

Re-Architecting Apache Spark for Performance Understandability

Kay Ousterhout

Joint work with Christopher Canel, Max Wolff, Sylvia Ratnasamy, Scott Shenker



About Me

PhD candidate at UC Berkeley

Thesis work on performance of large-scale distributed systems

Apache Spark PMC member

About this talk

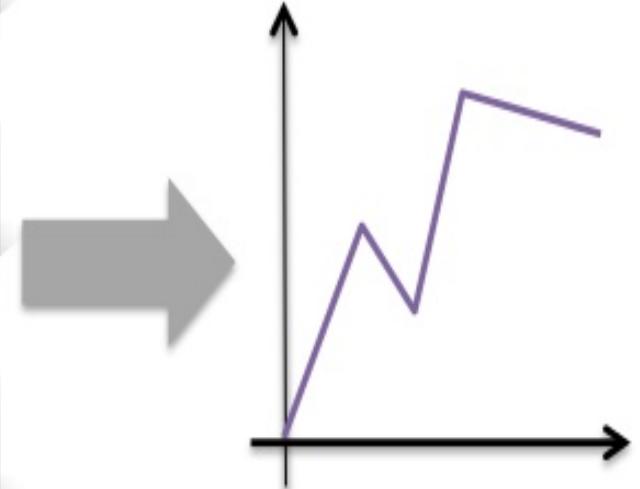
Future architecture for systems like Spark

Implementation is API-compatible with Spark

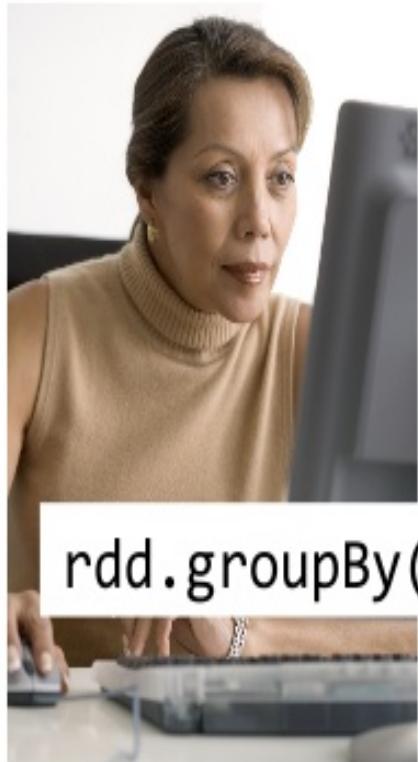
Major change to Spark's internals (~20K lines of code)



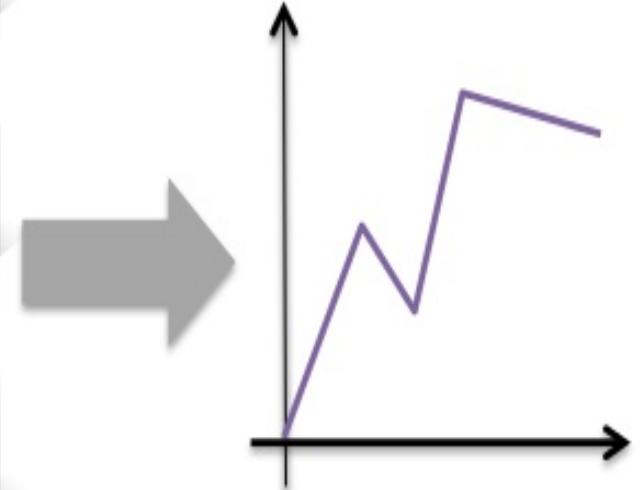
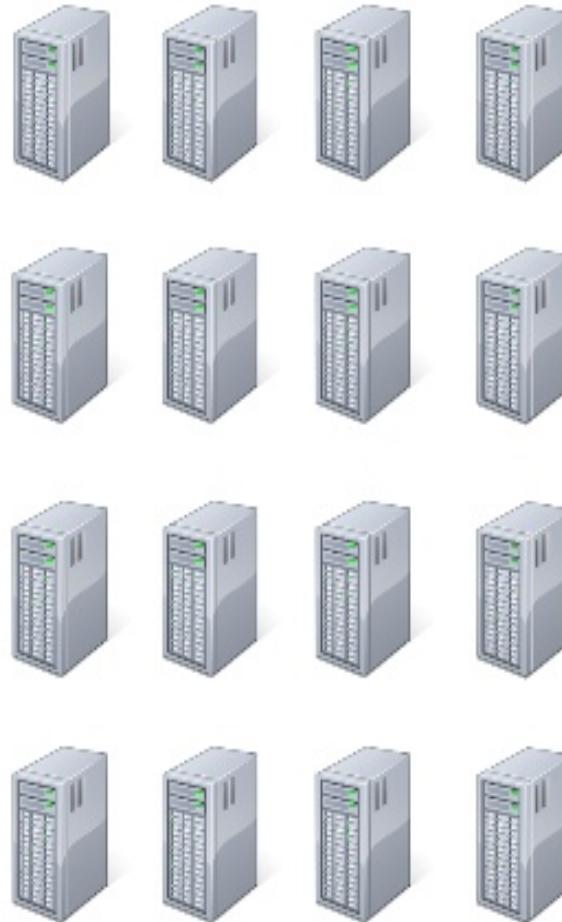
`rdd.groupBy(...)`



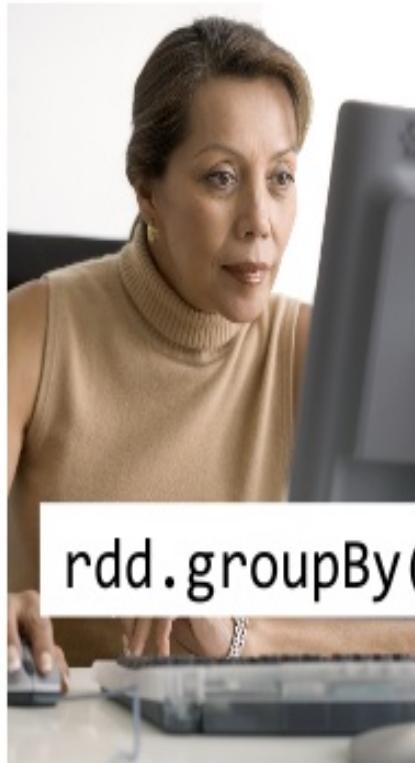
Spark cluster is a black box, runs the job fast



`rdd.groupBy(...)`



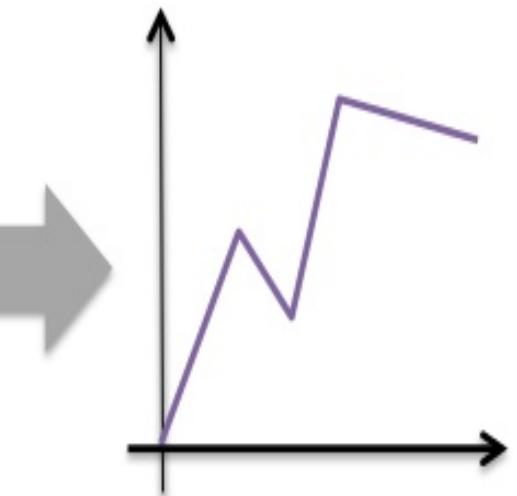
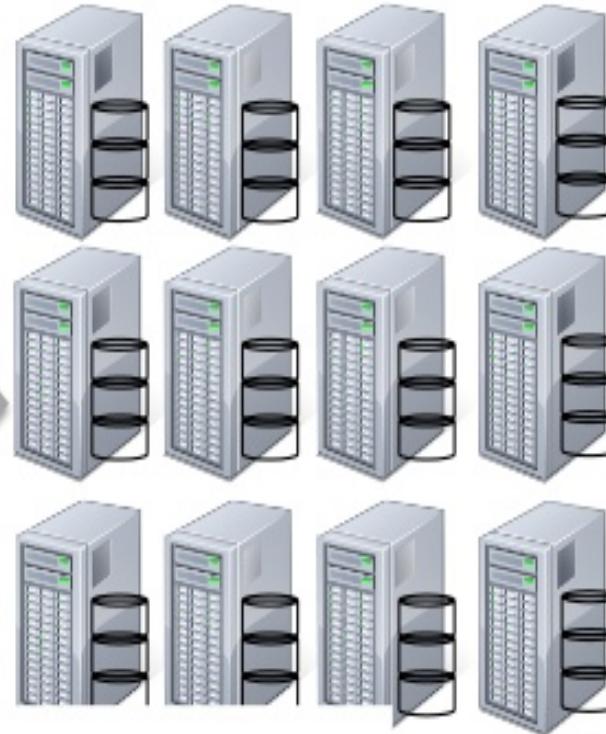
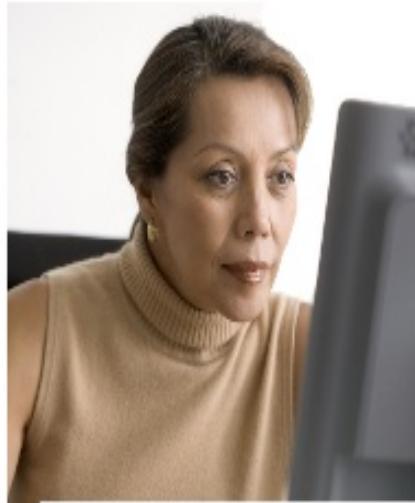
Idealistic view: Spark cluster is a black box, runs the job fast



`rdd.groupBy(...)`



Idealistic view: Spark cluster is a black box, runs the job fast



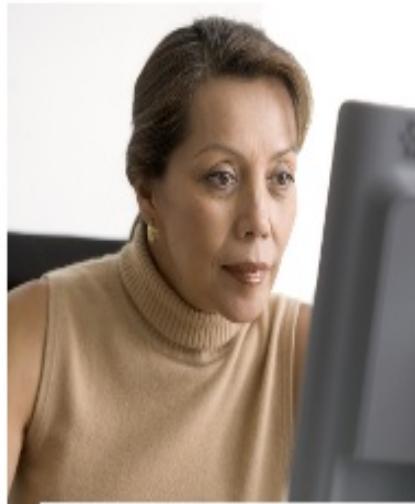
~~rdd.groupBy(...)~~
rdd.reduceByKey(...)

- Configuration:

spark.serializer KryoSerializer
spark.executor.cores 8

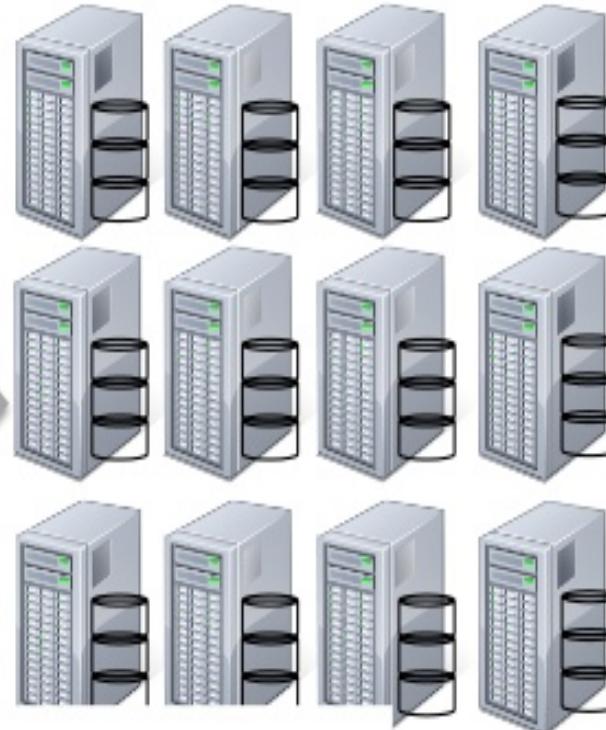


Idealistic view: Spark cluster is a black box, runs the job fast



~~rdd.groupBy(...)~~
~~rdd.reduceByKey(...)~~

- Configuration:
spark.serializer KryoSerializer
spark.executor.cores 8



Realistic view: user uses performance characteristics to tune job configuration, hardware, etc



Users need to be able to reason about performance

~~rdd.groupBy(...)~~
~~rdd.reduceByKey(...)~~

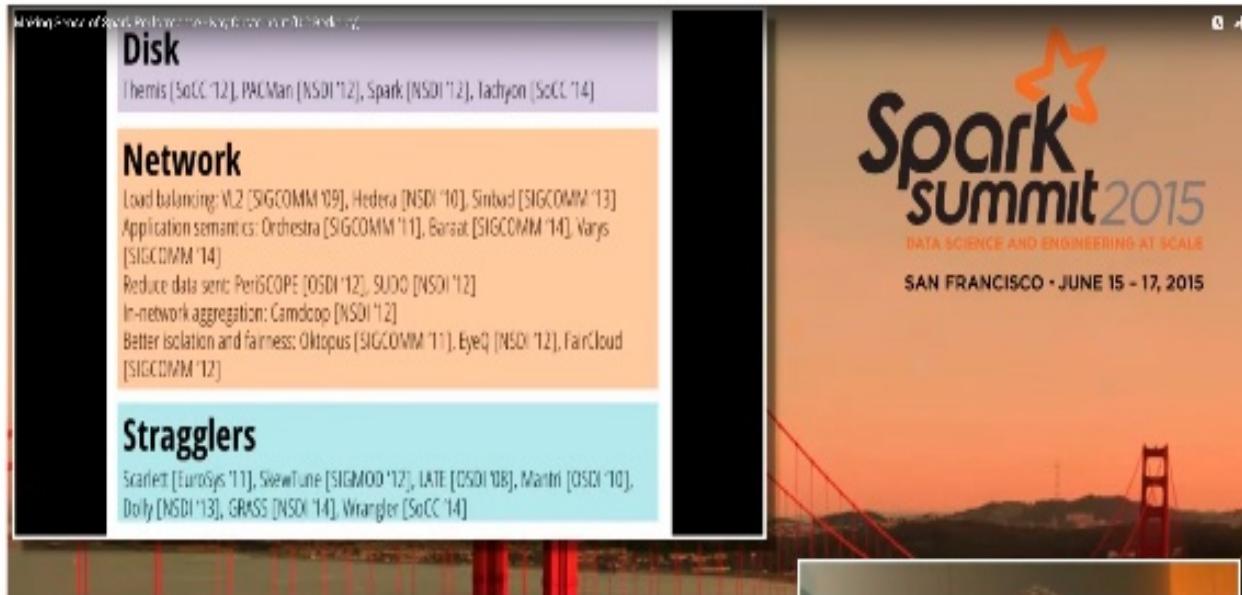
Configuration:

spark.serializer KryoSerializer
spark.executor.cores 8



Realistic view: user uses performance characteristics to tune job configuration, hardware, etc

Reasoning about Spark Performance



Widely accepted that network and disk I/O are bottlenecks

CPU (not I/O) typically the bottleneck

network optimizations can improve job completion time by at most 2%

Reasoning about Spark Performance



Spark Summit 2015:
CPU (not I/O) often
the bottleneck

Project Tungsten:
**initiative to optimize Spark's CPU use,
driven in part by our measurements**

databricks PRODUCT SPARK RESOURCES COMPANY BLOG

COMPANY

All Posts

Partners

Events

Press Releases

DEVELOPER

All Posts

Spark

Spark SQL

Spark Streaming

MLLib



Project Tungsten: Bringing Spark Closer to Bare Metal

April 15, 2015 by Reynold Xie and Jeff Rozen

In previous [blog post](#), we looked back and surveyed performance improvements made to Spark in the past year. In this post, we look forward and dive into the next big step, which we’re calling Project Tungsten. 2014 was hard on Spark setting the world record in large-scale sorting and memory improvements across the entire engine from Python to SQL to machine learning. Performance optimization, however, is a never-ending process.

Project Tungsten will be the largest change to Spark’s execution engine since the project’s inception. It focuses on substantially improving the efficiency of memory and CPU for Spark applications, to push performance closer to the limits of modern hardware. This effort includes three initiatives:

1. **Memory Management and Binary Processing:** leveraging application semantics to manage memory more efficiently and eliminate the overhead of JVM object model and garbage collection
2. **Caches:** cache computation results from multiple nodes to provide faster access to them

Reasoning about Spark Performance



Spark Summit 2015:
CPU (not I/O) often
the bottleneck

Project Tungsten:
initiative to optimize
Spark's CPU use

A screenshot of a LinkedIn post by Ali Ghodsi. The post is titled "Project Tungsten brings Spark Close to Bare Metal". It includes a small profile picture of Ali Ghodsi. The post text discusses the performance improvements of Project Tungsten, stating it can reduce memory access latency by up to 10x, increase memory bandwidth by up to 20x, and reduce disk access latency by up to 100x. It also mentions that Project Tungsten can reduce memory usage by up to 25% and improve disk access times by up to 20x. The post ends with a link to a paper titled "Project Tungsten: Optimizing Spark for Bare Metal".

Spark 2.0:
Some evidence that I/O
is again the bottleneck
[HotCloud '16]

Users need to understand performance to extract the best runtimes

Reasoning about performance is currently difficult

**Software and hardware are constantly evolving,
so performance is always in flux**

Vision: jobs report high-level performance metrics



Details for Stage 17

Total task time across all tasks: 13 min

Shuffle read: 2.5 GB / 31589120

Performance Information

Bottleneck: Disk (if disk bandwidth were increased by 23% or more, network would become the bottleneck)

Vision: jobs report high-level performance metrics



Details for Stage 17

Total task time across all tasks: 13 min

Shuffle read: 2.5 GB / 31589120

Performance Information

Bottleneck: Disk (if disk bandwidth were increased by 23% or more, network would become the bottleneck)

Non-bottlenecks: Network (could reduce network bandwidth by up to 30% slower without impacting runtime), CPU (could increase CPU time by up to 2x without impacting runtime)

Vision: jobs report high-level performance metrics



Details for Stage 17

Total task time across all tasks: 13 min

Shuffle read: 2.5 GB / 31589120

Performance Information

Bottleneck: Disk (if disk bandwidth were increased by 23% or more, network would become the bottleneck)

Non-bottlenecks: Network (could reduce network bandwidth by up to 30% slower without impacting runtime), CPU (could increase CPU time by up to 2x without impacting runtime)

Benefit of caching: Storing input in-memory would reduce job completion time by 42%

Vision: jobs report high-level performance metrics

The screenshot shows the Apache Spark UI interface. At the top, there is a navigation bar with the Spark logo and the word "Monotasks". Below the logo, there are tabs for "Jobs", "Stages", "Storage", "Environment", and "Executors". The "Stages" tab is currently active, indicated by a darker background and a white border.

Details for Stage 17

Total task time across all tasks: 13 min

Shuffle read: 2.5 GB / 31589120

Performance Information

Bottleneck: Disk (if disk bandwidth were increased by 23% or more, network would become the bottleneck)

Non-bottlenecks: Network (could reduce network bandwidth by up to 30% slower without impacting runtime), CPU (could increase CPU time by up to 2x without impacting runtime)

Benefit of caching: Storing input in-memory would reduce job completion time by 42%

Job Runtime Predictor: (enter in properties of different cluster to estimate job's runtime)

Number of machines:

CPU cores per machine:

Network bandwidth per machine: Gbps

I/O bandwidth per machine: MB/s

Calculate new runtime

Predicted new job runtime:

How can we achieve this vision?

Spark overview

Reasoning about Spark's performance: why it's hard

New architecture: monotasks

Reasoning about monotasks performance: why it's easy

Monotasks in action (results)

Example Spark Job:
Read remote data
Filter records
Write result to disk

Task 1:
Read and filter **block 1**
write result to disk

Task 2:
Read and filter **block 2**
write result to disk

Task 3:
Read and filter **block 3**
write result to disk

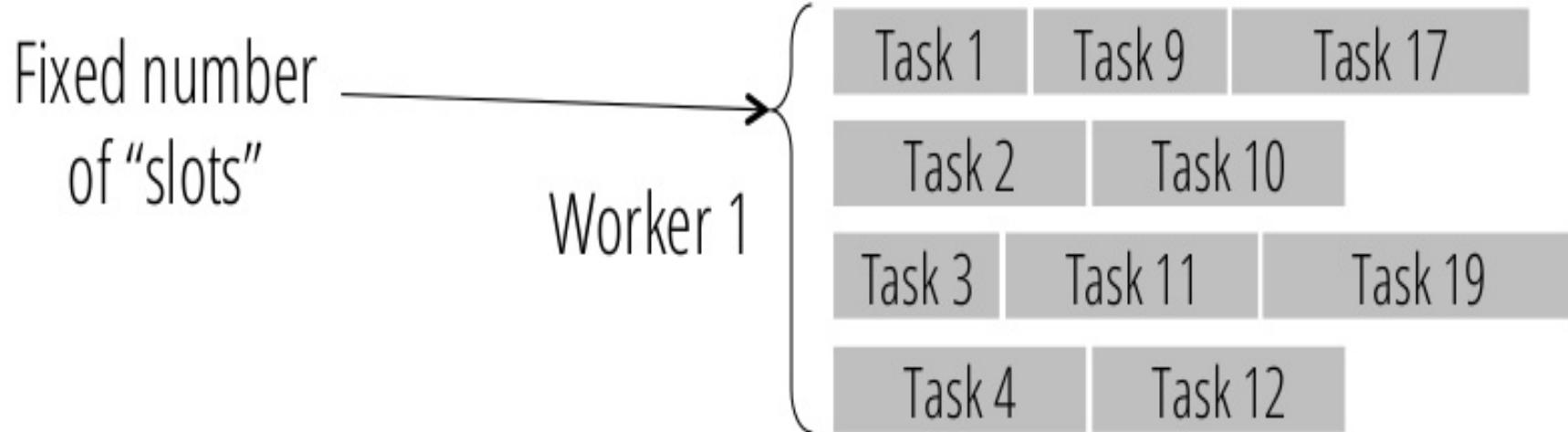
Task 4:
Read and filter **block 4**
write result to disk

Task 5:
Read and filter **block 5**
write result to disk

Task 6:
Read and filter **block 6**
write result to disk

:

Task n:
Read and filter **block n**
write result to disk



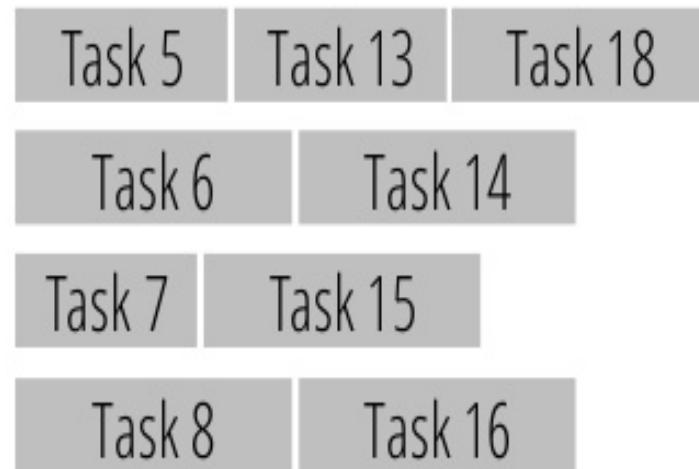
Example Spark Job:

Read remote data

Filter records

Write result to disk

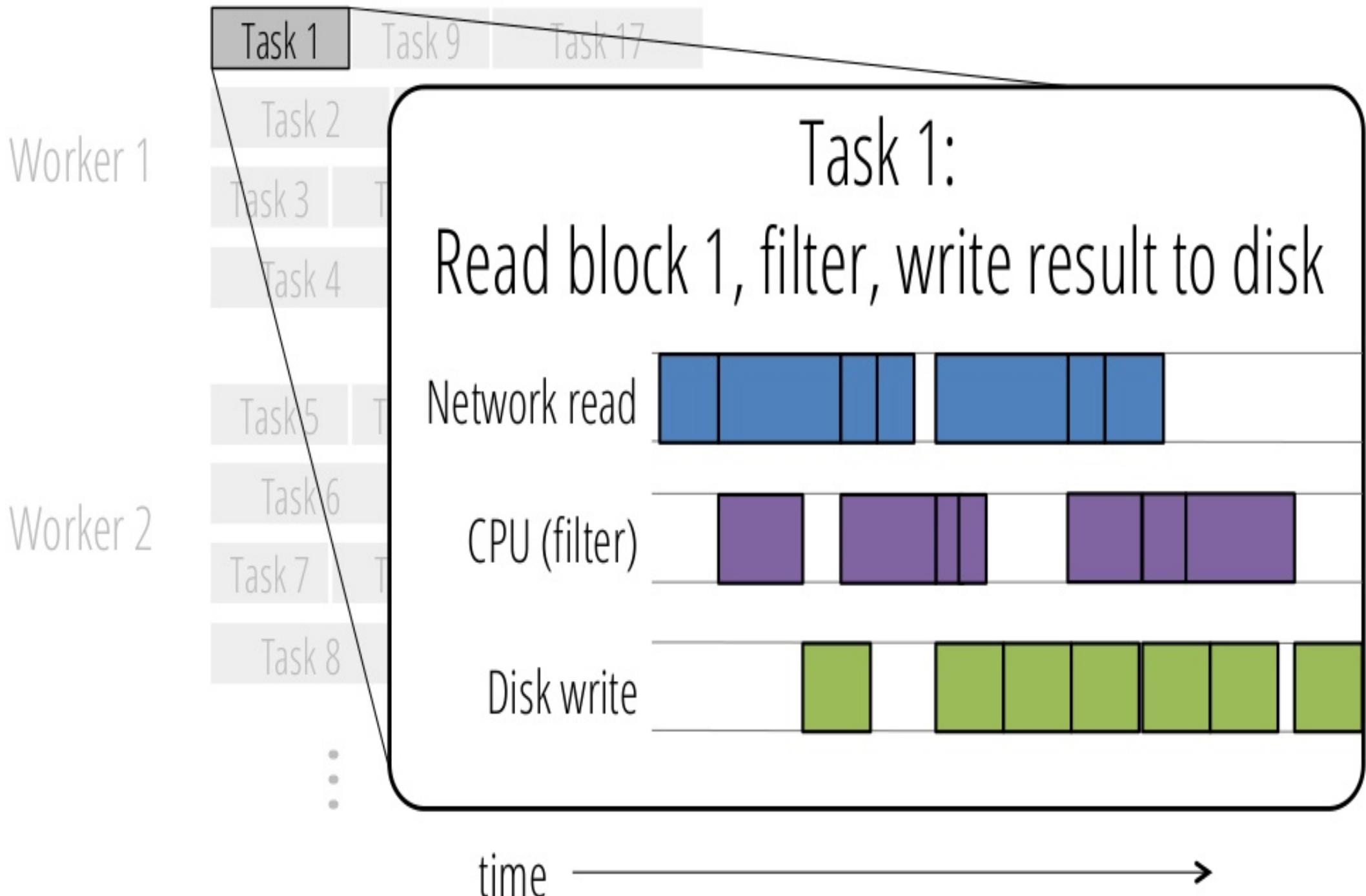
Worker 2



⋮

time





How can we achieve this vision?

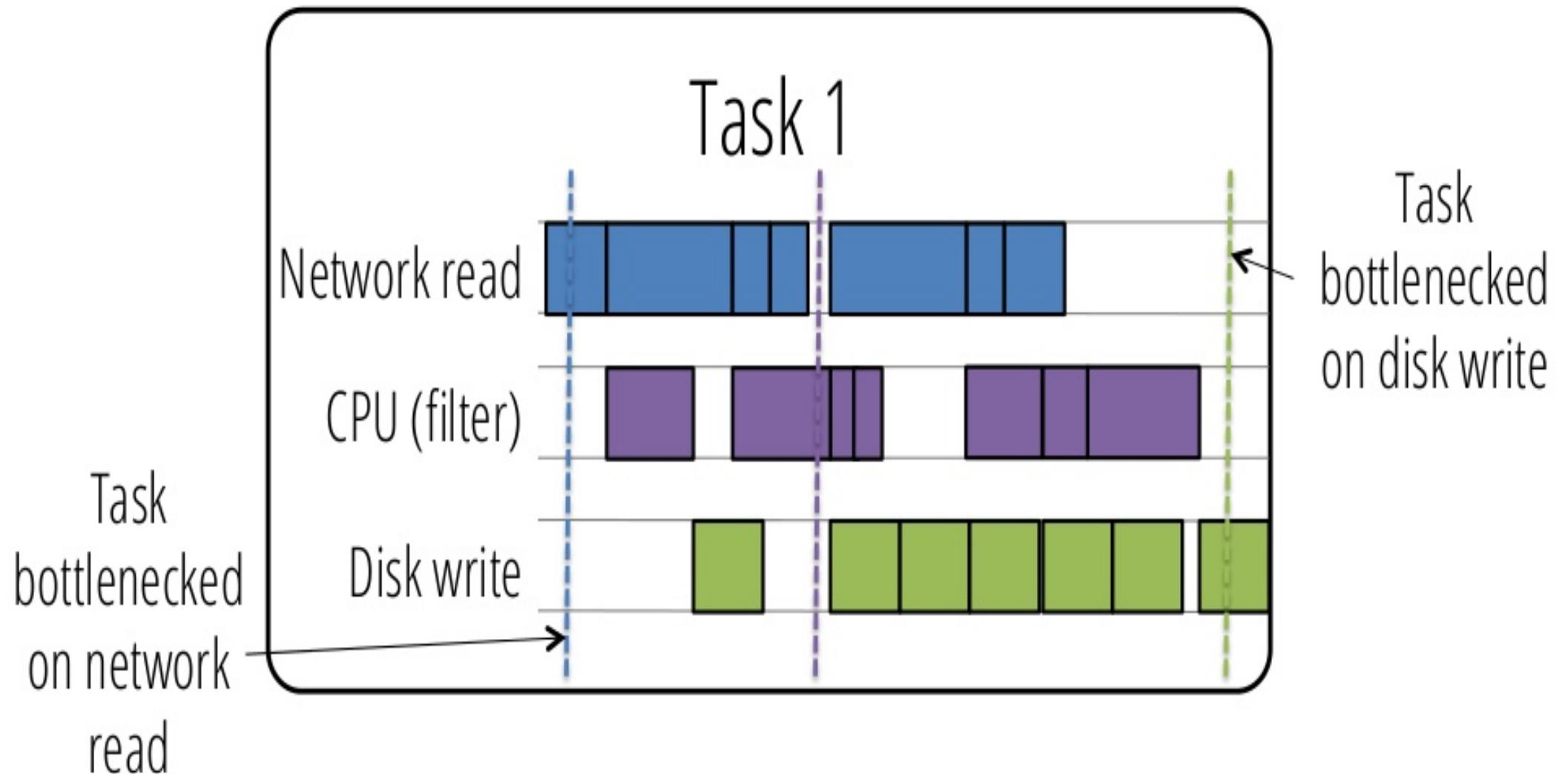
Spark overview

Reasoning about Spark's performance: why it's hard

New architecture: monotasks

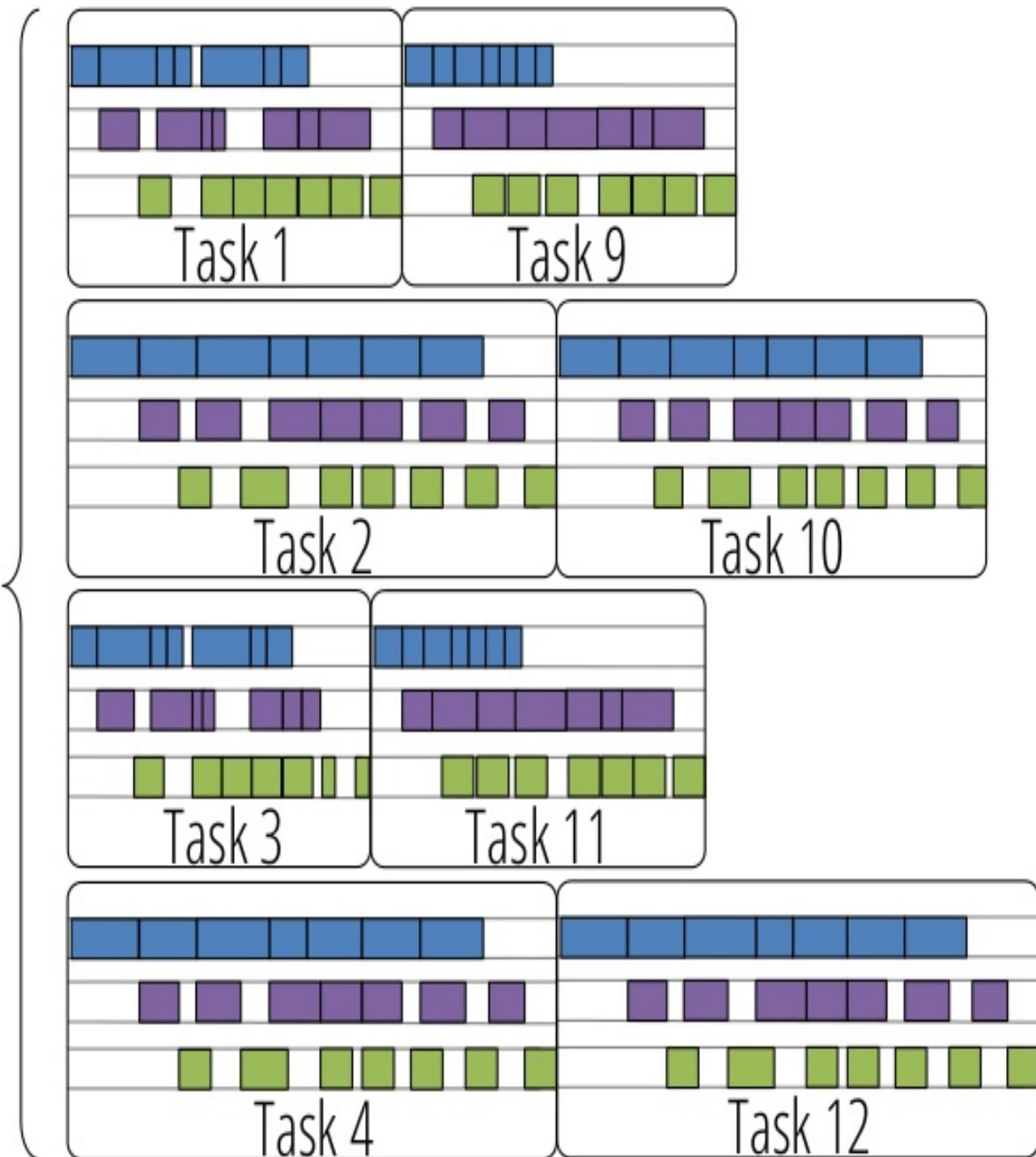
Reasoning about monotasks performance: why it's easy

Monotasks in action (results)

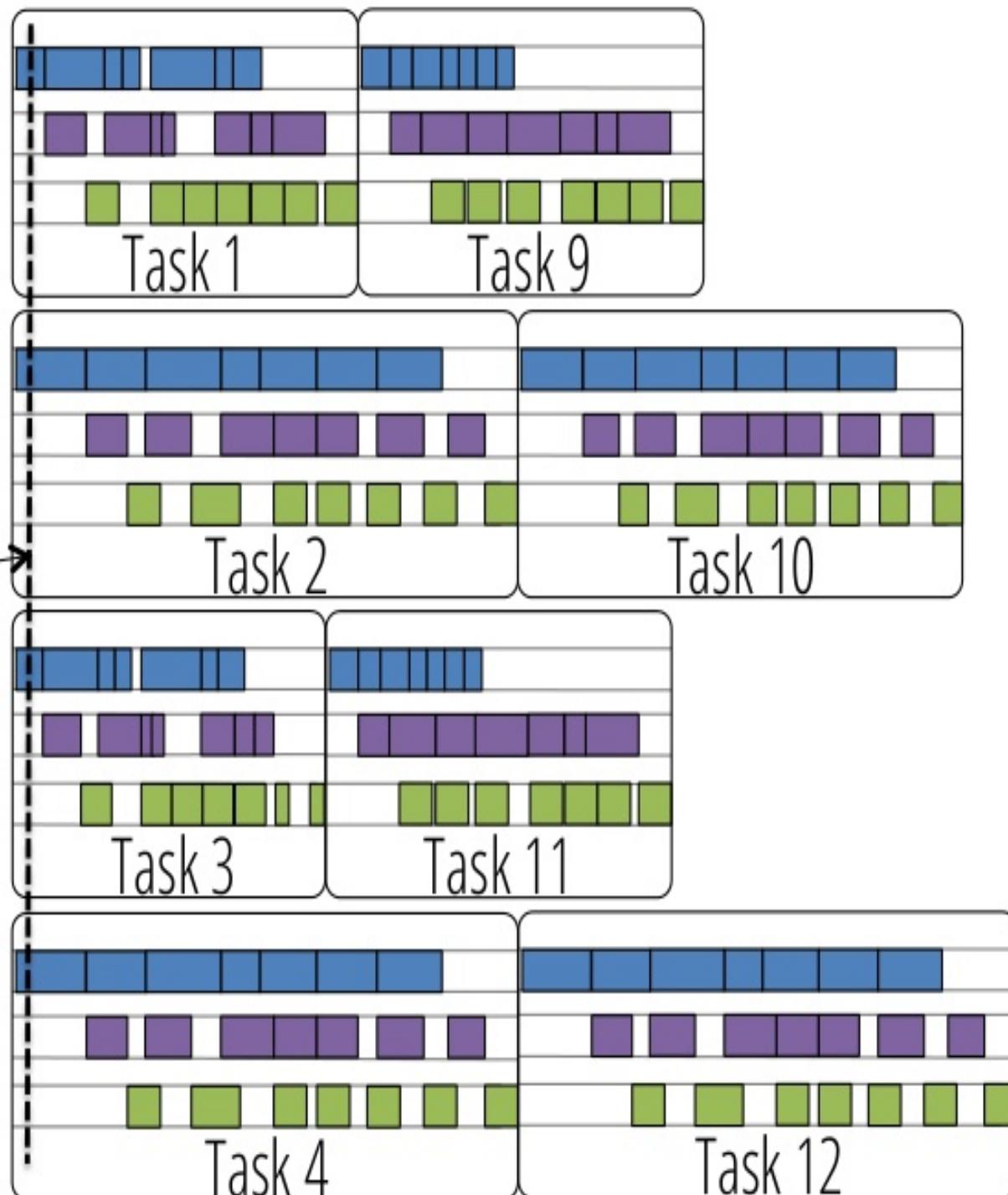


Challenge 1: Task pipelines multiple resources, bottlenecked on different resources at different times

4 concurrent tasks
on Worker 1



Challenge 2:
Concurrent tasks may contend for the same resource (e.g., network)



Spark Summit 2015:

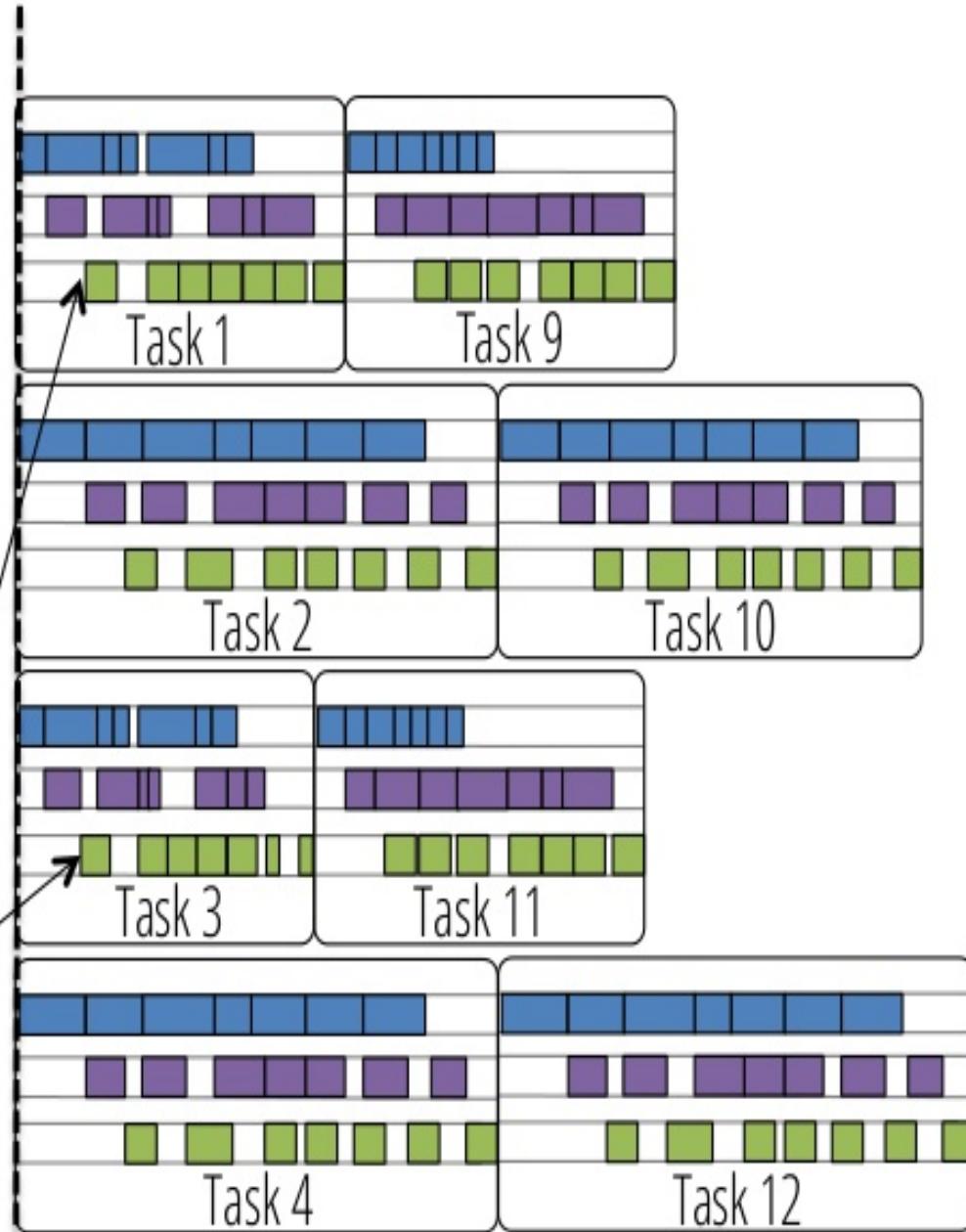
Blocked time analysis: how quickly
could a job have completed if a resource
were infinitely fast? (upper bound)

Result of ~1 year of adding metrics to Spark!

How much faster
would the job be with
2x disk throughput?

How would runtimes for these
disk writes change?

How would that change timing of
(and contention for) other resources?



Challenges to reasoning about performance

Tasks bottleneck on different resources at different times

Concurrent tasks on a machine may contend for resources

No model for performance

How can we achieve this vision?

Spark overview

Reasoning about Spark's performance: why it's hard

New architecture: monotasks

Reasoning about monotasks performance: why it's easy

Monotasks in action (results)

Spark:

Tasks
bottleneck on
different
resources

Concurrent
tasks may
contend

No model for
performance

Spark:

Tasks
bottleneck on
different
resources

Concurrent
tasks may
contend

No model for
performance

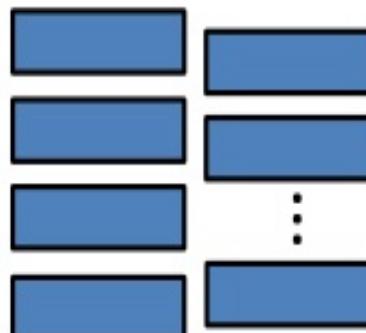
Monotasks: Each task uses one resource

Example Spark Job:

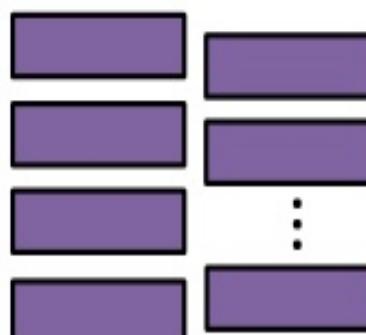
Read remote data

Filter records

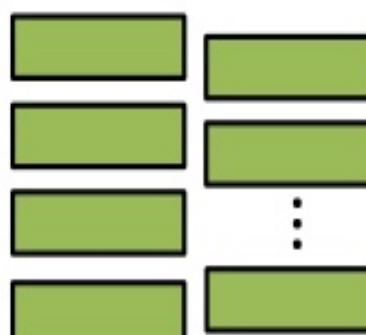
Write result to disk



Network monotasks:
Each read one remote block



CPU monotasks:
Each filter one block,
generate serialized output



Disk monotasks:
Each writes one block to disk

Spark:

Tasks bottleneck on different resources

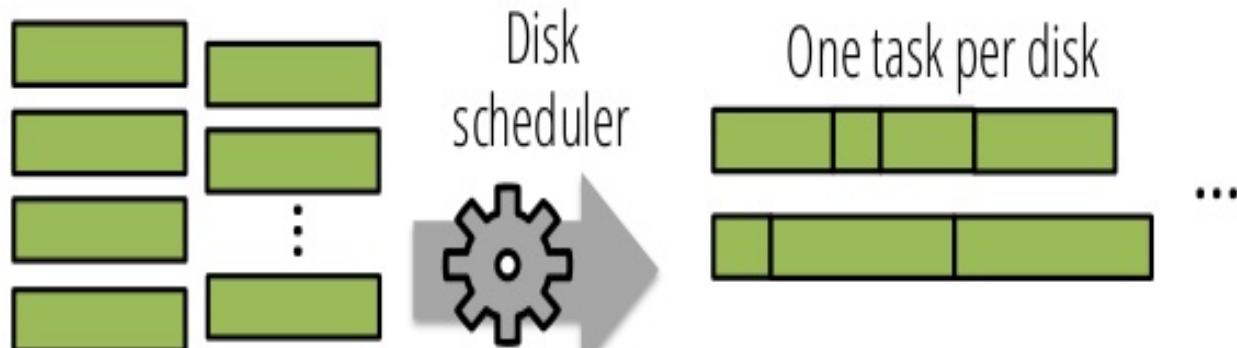
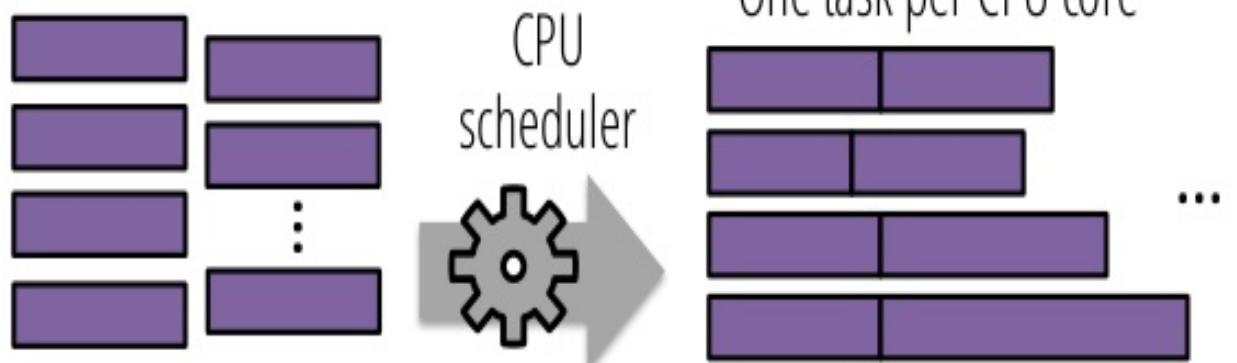
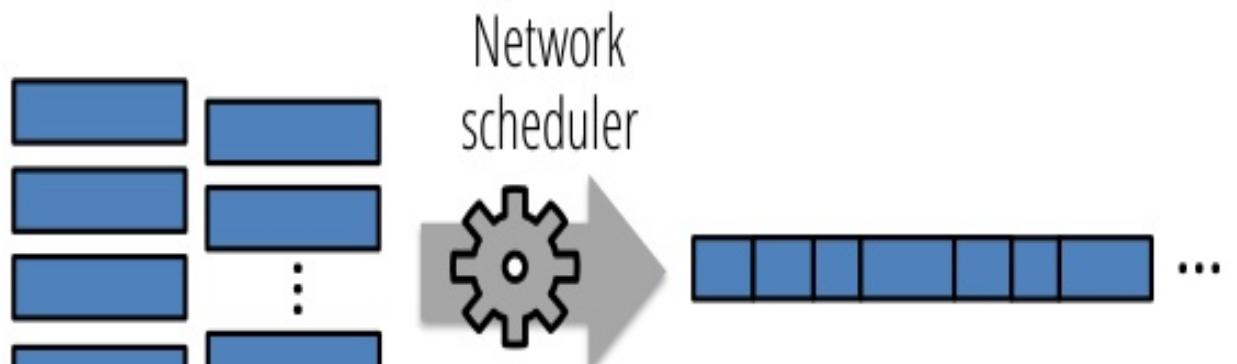
Concurrent tasks may contend

No model for performance

Monotasks:

Each task uses one resource

Dedicated schedulers control contention



Spark:

Tasks bottleneck on different resources

Concurrent tasks may contend

No model for performance

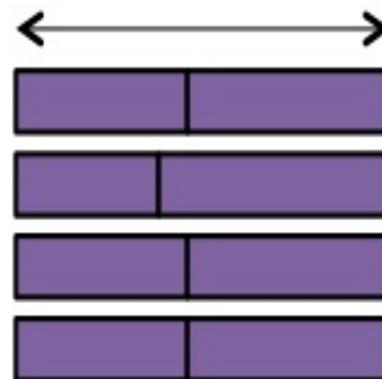
Monotasks:

Each task uses one resource

Dedicated schedulers control contention

Monotask times can be used to model performance

Ideal CPU time: total CPU monotask time / # CPU cores



Spark:

Tasks bottleneck on different resources

Concurrent tasks may contend

No model for performance

Monotasks:

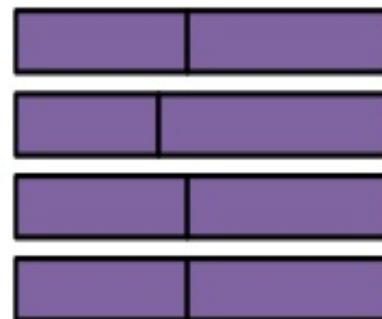
Each task uses one resource

Dedicated schedulers control contention

Monotask times can be used to model performance



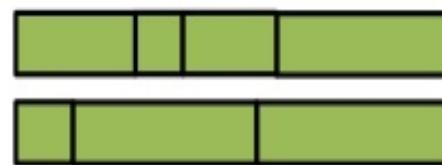
Ideal CPU time:
total CPU monotask time / # CPU cores



Ideal network runtime



Ideal disk runtime



Job runtime:
max of ideal times

Spark:

Tasks bottleneck on different resources

Concurrent tasks may contend

No model for performance

Monotasks:

Each task uses one resource

Dedicated schedulers control contention

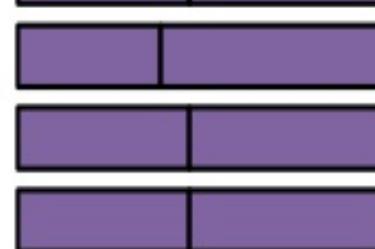
Monotask times can be used to model performance

How much faster would the job be with 2x disk throughput?

Ideal CPU time:



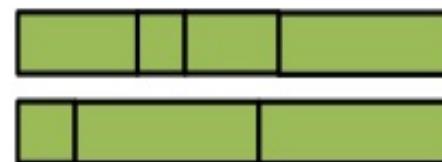
total CPU monotask time / # CPU cores



Ideal network runtime



Ideal disk runtime



Spark:

Tasks bottleneck on different resources

Concurrent tasks may contend

No model for performance

Monotasks:

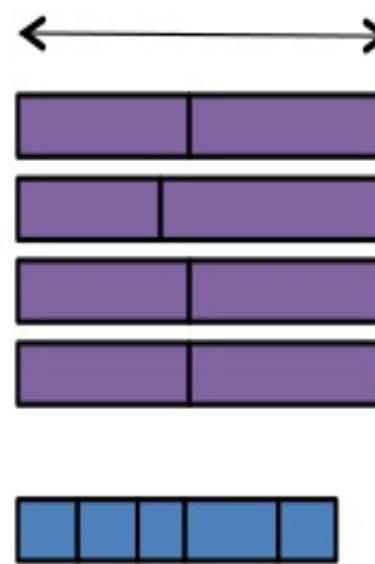
Each task uses one resource

Dedicated schedulers control contention

Monotask times can be used to model performance

How much faster would the job be with 2x disk throughput?

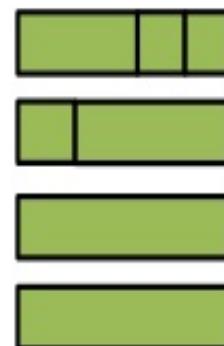
Ideal CPU time:
total CPU monotask time / # CPU cores



Ideal network runtime



Ideal disk runtime
(2x disk concurrency)



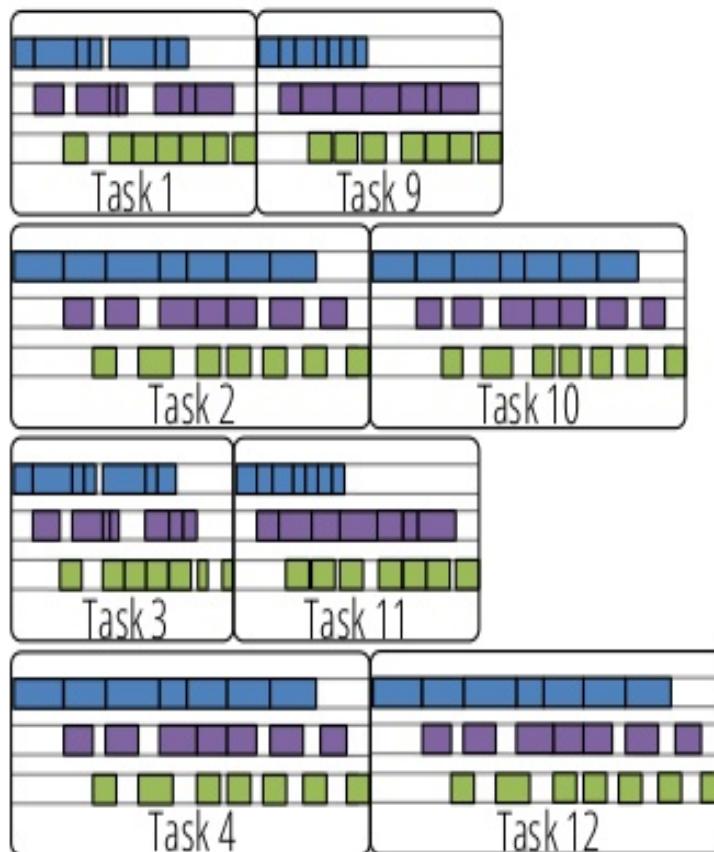
New job runtime

Spark:

Tasks bottleneck on different resources

Concurrent tasks may contend

No model for performance

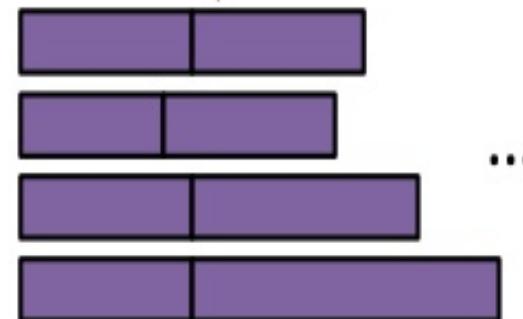


How does this decomposition work?

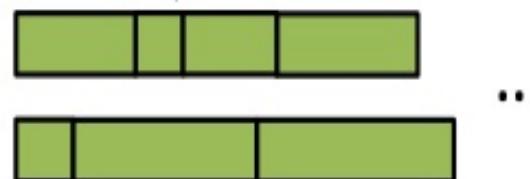
Network monotasks



Compute monotasks:
one task per CPU core



Disk monotasks: one
per disk



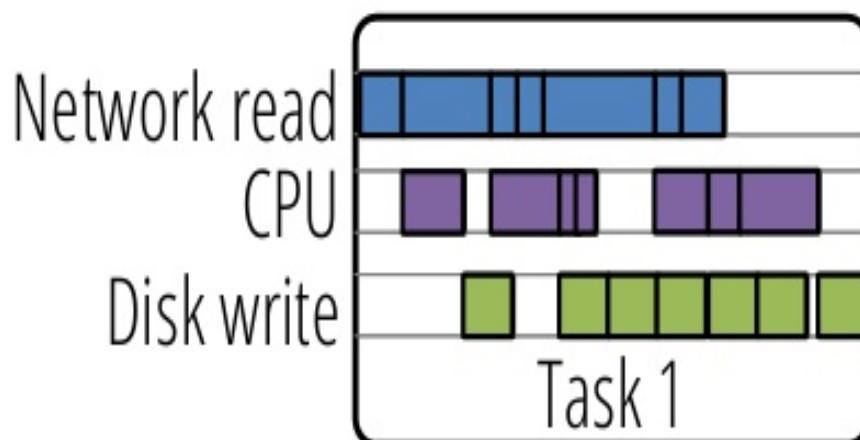
Monotasks:

Each task uses one resource

Dedicated schedulers control contention

Monotask times can be used to model performance

How does this decomposition work?



Monotasks



Implementation

API-compatible with Apache Spark

Workloads can be run on monotasks without re-compiling

Monotasks decomposition handled by Spark internals

Monotasks works at the application level

No operating system changes

How can we achieve this vision?

Spark overview

Reasoning about Spark's performance: why it's hard

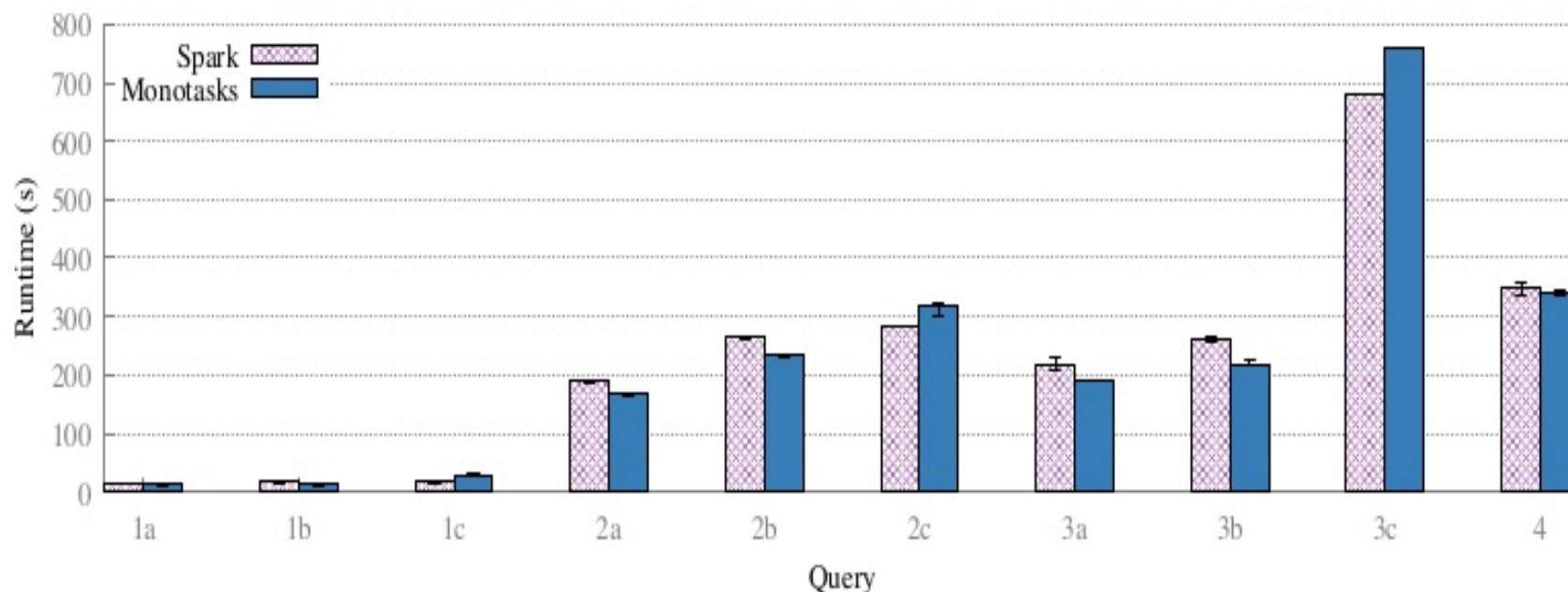
New architecture: monotasks

Reasoning about monotasks performance: why it's easy

Monotasks in action (results)

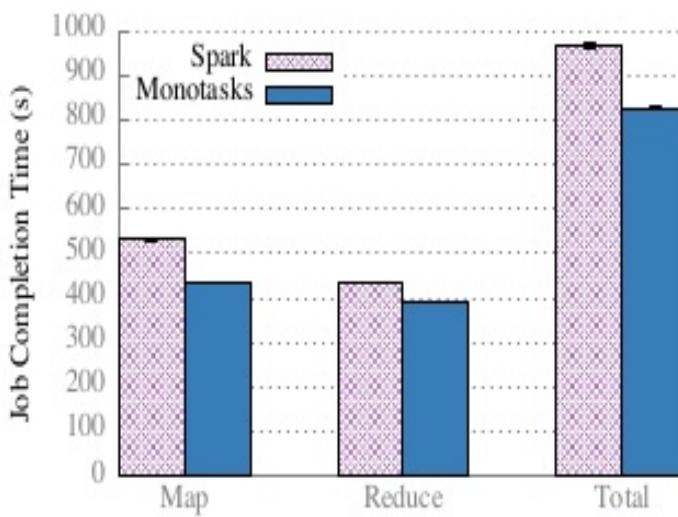
Performance on-par with Apache Spark

Big data benchmark
(SQL workload)

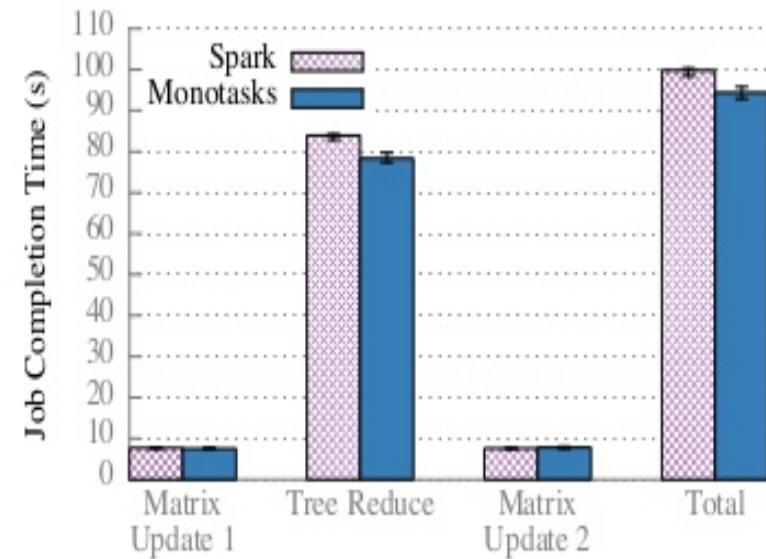


Performance on-par with Apache Spark

Sort
(600 GB, 20 machines)



Block coordinate descent
(Matrix workload used in ML applications)
16 machines

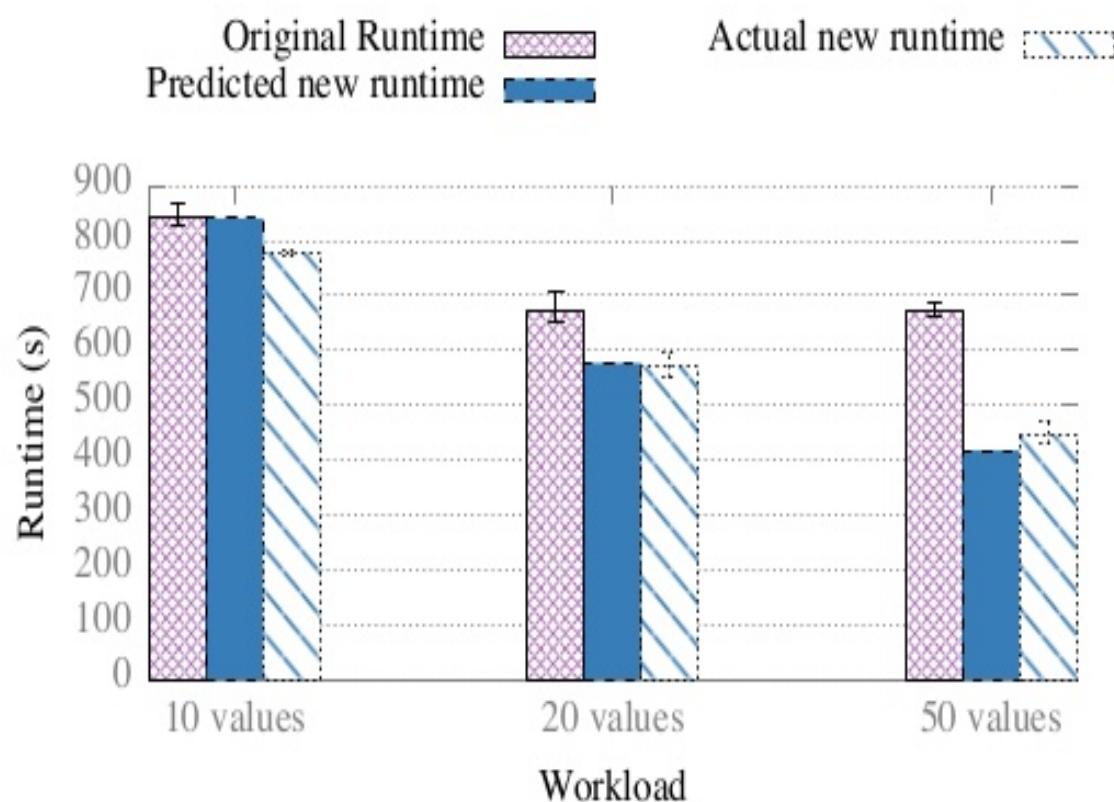


Monotasks in action

Modeling performance

Leveraging performance clarity to optimize performance

How much faster would jobs run if each machine had 2 disks instead of 1?



Predictions for different hardware within 10% of the actual runtime

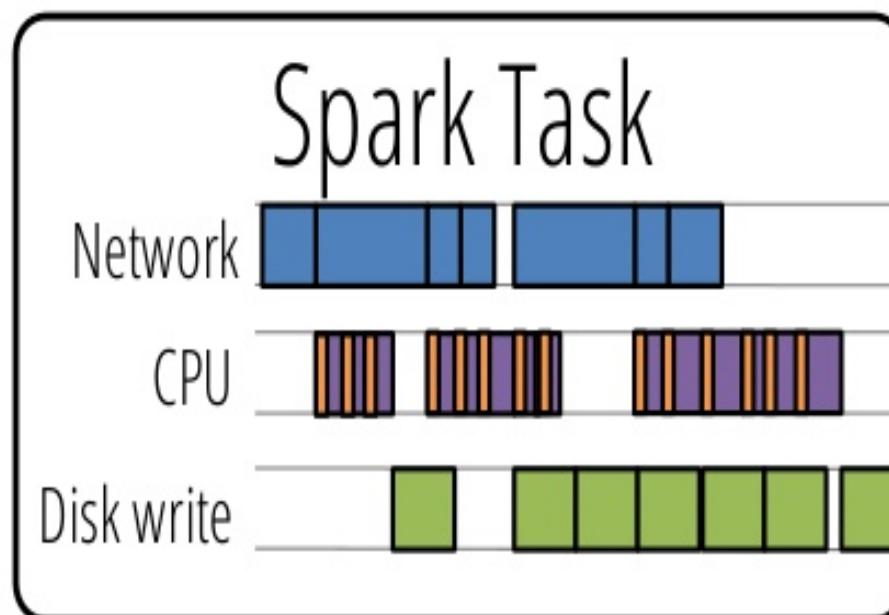
How much faster would job run if data were de-serialized and in memory?

Eliminates disk time to read input data

Eliminates CPU time to de-serialize data

How much faster would job run if data were de-serialized and in memory?

Measuring (de) serialization time with Spark



(De) serialization pipelined with other processing *for each record*

Application-level measurement incurs high overhead

■ : (de)serialization time

How much faster would job run if data were de-serialized and in memory?

Measuring (de) serialization time with Monotasks



Original compute monotask



Un-rolled monotask

Eliminating fine-grained
pipelining enables
measurement!

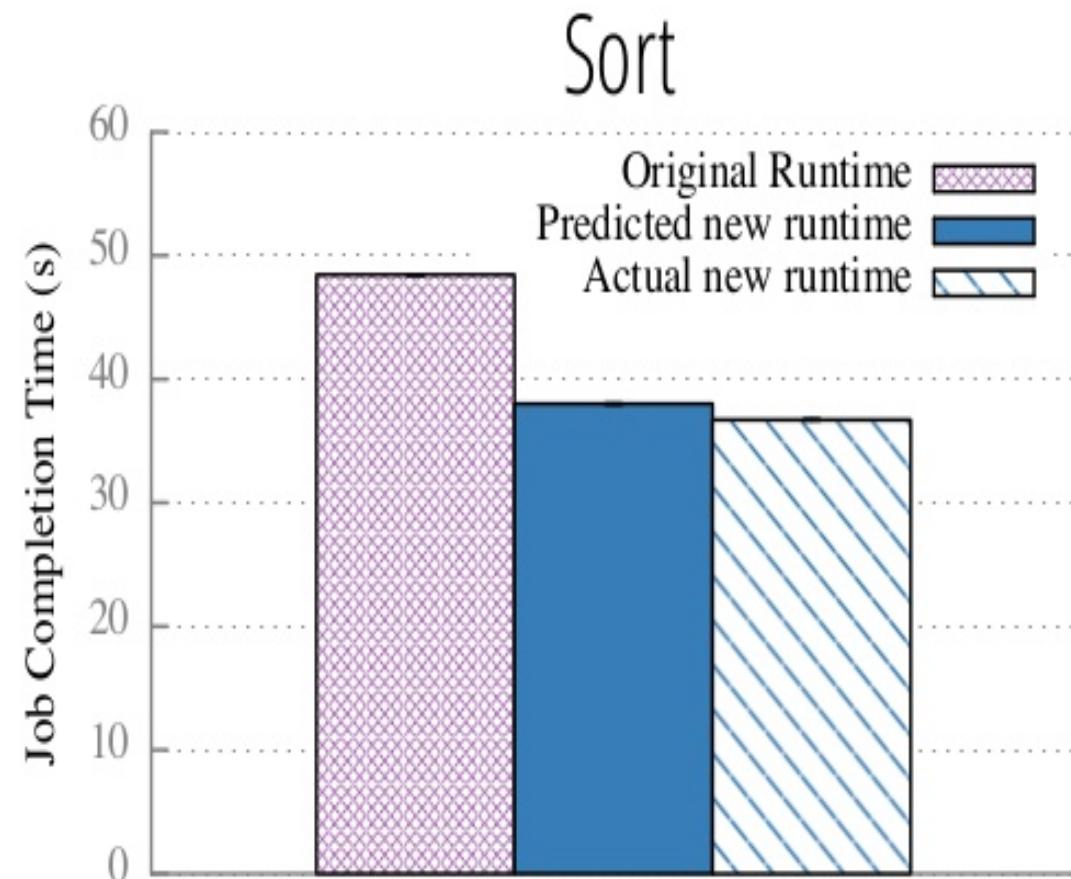
■ : (de)serialization time

How much faster would job run if data were de-serialized and in memory?

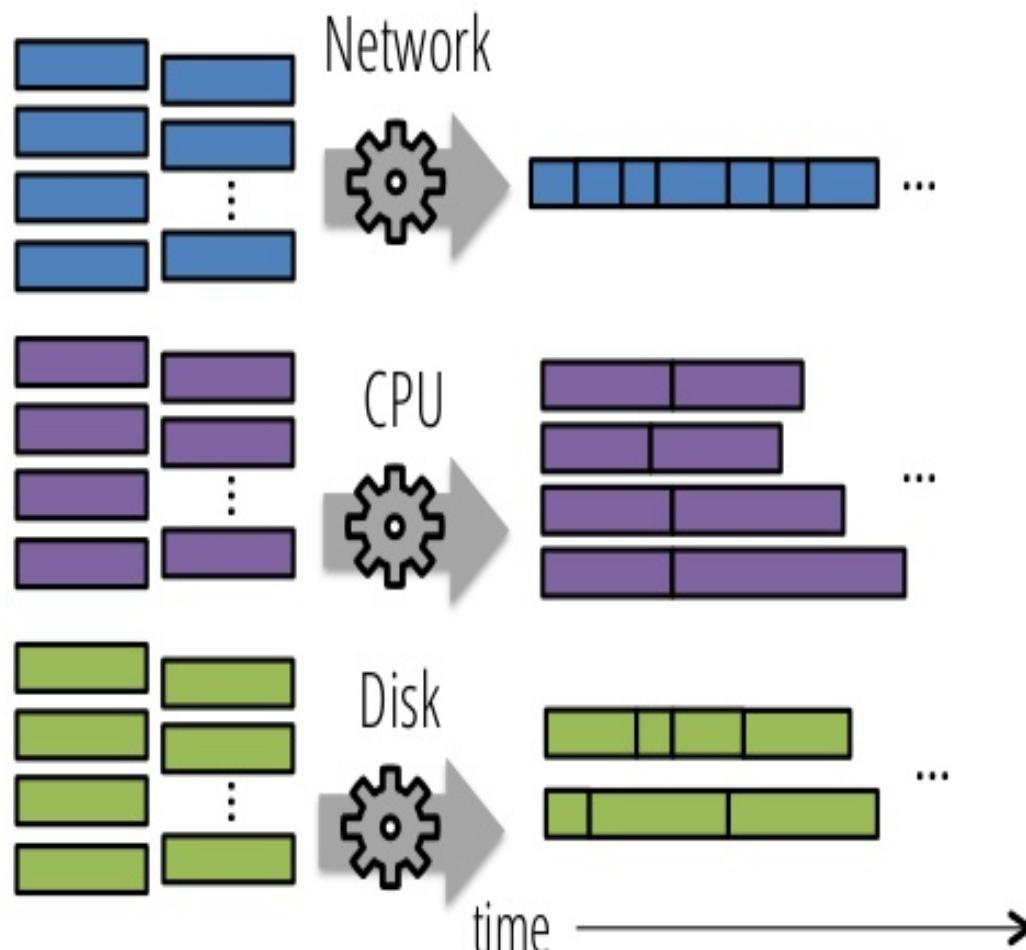
Eliminate disk monotask time

Eliminate CPU monotask time
spent (de)serializing

Re-run model



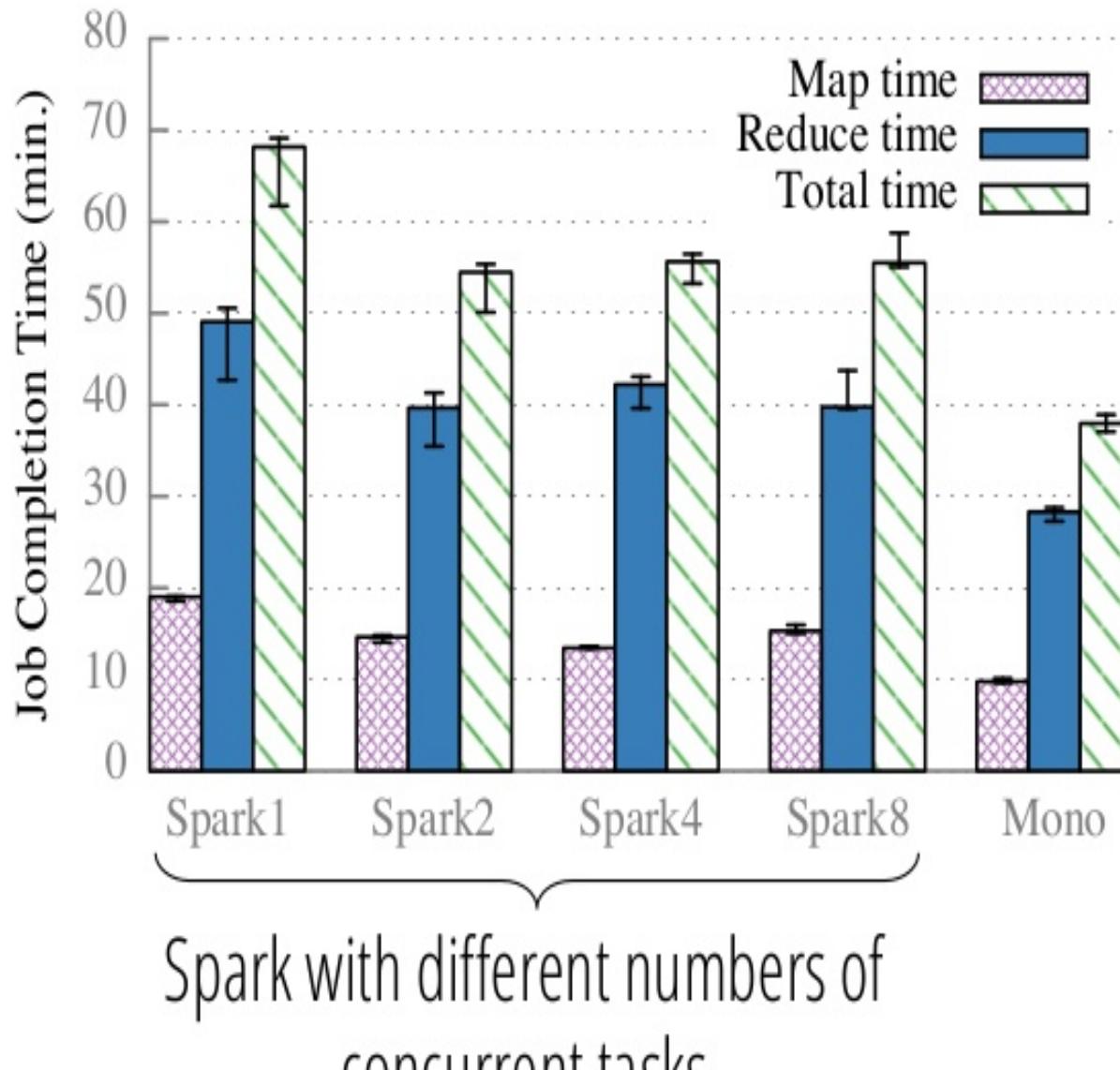
Leveraging Performance Clarity to Improve Performance



Schedulers have complete visibility over resource use

Framework can configure for best performance

Configuring the number of concurrent tasks



Monotasks better than any configuration:
per-resource schedulers automatically schedule with the ideal concurrency

Future Work

Leveraging performance clarity to improve performance

- Use resource queue lengths to dynamically adjust job

Automatically configuring for multi-tenancy

- Don't need jobs to specify resource requirements
- Can achieve higher utilization: no multi-resource bin packing

Details for Stage 17

Total task time across all tasks: 13 min
Shuffle read: 2.5 GB / 31568120

Performance Information

Bottleneck: Disk (if disk bandwidth were increased by 23% or more, network would become the bottleneck)
Non-bottlenecks: Network (could reduce network bandwidth by up to 30% slower without impacting runtime; increase CPU time by up to 2x without impacting runtime)

Benefit of caching: Storing input in-memory would reduce job completion time by 42%

Job Runtime Predictor: (enter in properties of different cluster to estimate job's runtime)

Number of machines:

CPU cores per machine:

Network bandwidth per machine: Gbps

Vision:

Spark always reports bottleneck information

Challenging with existing architecture

Monotasks:

Network monotasks

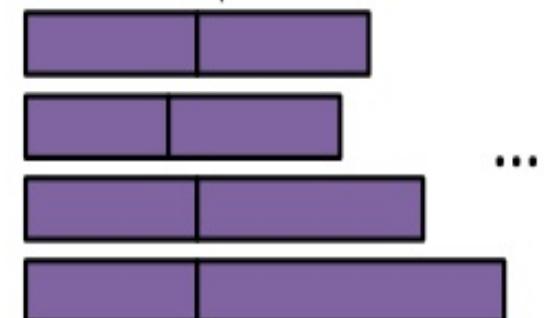


Each task uses one resource

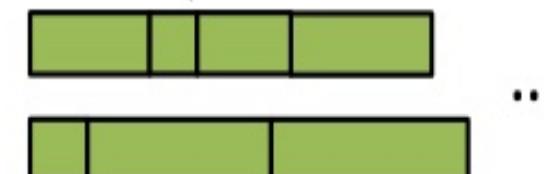
Dedicated schedulers control contention

Monotask times can be used to model performance

Compute monotasks:
one task per CPU core



Disk monotasks: one per disk



Interested? Have a job whose performance you can't figure out? Email me: