Cloud Query Planning: Laziness as a Service

Keith Winstein

Stanford Computer Science Department
https://cs.stanford.edu/~keithw
Keith Winstein

- Assistant Professor, Stanford
  - Computer Science
  - Law (by courtesy)
  - Robert N. Noyce Family Faculty Scholar
- Ph.D. in CS from MIT, 2014
Part 1: workload $\rightarrow$ compute engine $\rightarrow$ answer

Part 2: query the answer
Dataflow programming today

Part 1: workload $\rightarrow$ compute engine (Spark, Hadoop) $\rightarrow$ answer

Part 2: query the answer
Dataflow programming today

Part 1: workload $\rightarrow$ compute engine ($\text{Spark, Hadoop}$) $\rightarrow$ answer

Part 2: query the answer
Better: climb up the stack

Ask app: **what are you really trying to do?**

(Cf. Mosh, Remy, xFabric)
Example: 100 TB sort

Sort Benchmark Home Page

New: The next deadline for submitting entries is September 1, 2015.

We are deprecating and will no longer accept results for PennySort and the $10^8$, $10^9$ and $10^{12}$ record JouleSorts. PennySort and $10^{12}$ record JouleSort have outlived their usefulness and have been superceded by CloudSort. $10^8$ and $10^9$ record JouleSort are too similar to the $10^{10}$ record JouleSort which continues on.

Other than the aforementioned deprecations, there are no rule changes for 2015.

The 2014 records are listed below in green. Thank you to all the 2014 entrants!

Background

Until 2007, the sort benchmarks were primary defined, sponsored and administered by Jim Gray. Following Jim's disappearance at sea in January 2007, the sort benchmarks have been continued by a committee of past colleagues and sort benchmark winners. The Sort Benchmark committee members include:

- Chris Nyberg of Ordinal Technology Corp
- Mehul Shah of Amiato
- Naga Govindaraju of Microsoft

Top Results

<table>
<thead>
<tr>
<th>Daytona</th>
<th>Indy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-way tie: 2014, 4.35 TB/min</td>
<td>2014, 8.38 TB/min</td>
</tr>
<tr>
<td><strong>TritonSort</strong></td>
<td><strong>BaiduSort</strong></td>
</tr>
<tr>
<td>100 TB in 1,378 seconds</td>
<td>100 TB in 716 seconds</td>
</tr>
<tr>
<td>186 Amazon EC2 i2.8xlarge nodes x (32 vCores - 2.50GHz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD)</td>
<td>982 nodes x (2 2.10GHz Intel Xeon E5-2450, 192 GB memory, 8x3TB 7200 RPM SATA)</td>
</tr>
<tr>
<td>Michael Conley, Amin Vahdat.</td>
<td>Dasheng Jiang Baidu Inc. and Peking University</td>
</tr>
</tbody>
</table>
2014 world-record sort implementation (TritonSort, Spark)

1. Start: 1 trillion records across 200 EC2 machines
2. Each source machine partitions its records for 200 targets
3. [700 s] Shuffle all payloads: 200 sources to 200 targets
4. [700 s] Each target machine sorts its records
5. End: Store in HDFS across 200 machines

Eventual read speed is unmeasured/unspecified.
Treat the problem holistically

Taking workload and resources as given misses an opportunity.

- Wiggle room 1: choice of algorithm/implementation
- Wiggle room 2: choice of substrate (EC2 machines).
- Wiggle room 3: worry about eventual query workload!
Several equivalent strategies for the problem

1. Shuffle all payloads. Each target gets contiguous substring of overall. [1400 s Spark/Triton, 400 s possible]
Several equivalent strategies for the problem

1. Shuffle all payloads. Each target gets contiguous substring of overall. [1400 s Spark/Triton, 400 s possible]

2. Shuffle all keys. Each target gets contiguous subset of global index. [90 s]
Several equivalent strategies for the problem

1. Shuffle all payloads. Each target gets contiguous substring of overall. [1400 s Spark/Triton, 400 s possible]

2. Shuffle all keys. Each target gets contiguous subset of global index. [90 s]

3. Build local index. Shuffle nothing. [30 s]
   (Single-threaded reads still faster than HDFS!)
Several equivalent strategies for the problem

1. Shuffle all payloads. Each target gets contiguous substring of overall. [1400 s Spark/Triton, 400 s possible]

2. Shuffle all keys. Each target gets contiguous subset of global index. [90 s]

3. Build local index. Shuffle nothing. [30 s]
   (Single-threaded reads still faster than HDFS!)

4. Do nothing upfront at all! Reconstruct on read. [0 s]
For heterogenous architecture and given strategy:

**Range of tradeoffs** among:

- Time to first record
- Single-node linear read throughput
- Multi-node linear read throughput
- Random read throughput
- cost ($)
Proposal: understand and expose the tradeoffs
Hard part: need to understand the machine

- Does system have the same performance today as yesterday?
- Have we modeled all the types of interference?
Model and reality

- **Model** informs **reality**: recommends strategies based on performance/cost predictions.

- **Reality** informs **model**: learns constants that drive model predictions.
Goals

- Dataflow system with range of solutions for:
  - sort
  - parallel join
  - recommendation engine (ALS)
  - key/value store

- Recommender that guides choice of strategy
- Intelligent multiplexing of diverse resources
Goals

- Dataflow system with range of solutions for:
  - sort
  - parallel join
  - recommendation engine (ALS)
  - key/value store
- Recommender that guides choice of strategy
- Intelligent multiplexing of diverse resources
- (Intention: win the sort benchmark \( n \) ways.)
Conclusion

- Platform for solving problems (not workloads)
- Important to consider *end-to-end* needs
- Provide app with usable intelligence about “how to do it”
- **Hard part**: dynamic, evolving understanding of the machine

Keith Winstein (keithw@cs.stanford.edu)