Re-Architecting Apache Spark for Performance Understandability

Kay Ousterhout
Joint work with Christopher Canel, Max Wolfe, 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)
Spark cluster is a black box, runs the job fast

\texttt{rdd.groupBy(...)}
Idealistic view: Spark cluster is a black box, runs the job fast

```java
rdd.groupBy(...)
```
Idealistic view: Spark cluster is a black box, runs the job fast

```python
rdd.groupBy(...)...
```
`rdd.groupBy(...)`
`rdd.reduceByKey(...)`

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

Idealistic view: Spark cluster is a black box, runs the job fast
Realistic view: user uses performance characteristics to tune job, configuration, hardware, etc.

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

```
rdd.groupBy(...)
rdd.reduceByKey(...)
```
Users need to be able to reason about performance

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
Reasoning about Spark Performance

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

Project Tungsten: initiative to optimize Spark's CPU use

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
**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)
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

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

Fixed number of “slots”

Worker 1
Task 1
Task 2
Task 3
Task 4
Task 5
Task 6
Task 7
Task 8

Worker 2
Task 9
Task 10
Task 11
Task 12
Task 13
Task 14
Task 15
Task 16
Task 17
Task 18
Task 19

time
Task 1:
Read block 1, filter, write result to disk

Network read

CPU (filter)

Disk write

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.

Task 1

- Network read
- CPU (filter)
- Disk write

Task bottlenecked on network read

Task bottlenecked on disk write
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

Network scheduler: One task per network
CPU scheduler: One task per CPU core
Disk scheduler: One task per disk

time
Spark:

Monotasks:

- Each task uses one resource
- Dedicated schedulers control contention

Concurrent tasks may contend

No model for performance

Monotask times can be used to model performance

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

Monotasks:
- Each task uses one resource
- Dedicated schedulers control contention
- No model for performance

Tasks bottleneck on different resources
Concurrent tasks may contend

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

Monotasks:
- Each task uses one resource
- Dedicated schedulers control contention
- Monotask times can be used to model performance

How does this decomposition work?

Network monotasks
Compute monotasks: one task per CPU core
Disk monotasks: one per disk
How does this decomposition work?

Monotasks

Network read
CPU
Disk write

Task 1

Network monotask → Compute monotask → Disk monotask
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

Spark Task

<table>
<thead>
<tr>
<th>Network</th>
<th>CPU</th>
<th>Disk write</th>
</tr>
</thead>
</table>

: (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 deserialized 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

Spark with different numbers of concurrent tasks
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
Monotasks:
Each task uses one resource
Dedicated schedulers control contention
Monotask times can be used to model performance

Network monotasks
Compute monotasks: one task per CPU core
Disk monotasks: one per disk

Vision:
Spark always reports bottleneck information
Challenging with existing architecture

Interested? Have a job whose performance you can’t figure out? Email me:
keo@cs.berkeley.edu