# LOOM: Optimal Aggregation Overlays for In-Memory Big Data Processing

William Culhane, Kirill Kogan, Chamikara Jayalath, Patrick Eugster
Department of Computer Science, Purdue University

Presented at HotCloud '14
June 18, 2014

# Talk Outline

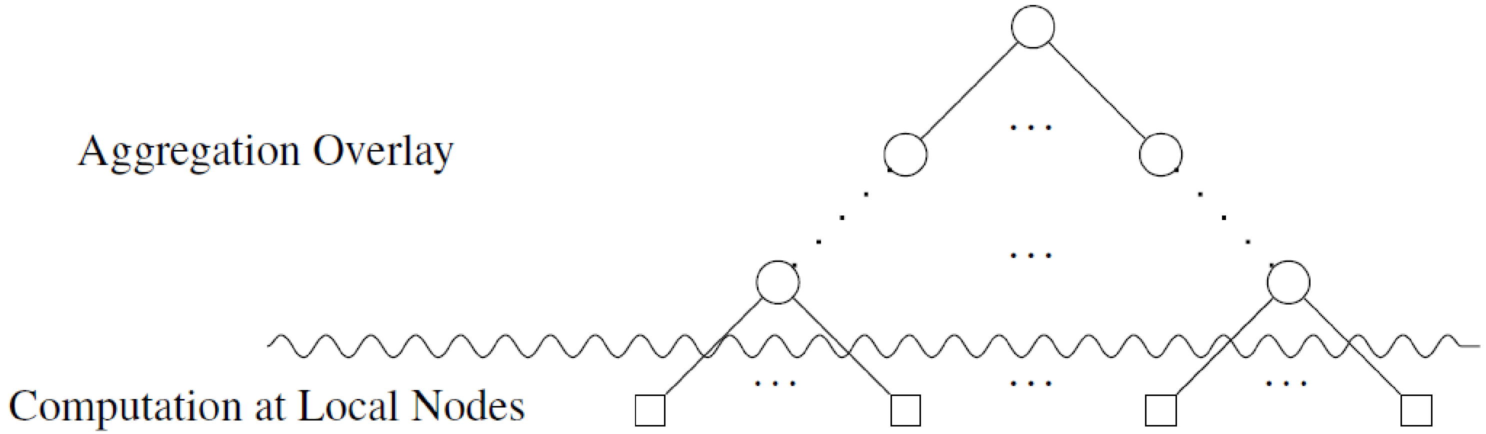
- Motivation and Background
- Model
- Heuristics
- Implementation
- Experimental Setup and Results
- Conclusions and Future Work

# Motivation and Background

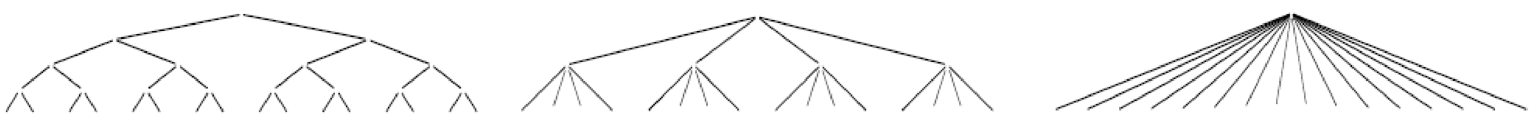
- In-memory Big Data, e.g. RDDs *[Zaharia et al.;NSDI 2012]* Presto *[Venkataraman et al.;Eurosys '13]*
- Aggregation specific *(MapReduceMerge, Yang et al., SIGMOD '07)*
- Minimize latency of tree overlay
- Mathematically modeled optima *[Kim et al.;IEEE Transactions on Aerospace and Electronic Systems 32, 2 ('96)]*
- Minimal analysis and configuration

# Model

- Compute-Aggregate $g(f(x_0) \cdots f(x_n))$




- Customizable fanout



(a) Fanout = 2 (b) Fanout = 4 (c) Fanout = 16

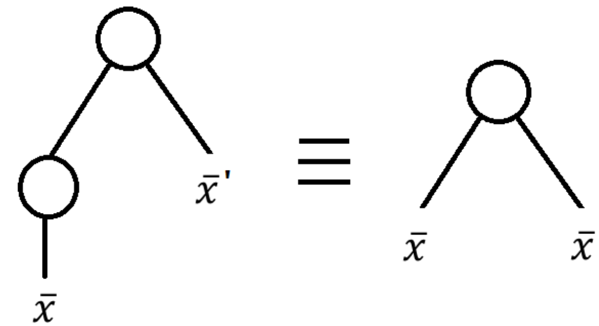
# Examples

- Merge sorted elements
- Min/Max/Average
- Word count
- Top-*k* matching

# Aggregation Function Rules

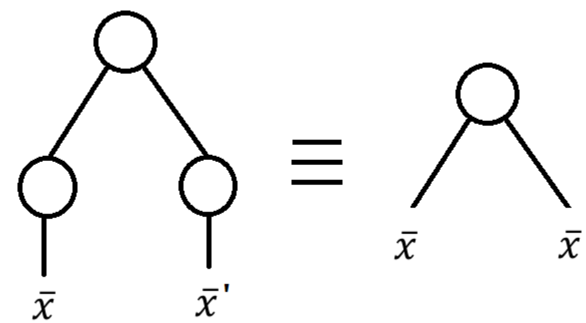
- Associative

$$g(g(\bar{x}), \bar{x}') \equiv g(\bar{x}, \bar{x}')$$



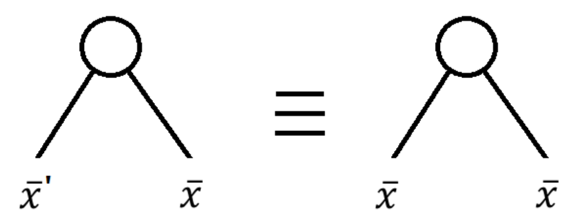

- Cumulative

$$g(g(\bar{x}), g(\bar{x}')) \equiv g(\bar{x}, \bar{x}')$$




- Commutative

$$g(\bar{x}, \bar{x}') \equiv g(\bar{x}', \bar{x})$$



# Assumptions

- Assumptions on latency, not correctness
- Trees – each input included exactly once
- Full and balanced trees
- Monotonic aggregation with respect to size
- Homogenous levels
- Monotonic and constant ratio size changes

# Variables

- $n$ – Number of leaf nodes/inputs
- $d$ – Fanout of aggregation tree
- $\overline{x}$ – Set of inputs (output from computation or prior aggregation)
- $g(\bar{x})$ – Aggregation on $\overline{x}$
- $g^c(\bar{x})$ – Time cost of aggregation function
- $y_0/y$ – Ratio of output size to single input size

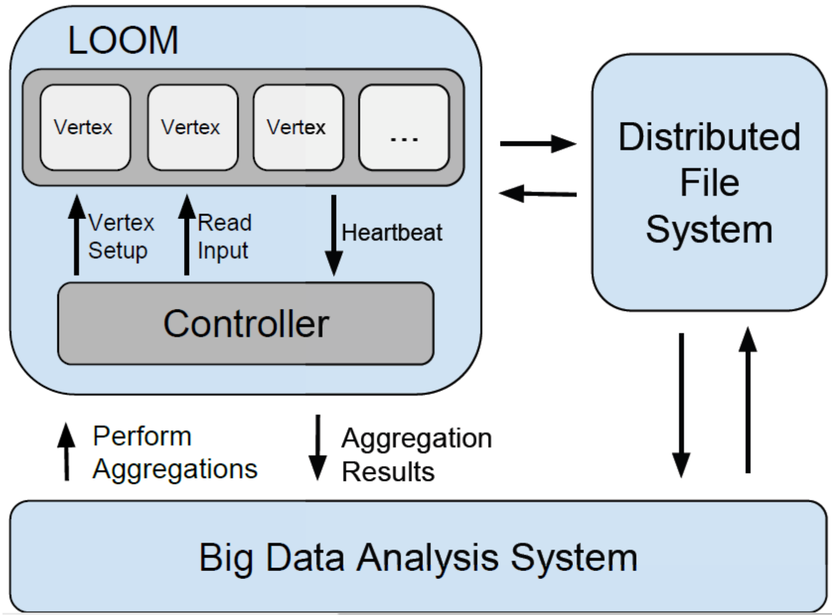
# Heuristics

| $y_0$ | Optimal Fanout | Sublin. $g^c(\overline{x})$ | Linear $g^c(\overline{x})$ | Superlin. $g^c(\overline{x})$ |
|---|---|---|---|---|
| $y_0 < 1$ | $2$ | | √ | √ |
| $y_0 = 1$ | $e$ | | √ | * |
| $1 < y_0 < n$ | $\min(n, (1 - \log_n y_0)^{-\log_{y_0} n})$ | | √ | |
| $y_0 > n$ | $n$ | √ | √ | √ |

√ - Proven Optima
* - Proven Near-Optima

# Implementation

- Independent aggregation subsystem

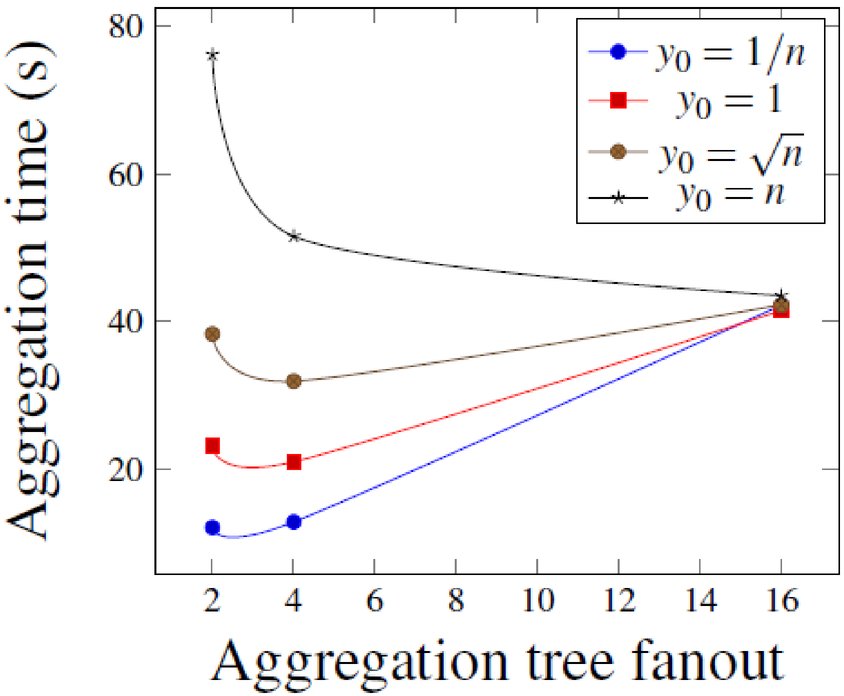



- Fifth main operation `parallelAggregate()` in FlumeJava *[Chambers et al.;PLDI '10]*

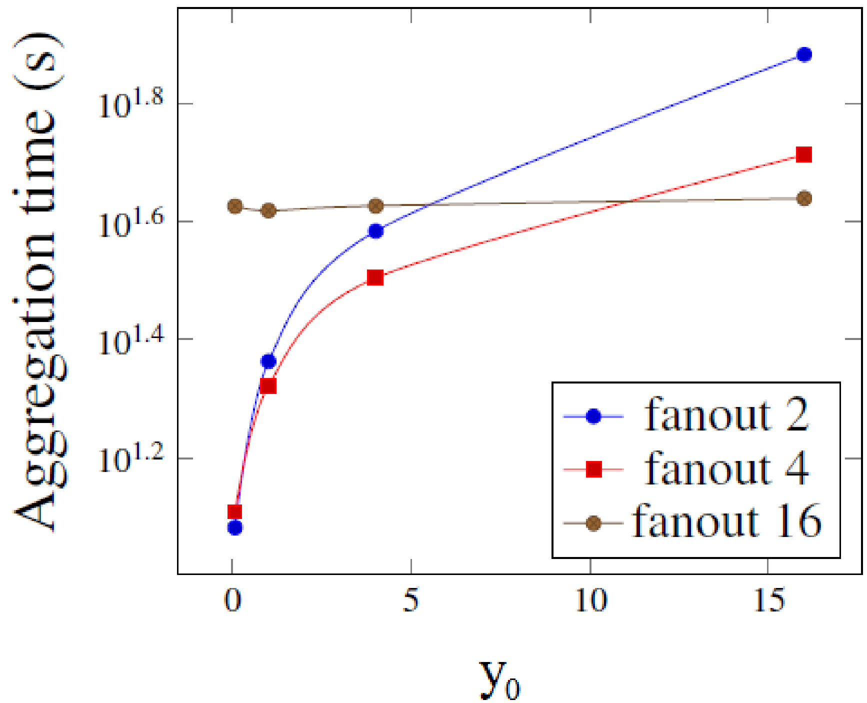
# Experimental Setup

- Amazon EC2
- Maintained assumptions (full and balanced)
- Microbenchmarks
  - Generated data and simulated linear aggregation
  - 16 leaves
- Real world applications
  - Word count and top-*k* match on Yahoo! Hadoop cluster logs
  - 16 and 64 leaves

# Results (Microbenchmarks – 1/2)

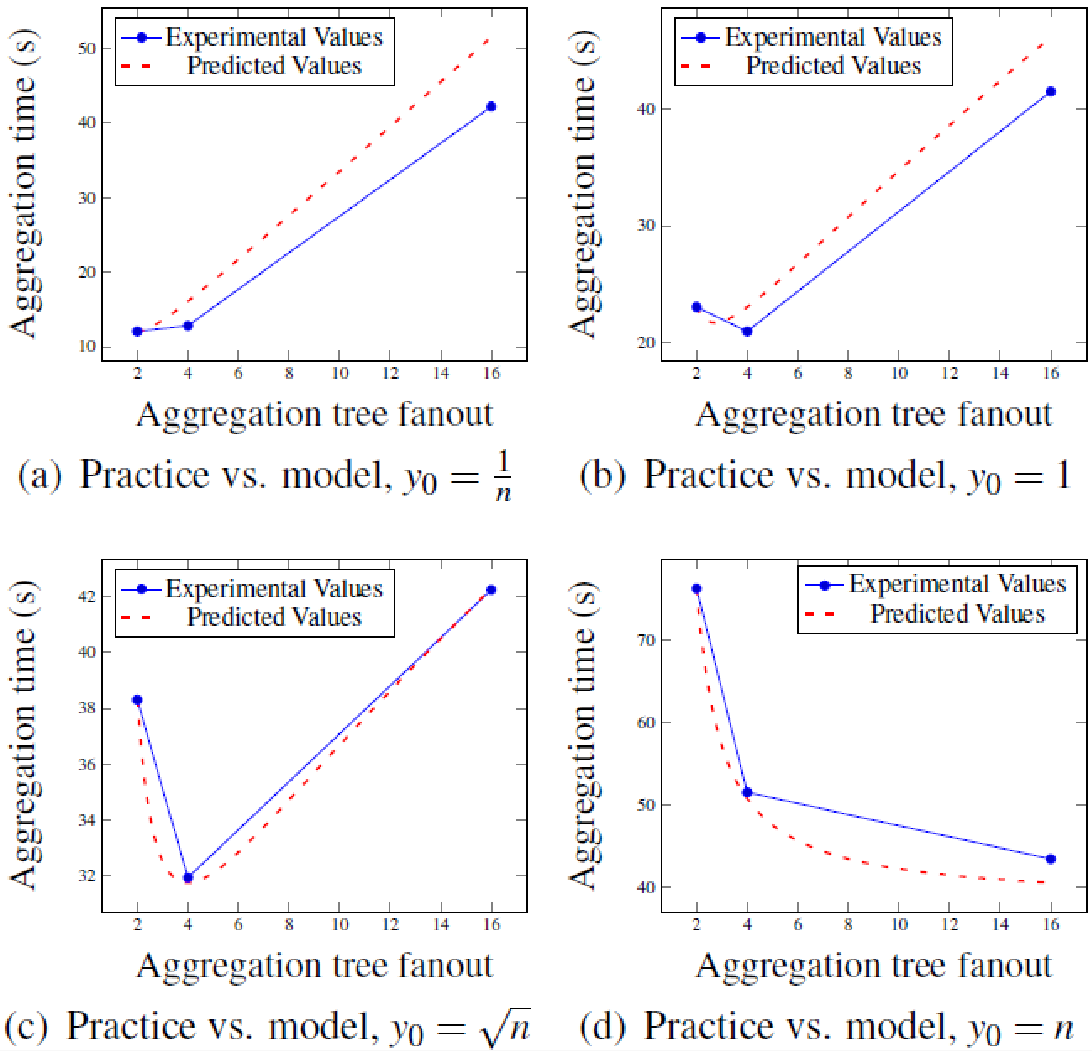



(a) Series as size ratio




(b) Series as fanout

# Results (Microbenchmarks - 2/2)



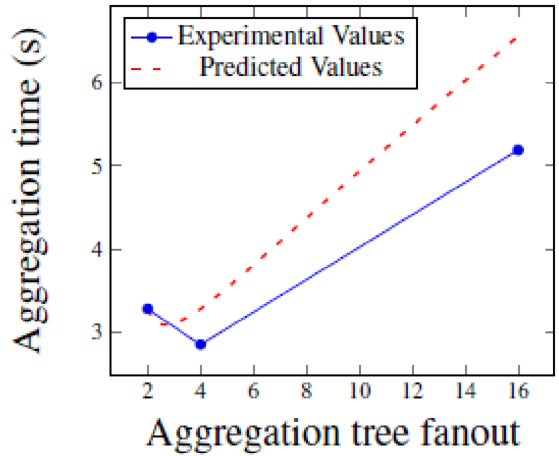

(a) Practice vs. model, $y_0 = \frac{1}{n}$

(b) Practice vs. model, $y_0 = 1$

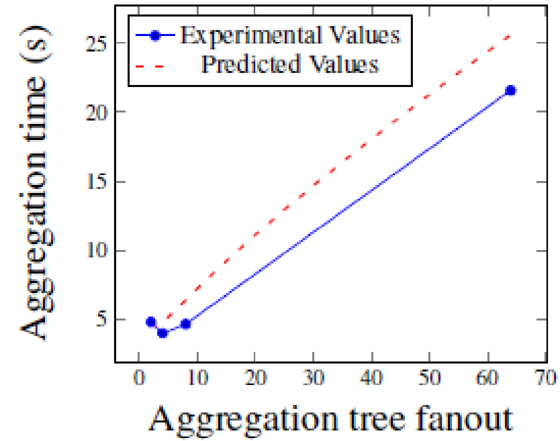
(c) Practice vs. model, $y_0 = \sqrt{n}$

(d) Practice vs. model, $y_0 = n$

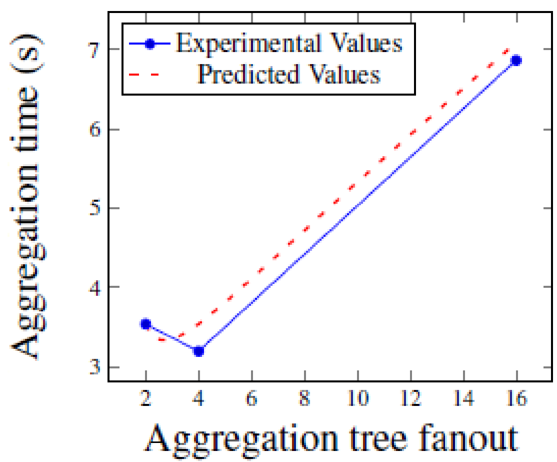
# Results (Applications)



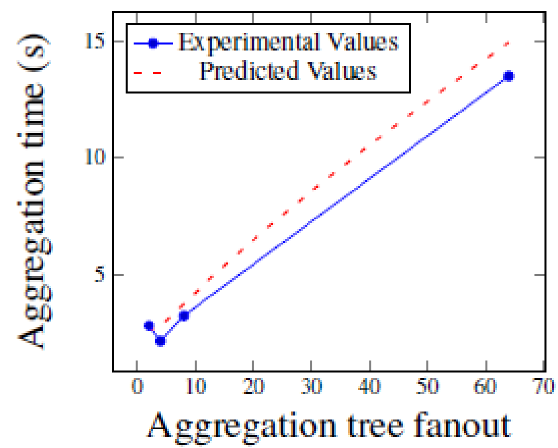

(a) Top-$k$ match, $n = 16$




(b) Top-$k$ match, $n = 64$




(c) Word count, $n = 16$




(d) Word count, $n = 64$

# What we have done

- Codified compute-aggregation definition
- Mathematically modeled aggregation time
- Provided heuristics for lightweight optimization
- Results usable even without our system with known $y_0$
- Implemented subsystem with FlumeJava
- Experimentally validated modeled optima

# What we are going to do

- Study the currently unproven cases
- Determine a good way to find/specify $y_0$ (preferably automatically)
- Expand the limits of the testing
- Deal with broken assumptions
- Deal with heterogeneity
- Work on streaming inputs

# Questions?