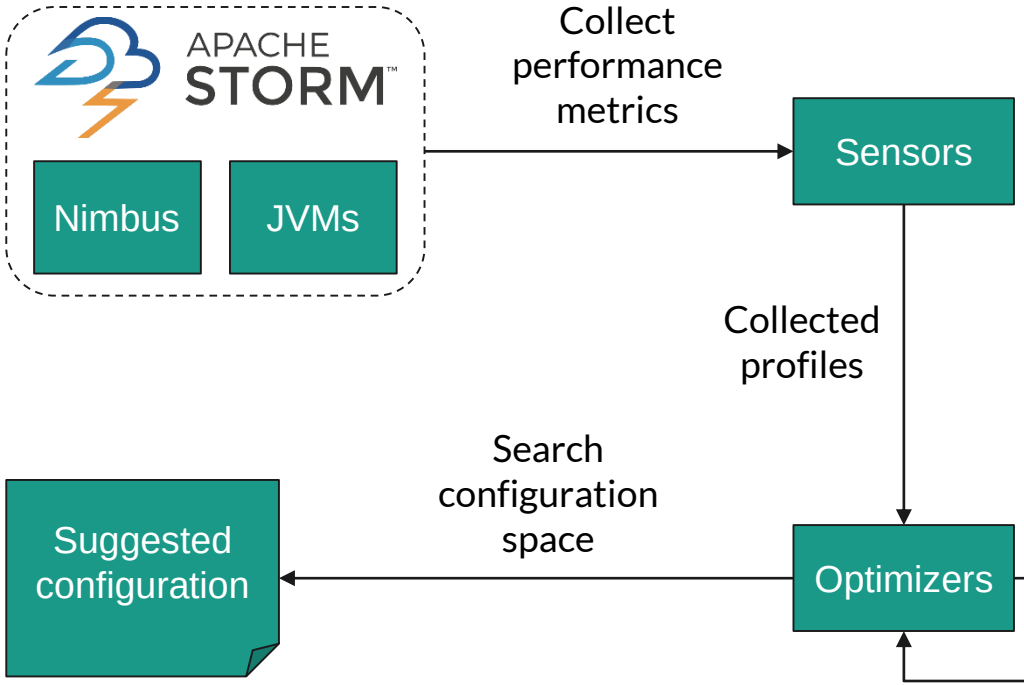
Li (2016)



- Use machine specifications to model workload performance
  - CPU cores
  - Memory
  - Number of threads
- Utilize Support Vector Regression (SVR) to model performance surface
  - Mean tuple processing latency
  - Mean tuple transfer latency among tasks
- Use heuristic search to identify a scheduling plan with the optimal degree of parallelism

Machine learning

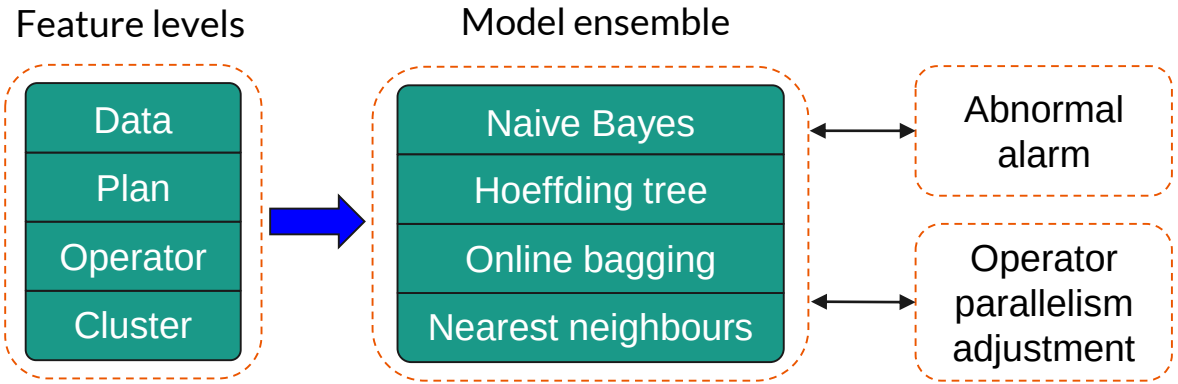
Trotter (2017)



- Two main components:
  - Sensors collect performance metrics from Nimbus and JVMs
  - Optimizers (either genetic algorithms or Bayesian optimization) analyze the collected profiles to search the configuration space
- Extended work (2019):
  - Used a classifier to discard possibly bad configurations (i.e. under 80% of max throughput)

Machine learning

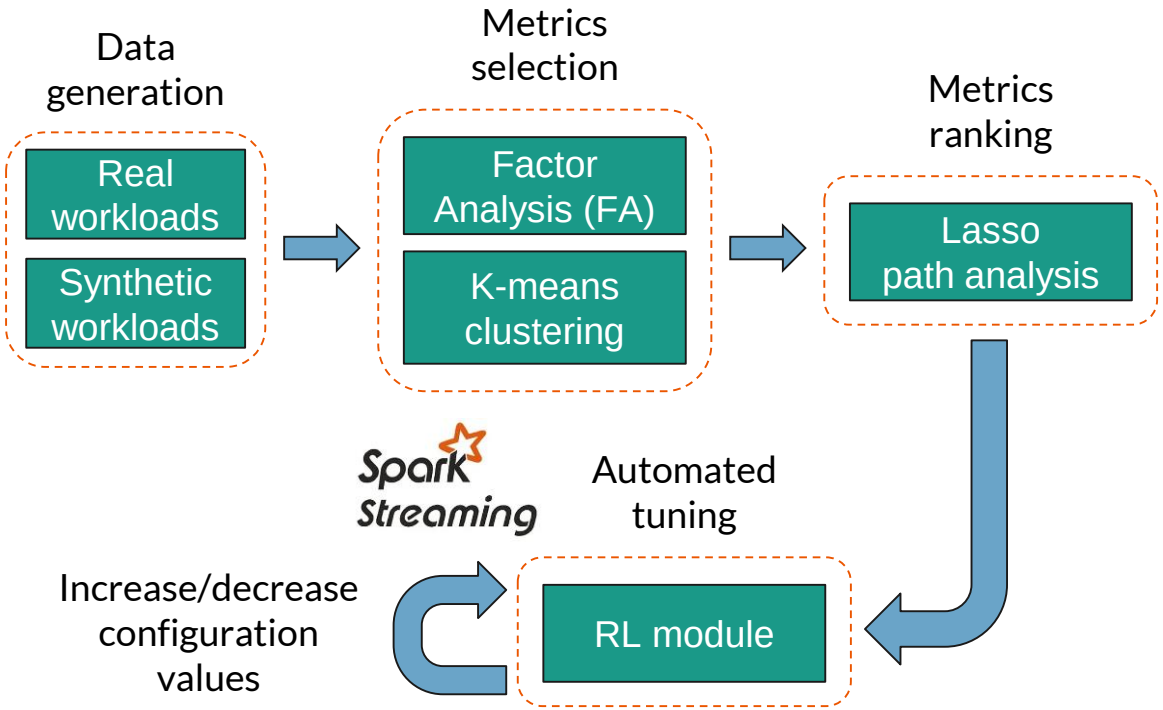
Wang (2017)



- OrientStream: A framework that exploits incremental ML for modeling and predicting resource usage in DSPEs
- Models various features at different levels
  - Data
  - Plan
  - Operator
  - Cluster
- Trains an ensemble of 4 models
  - Naive Bayes
  - Hoeffding tree
  - Online bagging
  - Nearest neighbours
- Detects and discards outliers in training data
- Automatically adjust operator parallelism based on thresholds to increase performance

Machine learning

Vaquero (2018)



- An approach that performs auto-tuning on Spark Streaming workloads
- Data generation
  - Real
  - Synthetic
- Metrics selection
  - Factor Analysis for configuration parameter correlation and importance
  - K-means for clustering parameters into meaningful groups
- Metrics Ranking
  - Lasso path analysis to rank parameter impact on performance
- Automated tuning
  - RL module explores and selects configurations

Machine Learning Approach: Pros & Cons

- Ability to capture complex system dynamics
- Independence from system internals and hardware
- Learning based on real observations of system performance

Pros

- Requires large training sets, which are expensive to collect
- Training from history logs leads to data under-fitting
- Typically low accuracy for unseen analytics applications

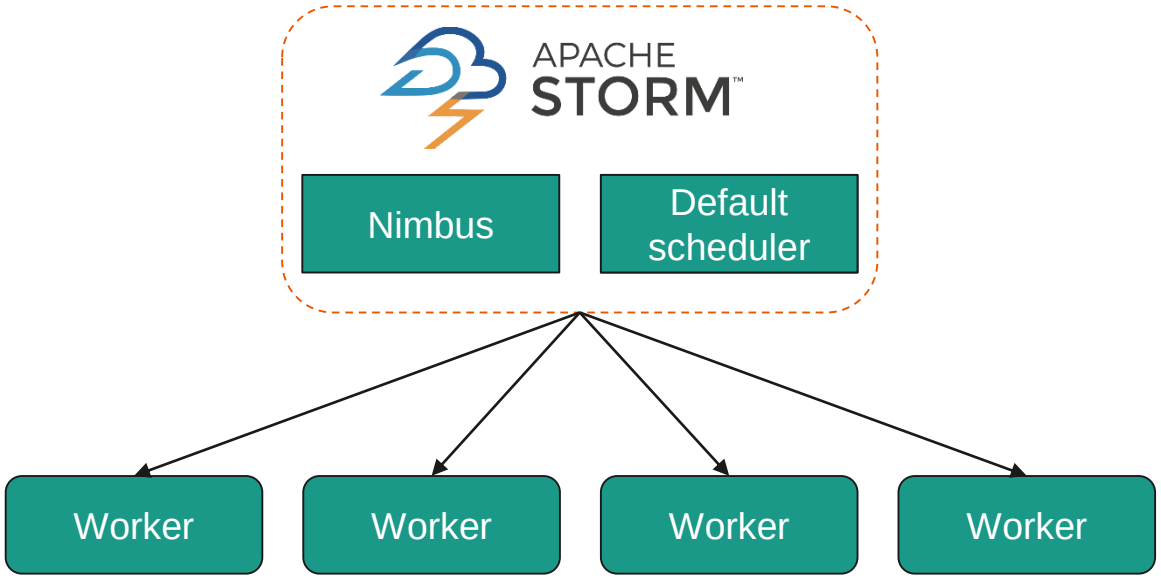
Cons

Outline

- Introduction & Motivation
- Method Taxonomy
- Parameter Tuning Approaches
  - Cost modeling
  - Simulation-based
  - Experiment-driven
  - Machine learning
  - Adaptive tuning
- Comparison of Approaches
- Open Problems & Challenges

Adaptive tuning

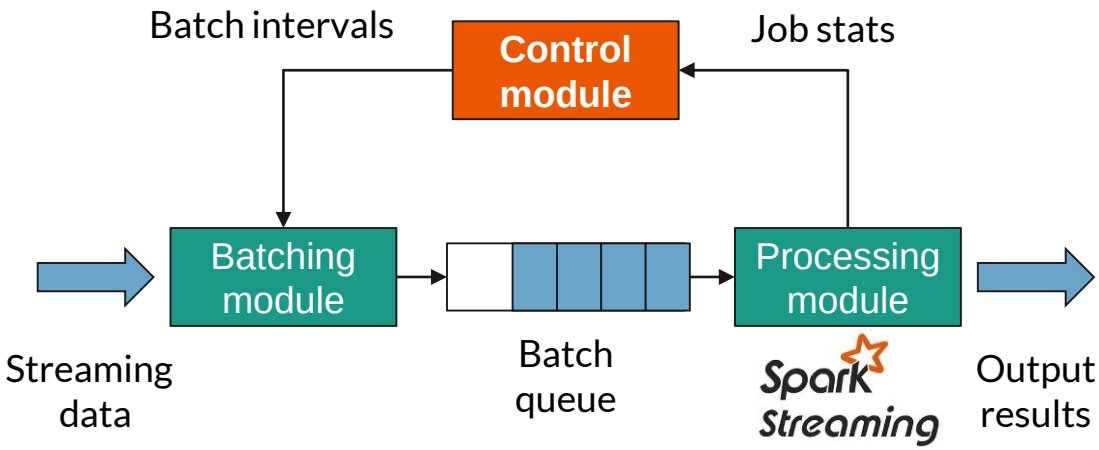
Xu (2014)



- T-Storm: An extension of Apache Storm which mainly concerns the replacement of the default scheduler to increase throughput
- The traffic-aware scheduler aims at minimizing the traffic between nodes and processes
- Occasionally determine the number of workers to use for each topology
  - Assign/re-assign tasks dynamically
- Consolidation: Minimize the number of worker nodes as much as possible

Adaptive tuning

Das (2014)

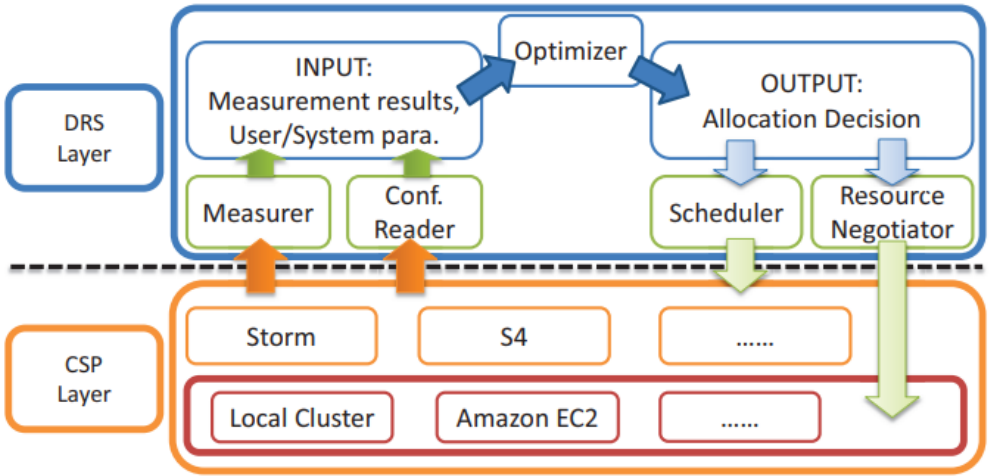


- Batch size is considered one of the most important tuning parameters in DSPSs
- This work focuses on dynamically adapting batch size in order to minimize latency
- The control module collects job stats to learn system behavior
- The batching module requests batch intervals and generates batches accordingly

Key: Keep batch processing time less than the batch interval

Adaptive tuning

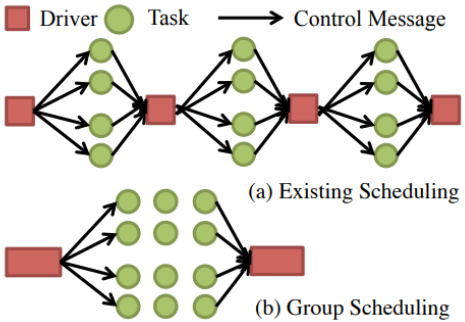
Fu (2015)

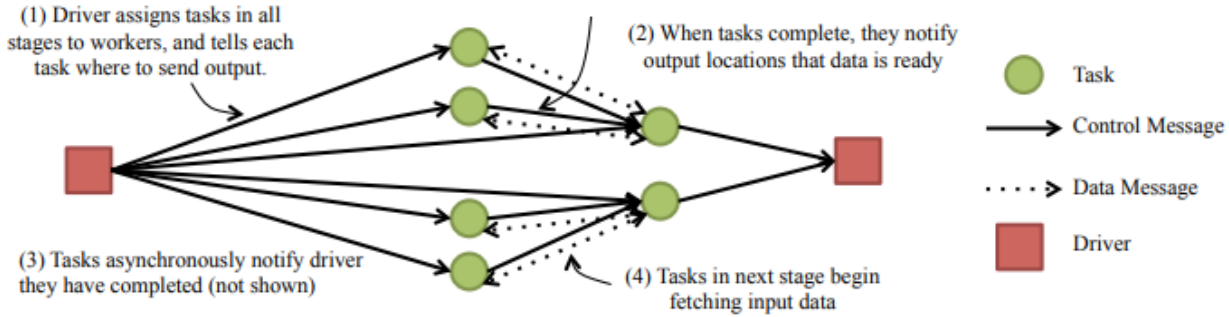


- DRS: Dynamic Resource Scheduler module
- Dynamically assigns resources to workloads by predicting them using Jackson networks from the queueing theory
- Employ a greedy algorithm for finding an optimal scheduling plan

Adaptive tuning

Venkataraman (2017)

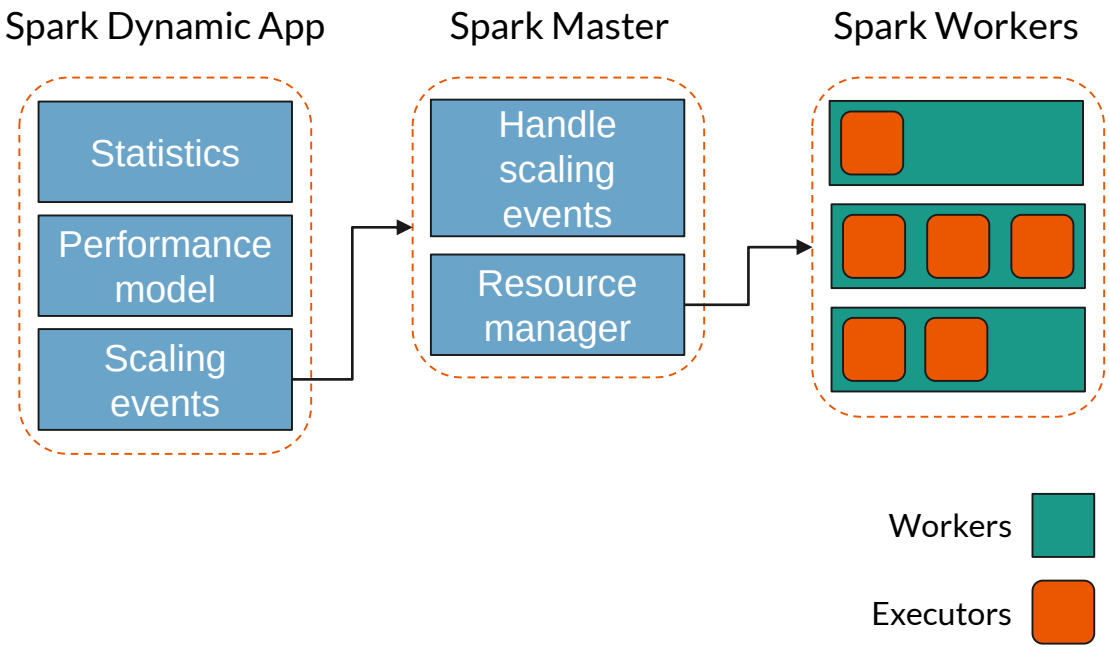




- Drizzle: A system that decouples the coordination interval from the processing interval in Spark streaming that improves:
  - Fault tolerance
  - Adaptability
- Group scheduling: Mitigates scheduling bottlenecks by enabling concurrent batch processing in tasks
- Pre-scheduling: Enable data exchange between executors without contacting the driver
- Optimizations within and across batches

Adaptive tuning

Petrov (2018)



- Collect performance statistics and utilization metrics
- Performance modeling
- Decide whether and how to scale current application to maximize throughput by adding/removing
  - Workers
  - Executors

Adaptive Approach: Pros & Cons

- Finds good settings based on actual task runs
- Able to adjust to dynamic runtime status
- Works well for ad-hoc streaming applications

Pros

- Inappropriate configuration can cause issues (e.g., stragglers)
- Neglects efficient resource utilization in the system

Cons

Outline

- Introduction & Motivation
- Method Taxonomy
- Parameter Tuning Approaches
  - Cost modeling
  - Simulation-based
  - Experiment-driven
  - Machine learning
  - Adaptive tuning
- Comparison of Approaches
- Open Problems & Challenges

Comparison of Approaches

Feature	Cost modeling	Simulation	Experiment-driven	Machine learning	Adaptive
Key modeling technique	cost functions	simulation	search algorithms	ML models	mixed
Number of parameters modeled	some	some	many	many	some
System understanding	strong	strong	light	no	strong
Need for history logs	light	light	strong	strong	light
Need for data input stats	light	light	no	strong	light
Real tests to run	some	no	yes	yes	yes
Time to build model	efficient	medium	slow	slow	medium
Prediction accuracy	medium	medium	medium	high	medium
Adapt to workload	light	light	no	no	adaptive
Adapt to system changes	no	light	no	adaptive	light

Which Approach to Choose

Available history logs:

- Yes → Machine learning
- No → Cost modeling

Workload frequent Changes:

- Change more quickly → Adaptive
- Comparatively stable → Simulation or cost modeling

Project deadline:

- More time for delivery → Experiment-driven or machine learning
- Less time for delivery → Cost modeling

Parameter number:

- Less parameters → Cost modeling or simulation
- More parameters → Experiment-driven or machine learning

Open Problems & Challenges

Cluster heterogeneity

Cloud computing

Edge computing

Cluster Heterogeneity

- Modern hardware such as NVRAM, GPUs, and FPGAs also calls for investigation, including its impact on the performance of streaming applications

Challenges:

- How to configure the number of GPU and CPU cores with the cluster heterogeneity?
- How to model the performance of the new systems with modern hardware?

Cloud Computing

- The proliferation of the Cloud led to new cloud-based data streaming engines such as Amazon Kinesis and Confluent Cloud

Challenges:

- How to manage performance interactions among multiple tenants?
- How to ensure high scalability and elasticity by dynamically adding more resources?
- How to navigate the tradeoffs between high performance and fault tolerance?

Edge Computing

- A recent trend in stream-based computing, especially in the Internet-of-Things (IoT) domain, involves decentralized processing at the source of the data (i.e., at the edge)

Challenges:

- How to alleviate the pressure of computation at the edge?
- How to manage devices with limited capabilities at the edge?
- How to perform the application reconfigurations at the edge?

Conclusion

- Distributed data stream processing systems (DSPSs) such as Storm, Flink, and Spark Streaming are widely used to process continuous data streams in (near) real-time
- This tutorial offers a comprehensive review of the state-of-the-art automatic performance tuning approaches for DSPSs
- Five categories: Cost modeling, Simulation-based, Experiment-driven, Machine learning, and Adaptive

Thanks!

Herodotos Herodotou
Lambros Odysseos
Yuxing Chen
Jiaheng Lu

Cyprus University of Technology

Tencent

UNIVERSITY OF HELSINKI

https://www2.helsinki.fi/en/researchgroups/unified-database-management-systems-udbms/tutorial/icde-2022-tutorial

References (Introduction)

- Herodotos Herodotou, Lambros Odysseos, Yuxing Chen, and Jiaheng Lu. Automatic Performance Tuning for Distributed Data Stream Processing Systems. In Proc. of the 38th IEEE Intl. Conf. on Data Engineering (ICDE '22), 2022.
- Lambros Odysseos and Herodotos Herodotou. Exploring System and Machine Learning Performance Interactions when Tuning Distributed Data Stream Applications. In Proc. of the 38th IEEE Intl. Conf. on Data Engineering Workshops (ICDEW '22), 2022.
- Herodotos Herodotou, Yuxing Chen, and Jiaheng Lu. A Survey on Automatic Parameter Tuning for Big Data Processing Systems. ACM Computing Surveys (CSur), Vol. 53, No. 2, Article 43, 37 pages, April 2020.
- Jiaheng Lu, Yuxing Chen, Herodotos Herodotou, and Shivnath Babu. Speedup Your Analytics: Automatic Parameter Tuning for Databases and Big Data Systems. Proc. of VLDB Endowment (PVLDB), Vol. 12, No. 12, August 2019.

References (Cost Modeling)

- Sax, Matthias J., Malu Castellanos, Qiming Chen, and Meichun Hsu. "Performance optimization for distributed intra-node-parallel streaming systems." In 2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW'13), pp. 62-69. IEEE, 2013.
- Bedini, Ivan, Sherif Sakr, Bart Theeten, Alessandra Sala, and Peter Cogan. "Modeling performance of a parallel streaming engine: bridging theory and costs." In Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE'13), pp. 173-184. 2013.
- Bansal, Manu, Eyal Cidon, Arjun Balasingam, Aditya Gudipati, Christos Kozyrakis, and Sachin Katti. "Trevor: Automatic configuration and scaling of stream processing pipelines." (CoRR'18) arXiv preprint arXiv:1812.09442 (2018).
- Kalim, Faria, Thomas Cooper, Huijun Wu, Yao Li, Ning Wang, Neng Lu, Maosong Fu et al. "Caladrius: A performance modelling service for distributed stream processing systems." In 2019 IEEE 35th International Conference on Data Engineering (ICDE'19), pp. 1886-1897. IEEE, 2019.

References (Simulation-based)

- Higashino, Wilson A., Miriam AM Capretz, and Luiz F. Bittencourt. "CEPSim: Modelling and simulation of Complex Event Processing systems in cloud environments." Future Generation Computer Systems 65 (FCUS'16) (2016): 122-139.
- Requeno, Jose-Ignacio, José Merseguer, and Simona Bernardi. "Performance analysis of apache storm applications using stochastic petri nets." In 2017 IEEE International Conference on Information Reuse and Integration (IRI'17), pp. 411-418. IEEE, 2017.
- Kroß, Johannes, and Helmut Krcmar. "Model-based performance evaluation of batch and stream applications for big data." In 2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'17), pp. 80-86. IEEE, 2017.
- Lin, Jia-Chun, Ming-Chang Lee, Ingrid Chieh Yu, and Einar Broch Johnsen. "Modeling and simulation of spark streaming." In 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA'18), pp. 407-413. IEEE, 2018.

References (Experiment-driven)

- Jamshidi, Pooyan, and Giuliano Casale. "An uncertainty-aware approach to optimal configuration of stream processing systems." In 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'16), pp. 39-48. IEEE, 2016.
- Fischer, Lorenz, Shen Gao, and Abraham Bernstein. "Machines tuning machines: Configuring distributed stream processors with bayesian optimization." In 2015 IEEE International conference on cluster computing (CLUSTER'15), pp. 22-31. IEEE, 2015.
- Bilal, Muhammad, and Marco Canini. "Towards automatic parameter tuning of stream processing systems." In Proceedings of the 2017 Symposium on Cloud Computing (SoCC'17), pp. 189-200. 2017.
- Liu, Xunyun, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, Chenhao Qu, and Rajkumar Buyya. "A stepwise auto-profiling method for performance optimization of streaming applications." ACM Transactions on Autonomous and Adaptive Systems (TAAS'18) 12, no. 4 (2017): 1-33.

References (Machine Learning)

- Nikos Zacheilas, Vana Kalogeraki, Nikolaos Zygouras, Nikolaos Panagiotou, and Dimitrios Gunopulos. 2015. Elastic complex event processing exploiting prediction. In Proceedings of the IEEE International Conference on Big Data (Big Data'15). IEEE Computer Society, 213–222.
- Teng Li, Jian Tang, and Jielong Xu. 2016. Performance modeling and predictive scheduling for distributed stream data processing. IEEE Trans. Big Data 2, 4 (2016), 353–364.
- Michael Trotter, Guyue Liu, and Timothy Wood. 2017. Into the storm: Descrying optimal configurations using genetic algorithms and Bayesian optimization. In Proceedings of the IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W'17). IEEE Computer Society, 175–180.
- Michael Trotter, Timothy Wood, and Jinho Hwang. 2019. Forecasting a storm: Divining optimal configurations using genetic algorithms and supervised learning. In Proceedings of the International Conference on Autonomic Computing (ICAC'19). IEEE, 136–146.
- Chunkai Wang, Xiaofeng Meng, Qi Guo, Zujian Weng, and Chen Yang. 2017. Automating characterization deployment in distributed data stream management systems. IEEE Trans. Knowl. Data Eng. 29, 12 (2017), 2669–2681.
- Luis M. Vaquero and Félix Cuadrado. 2018. Auto-tuning distributed stream processing systems using reinforcement learning. CoRR abs/1809.05495 (2018).

References (Adaptive)

- Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. 2014. T-Storm: Traffic-aware online scheduling in Storm. In Proceedings of the International Conference on Distributed Computing Systems (ICDCS'14). IEEE, 535–544.
- Tathagata Das, Yuan Zhong, Ion Stoica, and Scott Shenker. 2014. Adaptive stream processing using dynamic batch sizing. In Proceedings of the 5th ACM Symposium on Cloud Computing (SoCC'14). ACM, 16:1–16:13.
- Tom Z. J. Fu, Jianbing Ding, Richard T. B. Ma, Marianne Winslett, Yin Yang, and Zhenjie Zhang. 2015. DRS: Dynamic resource scheduling for real-time analytics over fast streams. In Proceedings of the International Conference on Distributed Computing Systems (ICDCS'15). IEEE, 411–420.
- Shivaram Venkataraman, Aurojit Panda, Kay Ousterhout, Michael Armbrust, Ali Ghodsi, Michael J. Franklin, Benjamin Recht, and Ion Stoica. 2017. Drizzle: Fast and adaptable stream processing at scale. In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'17). ACM, 374–389.
- Max Petrov, Nikolay Butakov, Denis Nasonov, and Mikhail Melnik. 2018. Adaptive performance model for dynamic scaling apache spark streaming. Procedia Comput. Sci. 136 (2018), 109–117.