On The [Ir]relevance of Network Performance for Data Processing

Animesh Trivedi, Patrick Stuedi, Jonas Pfefferle, Radu Stoica, Bernard Metzler, Ioannis Koltsidas, Nikolas Ioannou

IBM Research, Zurich
How [Ir]relevant is the Network?

Making Sense of Performance in Data Analytics Frameworks

Kay Ousterhout*, Ryan Rasti*†⋄, Sylvia Ratnasamy*, Scott Shenker*†, Byung-Gon Chun‡
*UC Berkeley, †ICSI, ⋄VMware, ‡Seoul National University

Network optimizations can only reduce job completion time by a median of at most 2%. The network is not a bottleneck because much less data is sent over the network than is transferred to and from disk. As a result, network I/O is mostly irrelevant to overall performance, even on 1Gbps networks.
How [Ir]relevant is the Network?

Making Sense of Performance in Data Analytics Frameworks
Kay Ousterhout*, Ryan Rasti*†♦, Sylvia Ratnasamy*, Scott Shenker*†, Byung-Gon Chun‡
*UC Berkeley, †ICSI, ♦VMware, ‡Seoul National University

Network optimizations can only reduce job completion time by a median of at most 2%. The network is not a bottleneck because much less data is sent over the network than is transferred to and from disk. As a result, network I/O is mostly irrelevant to overall performance, even on 1Gbps networks.
How [Ir]relevant is the Network?

<table>
<thead>
<tr>
<th>Runtimes in secs</th>
<th>TeraSort</th>
<th>PageRank</th>
<th>SQL</th>
<th>WordCount</th>
<th>GroupBy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How [Ir]relevant is the Network?

<table>
<thead>
<tr>
<th></th>
<th>TeraSort</th>
<th>PageRank</th>
<th>SQL</th>
<th>WordCount</th>
<th>GroupBy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime in secs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Gbps</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The diagram shows the runtime in seconds for different workloads at different network speeds. The workloads include TeraSort, PageRank, SQL, WordCount, and GroupBy.
How [Ir]relevant is the Network?

Network IO is very relevant - up to 64%
How [Ir]relevant is the Network?

Network IO is very relevant - up to 64% ??
Is It Spark Specific?

The bar chart shows the runtime in seconds for different systems running at different data transfer rates:

- **Flink-TS**: 725 seconds at 1 Gbps
- **Flink-PR**
- **GraphLab**
- **Timely**

The data transfer rates are 1 Gbps, 10 Gbps, and 40 Gbps.
Spark TeraSort: The Shuffle Story

distributed sorting
- simple
- shuffle data is input data
- highest chance of improvements

input

output
Spark TeraSort: The Shuffle Story

- Map tasks
- Cores
- Shuffle data
- Reduce tasks

input → Map tasks → Cores → Shuffle data → Reduce tasks → output
Spark TeraSort: The Shuffle Story

Map tasks

Cores

Shuffle data

Reduce tasks

input

output

reading in shuffle data

The 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16)
Spark TeraSort: The Shuffle Story

- **Map tasks**
- **Shuffle data**
- **Reduce tasks**

**Reading in shuffle data**

**Sorting shuffle data**

**Cores**

**Net**

**CPU**

**Input**

**Output**
Spark TeraSort: The Shuffle Story

Map tasks → Cores → Shuffle data → Reduce tasks

input → reading in shuffle data + sorting shuffle data = performance

output

Cores

Map tasks

shuffle data

Reduce tasks

net CPU

net CPU

net CPU
How Important is the Network?

Gains from the networks are shadowed by the high CPU footprint.
How Important is the Network?

Gains from the networks are shadowed by the high CPU footprint
How Important is the Network?

Gains from the networks are overshadowed by the high CPU footprint.
How Important is the Network?

The gains from the network are shadowed by the high CPU footprint.
How Important is the Network?

Network gains are shadowed by the CPU.
What Exactly is the CPU Doing?

Spark

- Misc.
- Iterator
- Serialization
- Sorting
- IO
- JVM
- Linux

The 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16)
What Exactly is the CPU Doing?

![Bar chart showing CPU activity during Map and Reduce stages in Spark.](chart.png)

- **Map** stages:
  - Misc.: 10%
  - Iterator: 20%
  - Serialization: 40%
  - IO: 20%
  - JVM: 10%
  - Linux: 10%

- **Reduce** stages:
  - Misc.: 5%
  - Iterator: 25%
  - Serialization: 45%
  - IO: 10%
  - JVM: 15%
  - Linux: 5%
What Exactly is the CPU Doing?

Overheads are spread across the entire stack - serialization, abstraction, execution model etc.
The Balancing Act: CPU vs Network
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

Sorting: $O(n\log(n))$
Network: $O(n)$

use smaller 'n'
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

![Chart showing the relationship between runtime (in seconds) and smaller partitions](image-url)
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

![Graph showing runtime (secs) vs smaller partitions](image)
The Balancing Act: CPU vs Network

1. Balance out the CPU with the network time

![Graph showing the relationship between Smaller Partitions and Runtime (secs)]
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

II. Use more cores to scale up

if a single core cannot do 40 Gbps
then use more
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

II. Use more cores to scale up
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

II. Use more cores to scale up
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

II. Use more cores to scale up
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

II. Use more cores to scale up

---

The 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16)
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

II. Use more cores to scale up

runtime = 9 + \frac{260}{\text{cores}}
The Balancing Act: CPU vs Network

I. Balance out the CPU with the network time

II. Use more cores to scale up

 Classical techniques are ineffective

The 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16)
Conclusion

1. Faster networks (IO) are very relevant
   - as long as you have CPU cycles
   - differentiate between user vs framework CPU usage
Conclusion

1. Faster networks (IO) are very relevant
   - as long as you have CPU cycles
   - differentiate between user vs framework CPU usage

2. Framework's CPU usage is bad
   - CPU-network imbalance: sorting, serialization, volcano execution model, etc.
   - scalability (serial vs parallel components)
   - ineffective classical balancing techniques
Conclusion

1. Faster networks (IO) are very relevant
   - as long as you have CPU cycles
   - differentiate between *user* vs *framework* CPU usage

2. Framework's CPU usage is bad
   - CPU-network imbalance: sorting, serialization, volcano execution model, etc.
   - scalability (serial vs parallel components)
   - ineffective classical balancing techniques

3. Knowing today's *usec-era* IO and CPU hardware, how would you re-design modern data processing framework?
Backup
Spark

Apache Spark as a Compiler: Joining a Billion Rows per Second on a Laptop
Deep dive into the new Tungsten execution engine

by Sameer Agarwal, Davide Liu and Reynold Xin
Posted in ENGINEERING BLOG, May 23, 2016

Spark Summit 2016 will be held in San Francisco on June 6–8. Check out the full agenda and get your ticket before it sells out!

Try the whole-stage code generation notebook in Databricks Community Edition

When our team at Databricks planned our contributions to the upcoming Apache Spark 2.0 release, we set out with an ambitious goal by asking ourselves: **Apache Spark is already pretty fast, but can we make it 10x faster?**

This question led us to fundamentally rethink the way we built Spark’s physical execution layer. When you look into a modern data engine (e.g., Spark or other MPP databases), a majority of the CPU cycles are spent in useless work, such as making virtual function calls or reading or writing intermediate data to CPU cache or memory. Optimizing performance by reducing the amount of CPU cycles wasted in this useless work has been a long-time focus of modern compilers.

Apache Spark 2.0 will ship with the **second generation Tungsten engine**. Built upon ideas from modern compilers and MPP databases and applied to data processing queries, Tungsten emits (SPARK-12795) optimized bytecode at runtime that collapses the entire query into a single function, eliminating virtual function calls and leveraging CPU registers for intermediate data. As a result of this streamlined strategy, called “whole-stage code generation,” we significantly improve CPU efficiency and gain performance.

The Past: Volcano Iterator Model

Before we dive into the details of whole-stage code generation, let us revisit how Spark (and most database systems) work currently. Let us illustrate this with a simple query that scans a single table and counts the number of elements with a given attribute value:

```
select count(*) from store_sales
```

Aggregate

Project

Stay up to date on Apache Spark.
Conclusion

Most of the work described in this blog post has been committed into Apache Spark’s code base and is slated for the upcoming Spark 2.0 release. The JIRA ticket for whole-stage code generation can be found in SPARK-12795, while the ticket for vectorization can be found in SPARK-12992.

To recap, this blog post described the second generation Tungsten execution engine. Through a technique called whole-stage code generation, the engine will (1) eliminate virtual function dispatches (2) move intermediate data from memory to CPU registers and (3) exploit modern CPU features through loop unrolling and SIMD. Through a technique called vectorization, the engine will also speed up operations that are too complex for code generation. For many core operators in data processing, the new engine is orders of magnitude faster. In the future, given the efficiency of the execution engine, bulk of our performance work will shift towards optimizing I/O efficiency and better query planning.

We are excited about the progress made, and hope you will enjoy the improvements. To try some of these out for free, sign up for an account on Databricks Community Edition.
What Exactly is the CPU Doing?

Spark Map Reduce

- Misc.
- Iterator
- Serialization
- Sorting
- IO
- JVM
- Linux