Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks

Talk structure: technical meat, then criticism

Dryad Goal

- Create a general-purpose distributed data flow execution platform
- Less restrictive semantics than MapReduce framework
- Extract parallelism from dependencies, not from within subroutines

Dryad Model

- Subroutines are vertices
- Communication channels are edges

Dryad Model

- Subroutines are vertices
  - Programs created from "factories"
    - Some pre-defined vertex classes (e.g., *map*, *reduce*)
- Communication channels are edges
  - Transmit structured but untyped *items*
  - TCP, disk, memory pipes supported

Dryad Architecture

- Job manager schedules vertices on machines
  - Greedy algorithm
  - Vertices are deterministic, and graph is acyclic, so manager can easily restart
  - Runtime manager can reschedule vertices for better locality (local disk or memory)
  - Graphs manually constructed...

Graph Operators

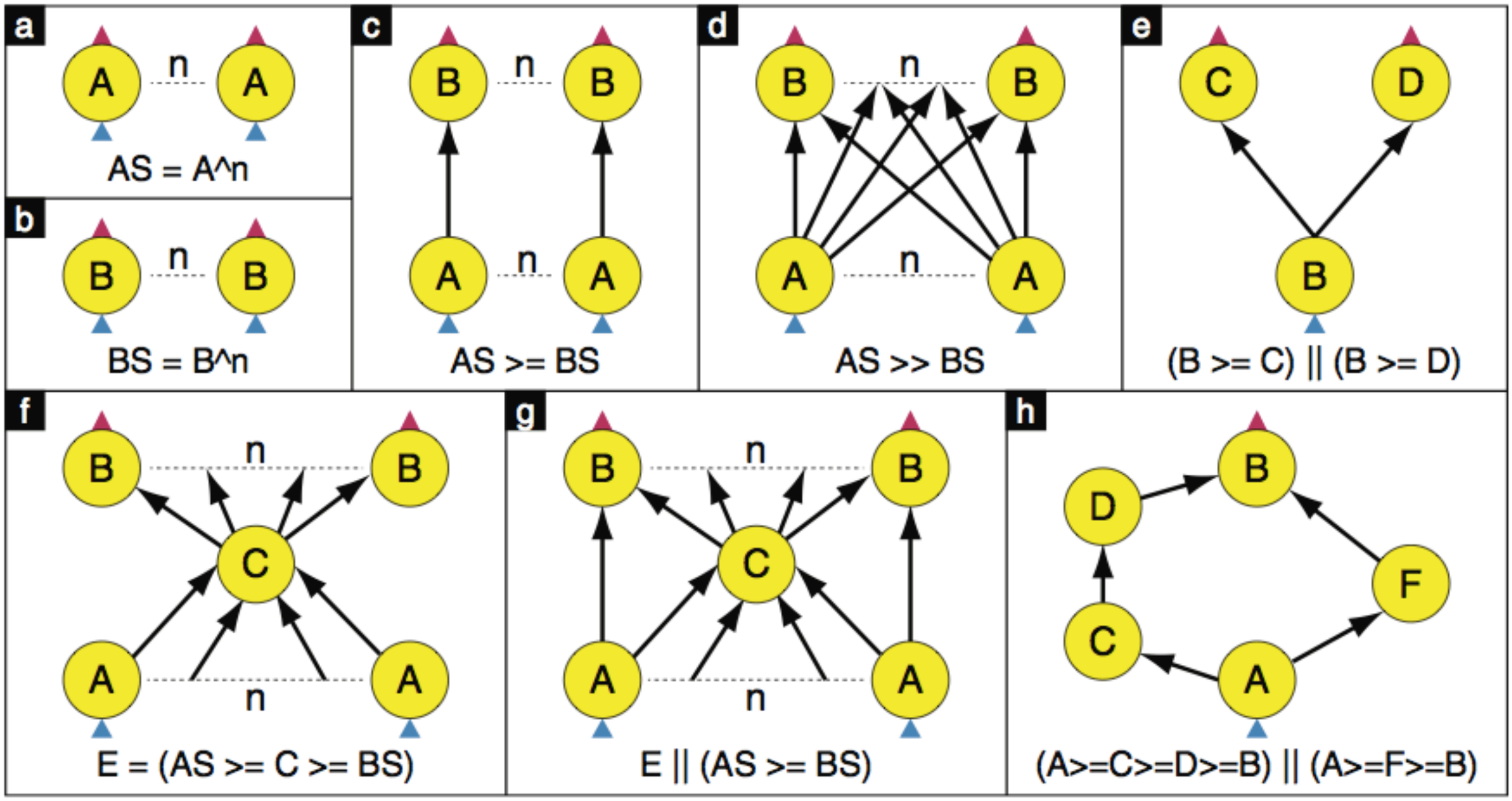


Figure 3: The operators of the graph description language. Circles are vertices and arrows are graph edges. A triangle at the bottom of a vertex indicates an *input* and one at the top indicates an *output*. Boxes (a) and (b) demonstrate cloning individual vertices using the ^ operator. The two standard connection operations are pointwise composition using >= shown in (c) and complete bipartite composition using >> shown in (d). (e) illustrates a merge using ||. The second line of the figure shows more complex patterns. The merge in (g) makes use of a "subroutine" from (f) and demonstrates a bypass operation. For example, each A vertex might output a summary of its input to C which aggregates them and forwards the global statistics to every B. Together the B vertices can then distribute the original dataset (received from A) into balanced partitions. An asymmetric fork/join is shown in (h).

Refinement

- "If a computation is associative and commutative, and performs a data reduction, then it can benefit from an aggregation tree."
  - Um, how do you detect this automatically?
  - Unclear if implemented...

Refinement

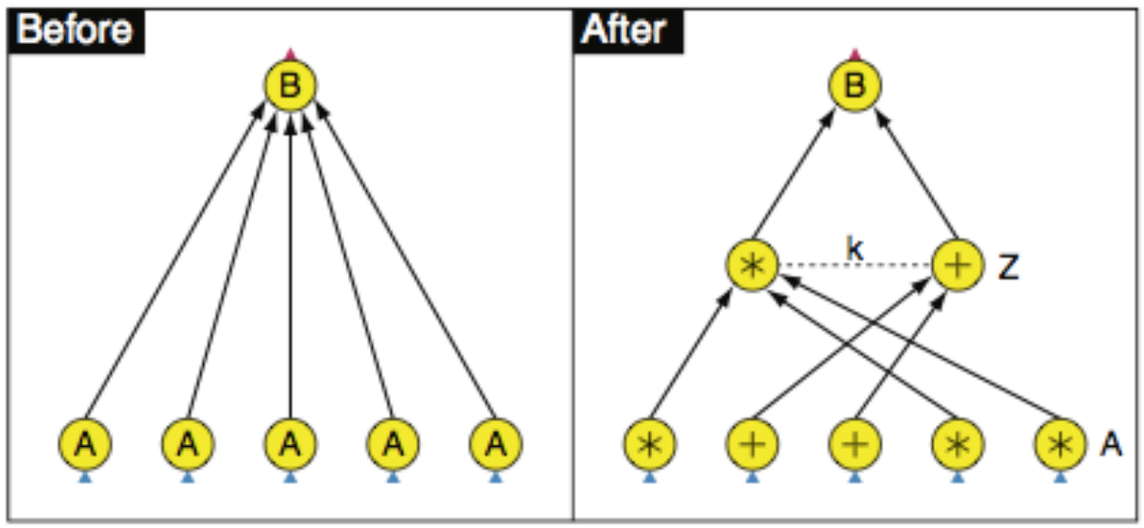


Figure 6: A dynamic refinement for aggregation. The logical graph on the left connects every input to the single output. The locations and sizes of the inputs are not known until run time when it is determined which computer each vertex is scheduled on. At this point the inputs are grouped into subsets that are close in network topology, and an internal vertex is inserted for each subset to do a local aggregation, thus saving network bandwidth. The internal vertices are all of the same user-supplied type, in this case shown as "Z." In the diagram on the right, vertices with the same label ('+' or '$*$') are executed close to each other in network topology.

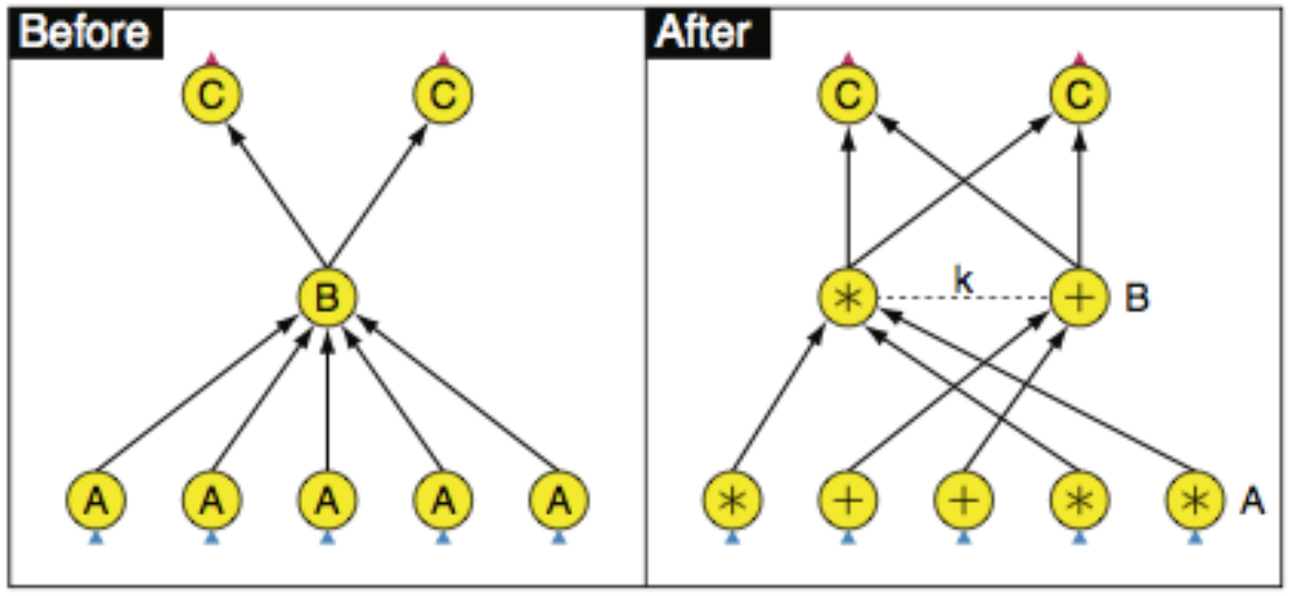


Figure 7: A partial aggregation refinement. Following an input grouping as in Figure 6 into k sets, the successor vertex is replicated k times to process all the sets in parallel.

Evaluation: in discussion section

Criticism intended as a means to discussion, not as definitive verdict

(read: I'm not this much of a jerk in real life)

Criticism intended as a means to discussion, not as definitive verdict

(read: I'm not this much of a jerk in real life--I think)

One plausible (?) interpretation of events

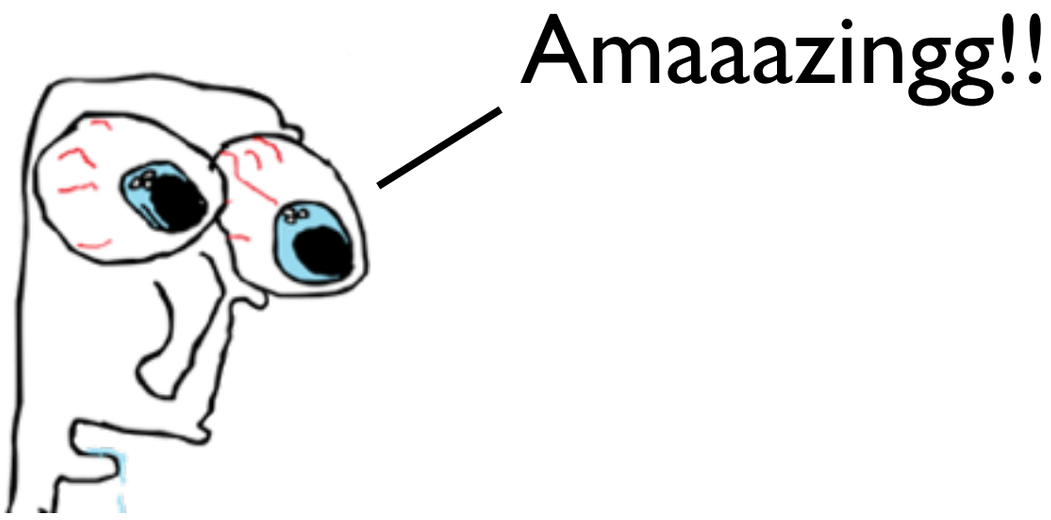
GOOG: Here's MapReduce!

Published 2004

GOOG: Here's MapReduce!

Published 2004

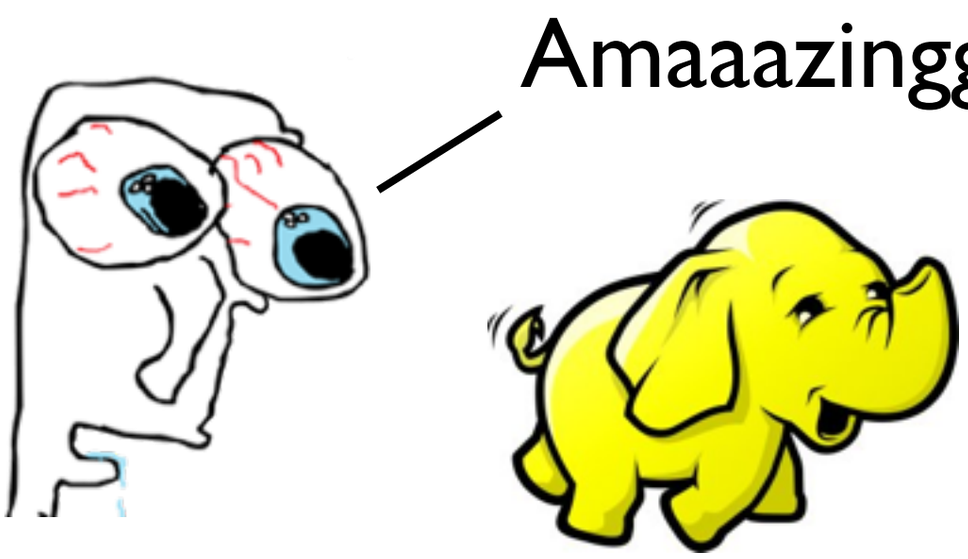
The World:



GOOG: Here's MapReduce!

Published 2004

The World:



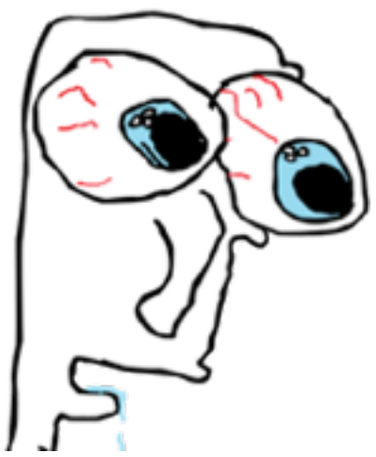
Hadoop born 2007

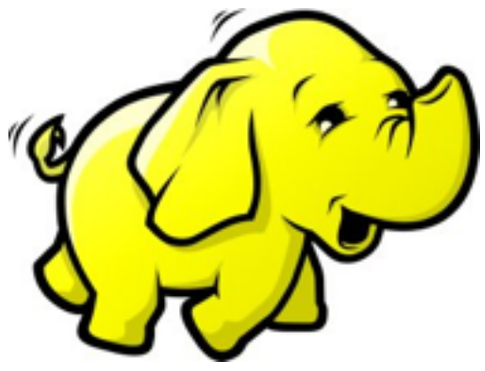
GOOG: Here's MapReduce!

Published 2004

The World:

Amaaazingg!!





Hadoop born 2007

M$R: US TOO, GUYS!

Dryad born 2007

Which do you prefer?

```
select distinct p.objID
from photoObjAll p
join neighbors n    — call this join "X"
on p.objID = n.objID
  and n.objID < n.neighborObjID
  and p.mode = 1
join photoObjAll l — call this join "Y"
on l.objid = n.neighborObjID
  and l.mode = 1
  and abs((p.u-p.g)-(l.u-l.g))<0.05
  and abs((p.g-p.r)-(l.g-l.r))<0.05
  and abs((p.r-p.i)-(l.r-l.i))<0.05
  and abs((p.i-p.z)-(l.i-l.z))<0.05
```

SQL

We mapped the query to the Dryad computation shown in Figure 2. Both data files are partitioned into n approximately equal parts (that we call U_1 through U_n and N_1 through N_n) by **objID** ranges, and we use custom C++ item objects for each data record in the graph. The vertices X_i (for $1 \le i \le n$) implement join "X" by taking their partitioned U_i and N_i inputs and merging them (keyed on **objID** and filtered by the $<$ expression and **p.mode=1**) to produce records containing **objID**, **neighborObjID**, and the color columns corresponding to **objID**. The D vertices distribute their output records to the M vertices, partitioning by **neighborObjID** using a range partitioning function four times finer than that used for the input files. The number four was chosen so that four pipelines will execute in parallel on each computer, because our computers have four processors each. The M vertices perform a non-deterministic merge of their inputs and the S vertices sort on **neighborObjID** using an in-memory Quicksort. The output records from $S_{4i-3} \ldots S_{4i}$ (for $i = 1$ through n) are fed into Y_i where they are merged with another read of U_i to implement join "Y". This join is keyed on **objID** (from U) = **neighborObjID** (from S), and is filtered by the remainder of the predicate, thus matching the colors. The outputs of the Y vertices are merged into a hash table at the H vertex to implement the **distinct** keyword in the query. Finally, an enumeration of this hash table delivers the result. Later in the paper we include more details about the implementation of this Dryad program.

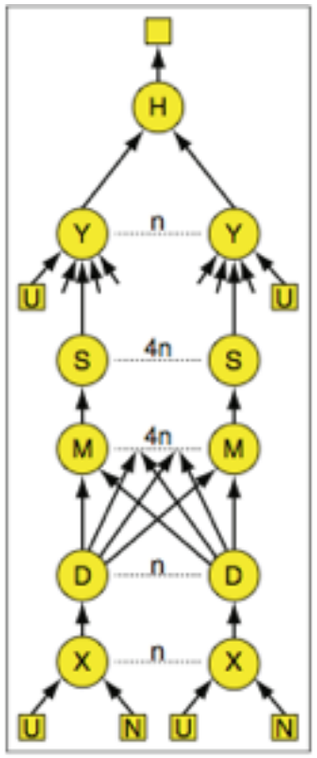


Figure 2: The communication graph for an SQL query. Details are in Section 2.1.

Dryad, in English

Which do you prefer?

- "A programmer can master the APIs required for most of the applications **in a couple of weeks**." (emph. added)
  - I can teach my (hypothetical) toddler to MapReduce in an afternoon.
- "Dryad is not a database engine; it does not include a query planner or optimizer"-- Damn! I sure wish it did...

Which do you prefer?

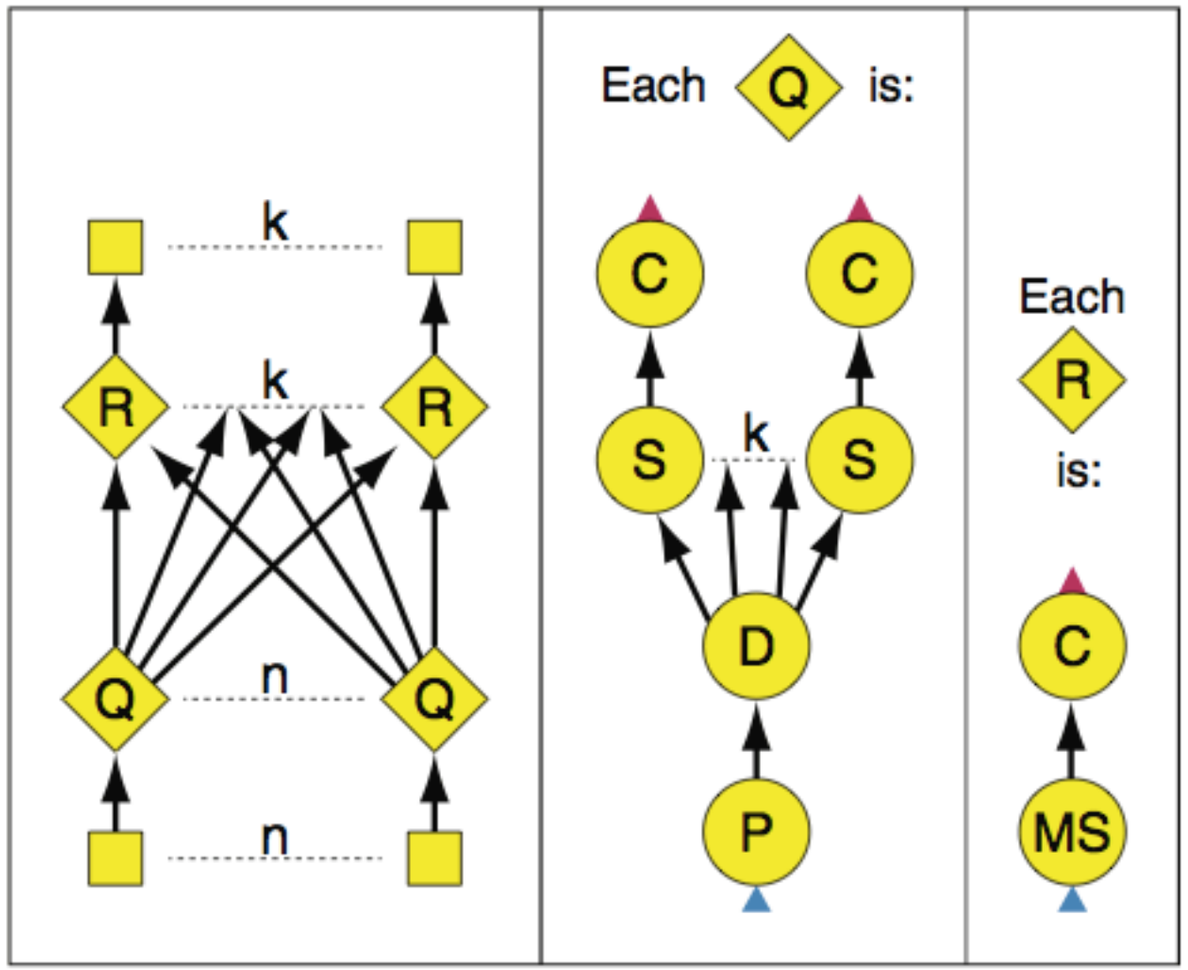


Figure 9: The communication graph to compute a query histogram. Details are in Section 6.3. This figure shows the first cut "naive" encapsulated version that doesn't scale well.

Ehh, this doesn't look too "naïve"!

Which do you prefer?

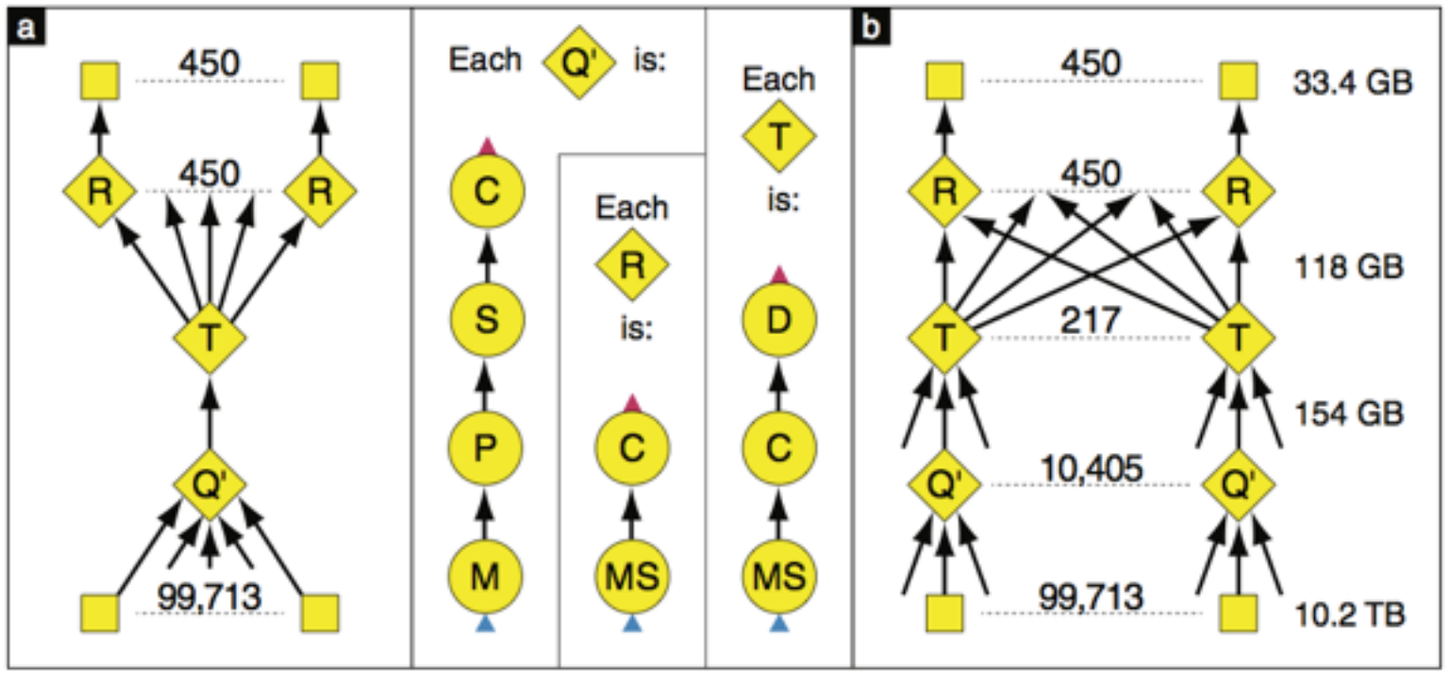
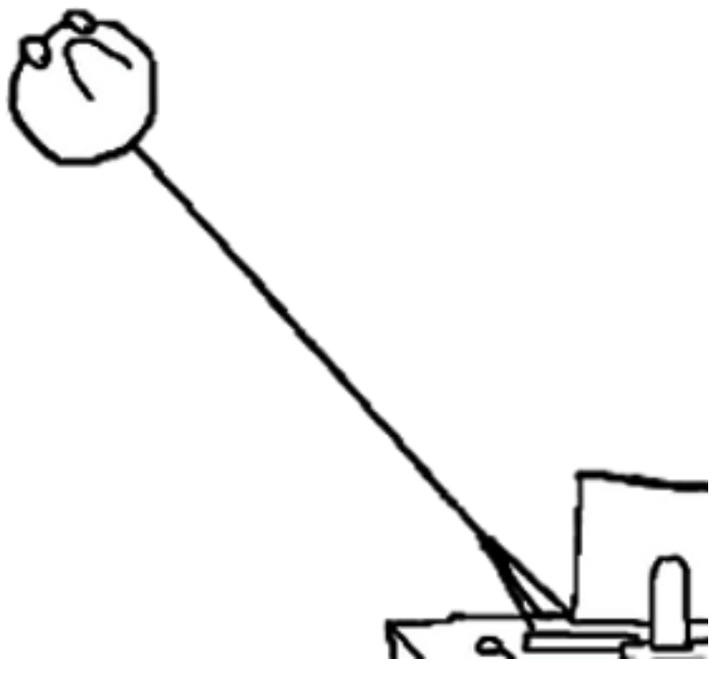


Figure 10: Rearranging the vertices gives better scaling performance compared with Figure 9. The user supplies graph (a) specifying that 450 buckets should be used when distributing the output, and that each Q' vertex may receive up to 1GB of input while each T may receive up to 600MB. The number of Q' and T vertices is determined at run time based on the number of partitions in the input and the network locations and output sizes of preceding vertices in the graph, and the refined graph (b) is executed by the system. Details are in Section 6.3.

Which do you prefer?



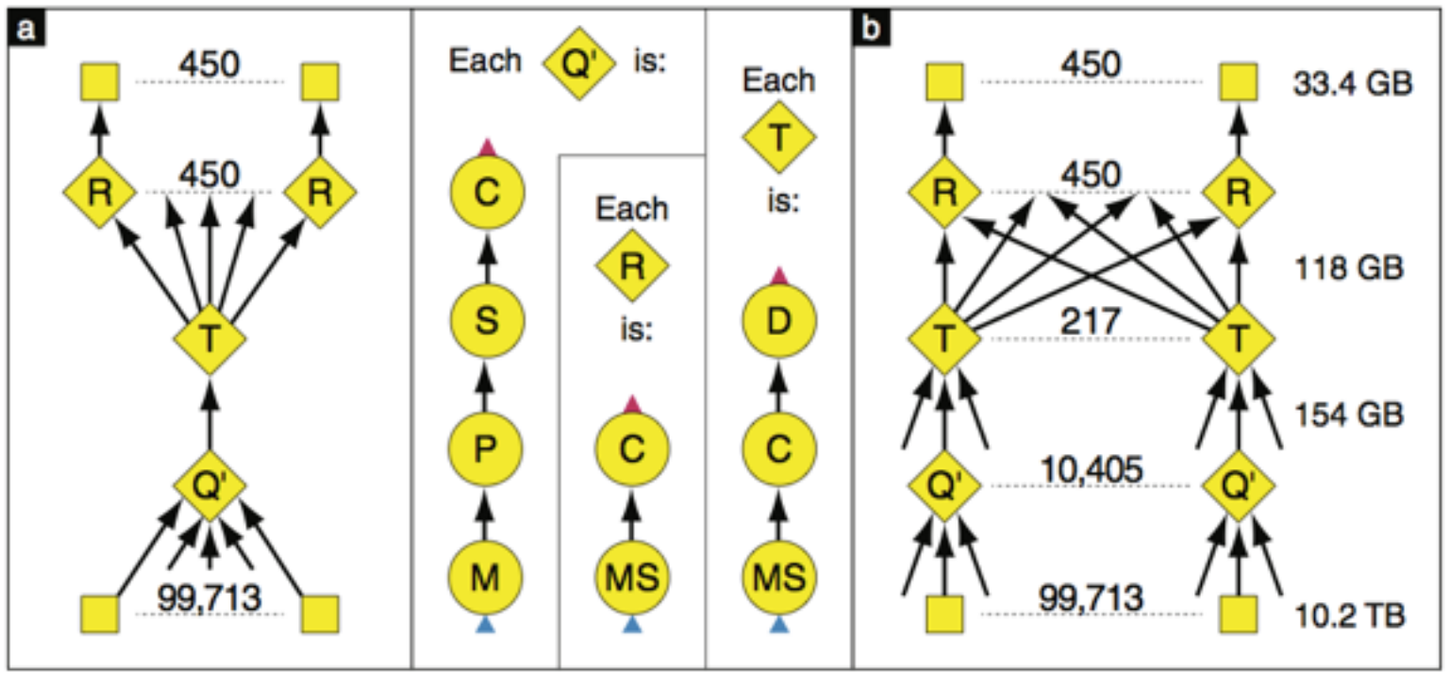


Figure 10: Rearranging the vertices gives better scaling performance compared with Figure 9. The user supplies graph (a) specifying that 450 buckets should be used when distributing the output, and that each Q' vertex may receive up to 1GB of input while each T may receive up to 600MB. The number of Q' and T vertices is determined at run time based on the number of partitions in the input and the network locations and output sizes of preceding vertices in the graph, and the refined graph (b) is executed by the system. Details are in Section 6.3.

manually optimized...

Why not MapReduce?

- Restrictive semantics
- Pipelining Map/Reduce stages possibly inefficient
- Solves problems within a narrow programming domain well

Why not MapReduce?

- DB community: our parallel RDBMSs have been doing this forever...
  - cf. Stonebraker
  - Not this paper's approach

Why not MapReduce?

- DB community: our parallel RDBMSs have been doing this forever...
  - cf. Stonebraker
  - Not this paper's approach

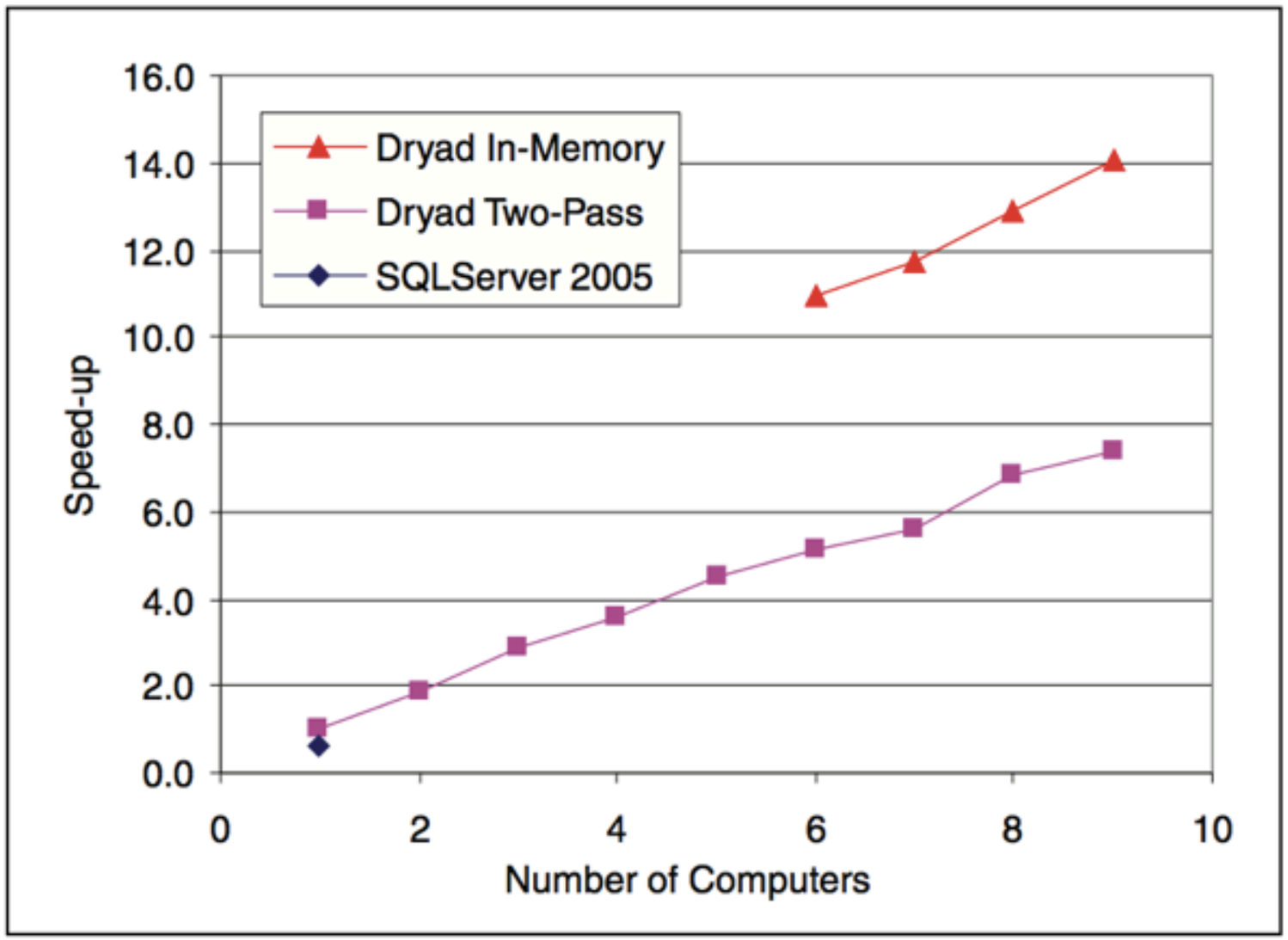
Are these just a bunch of old database guys complaining that no one uses their stuff?

Wow! It looks like MR there's a lot of room for improvement...

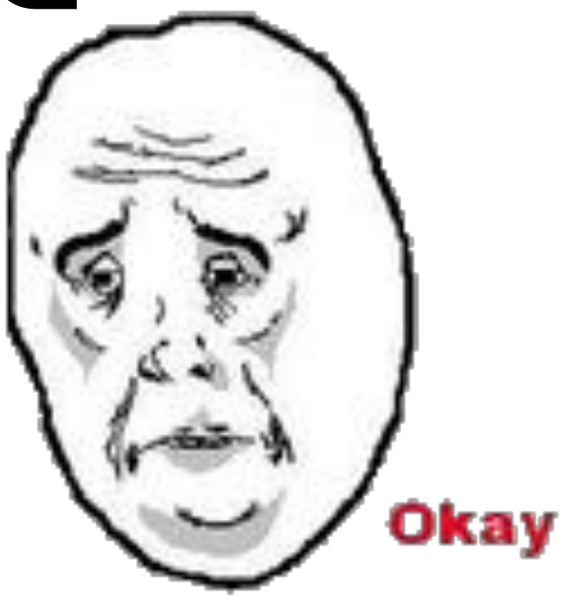
Wow! It looks like MR there's a lot of room for improvement...

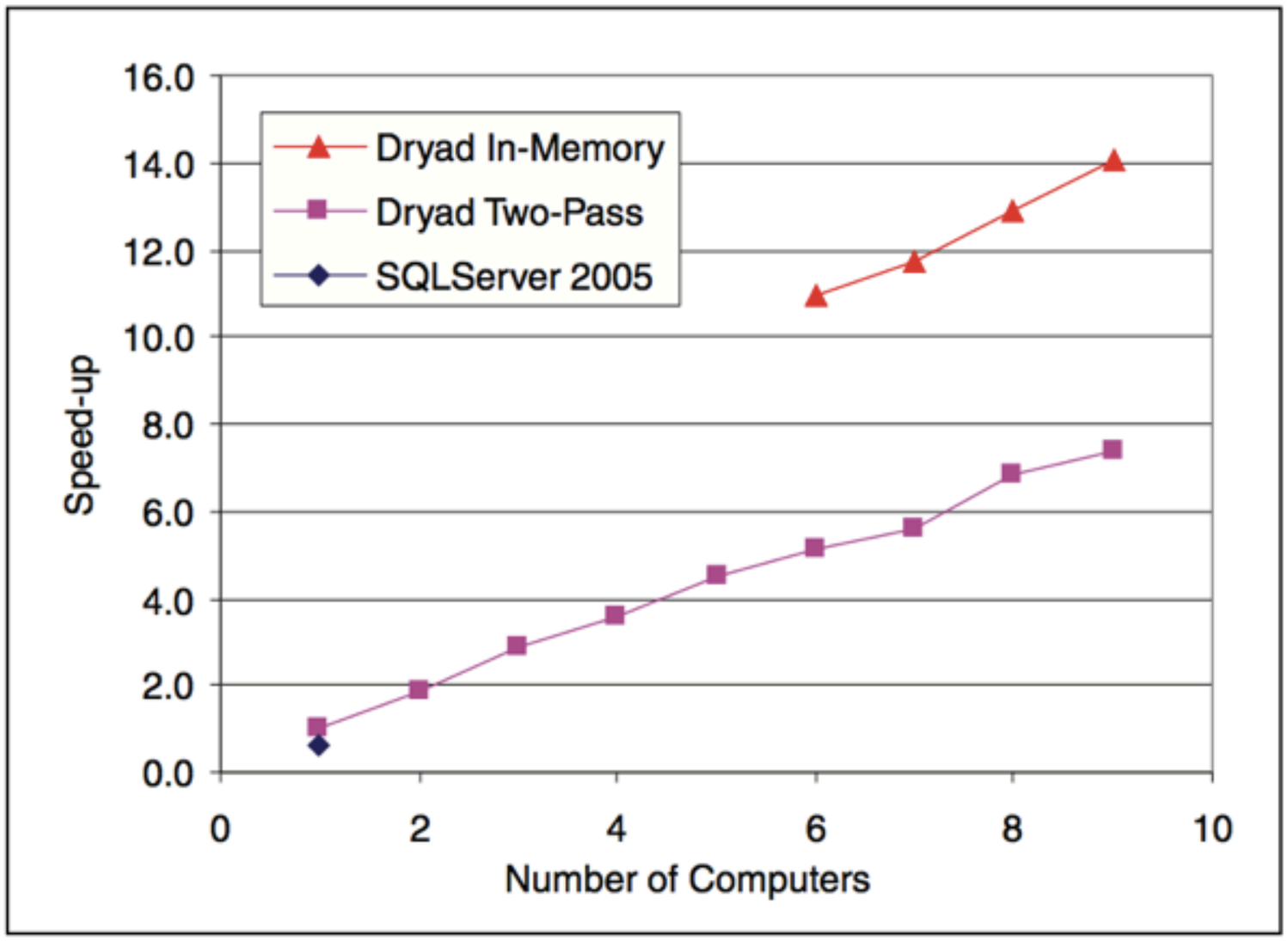
...too bad the authors didn't make an effort to demonstrate this

Evaluation: a missed opportunity

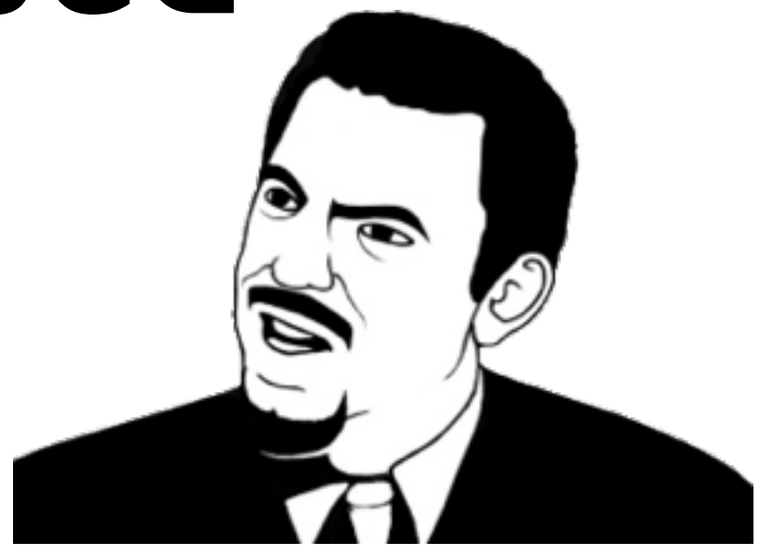


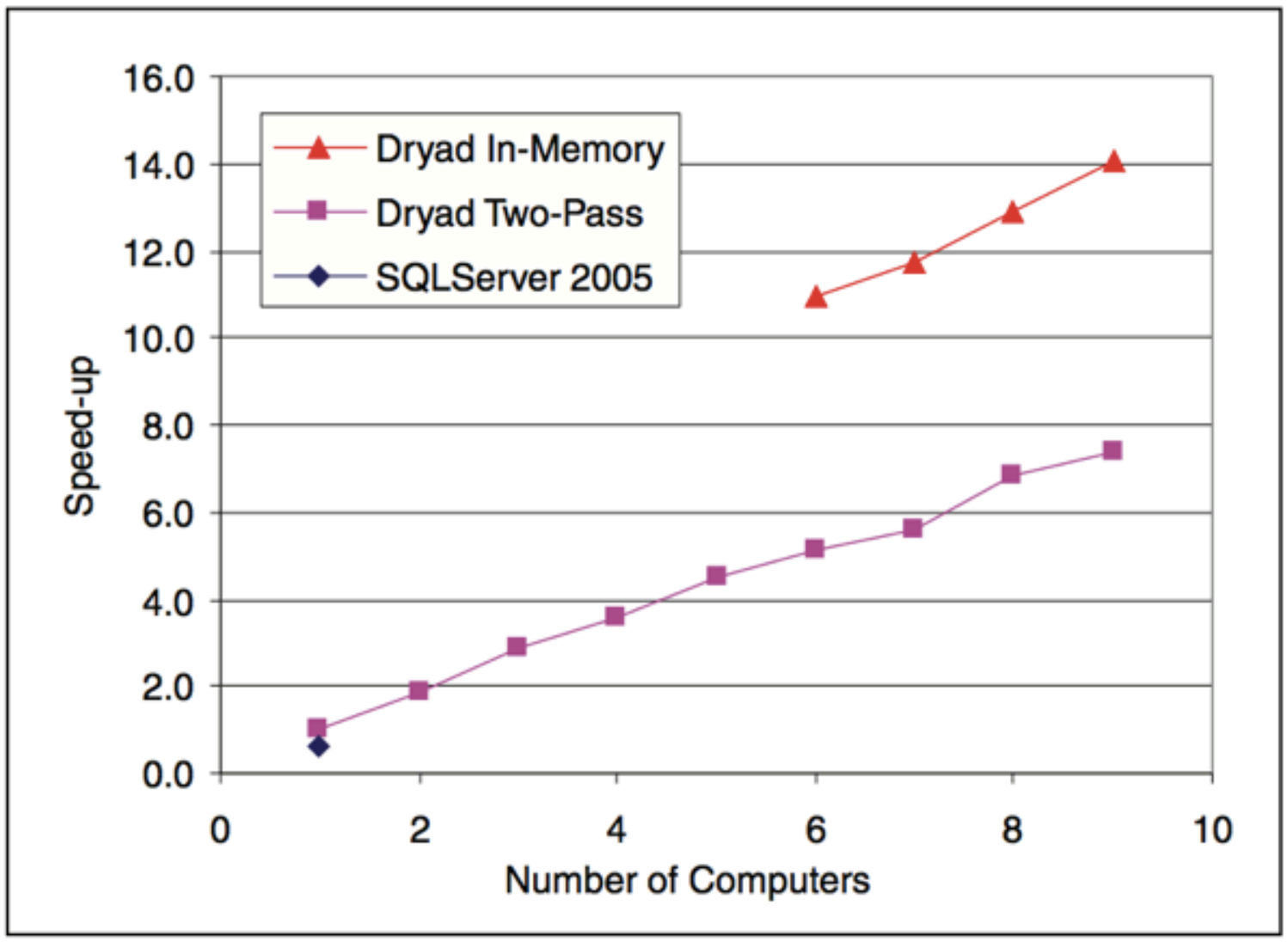
Evaluation: a missed opportunity



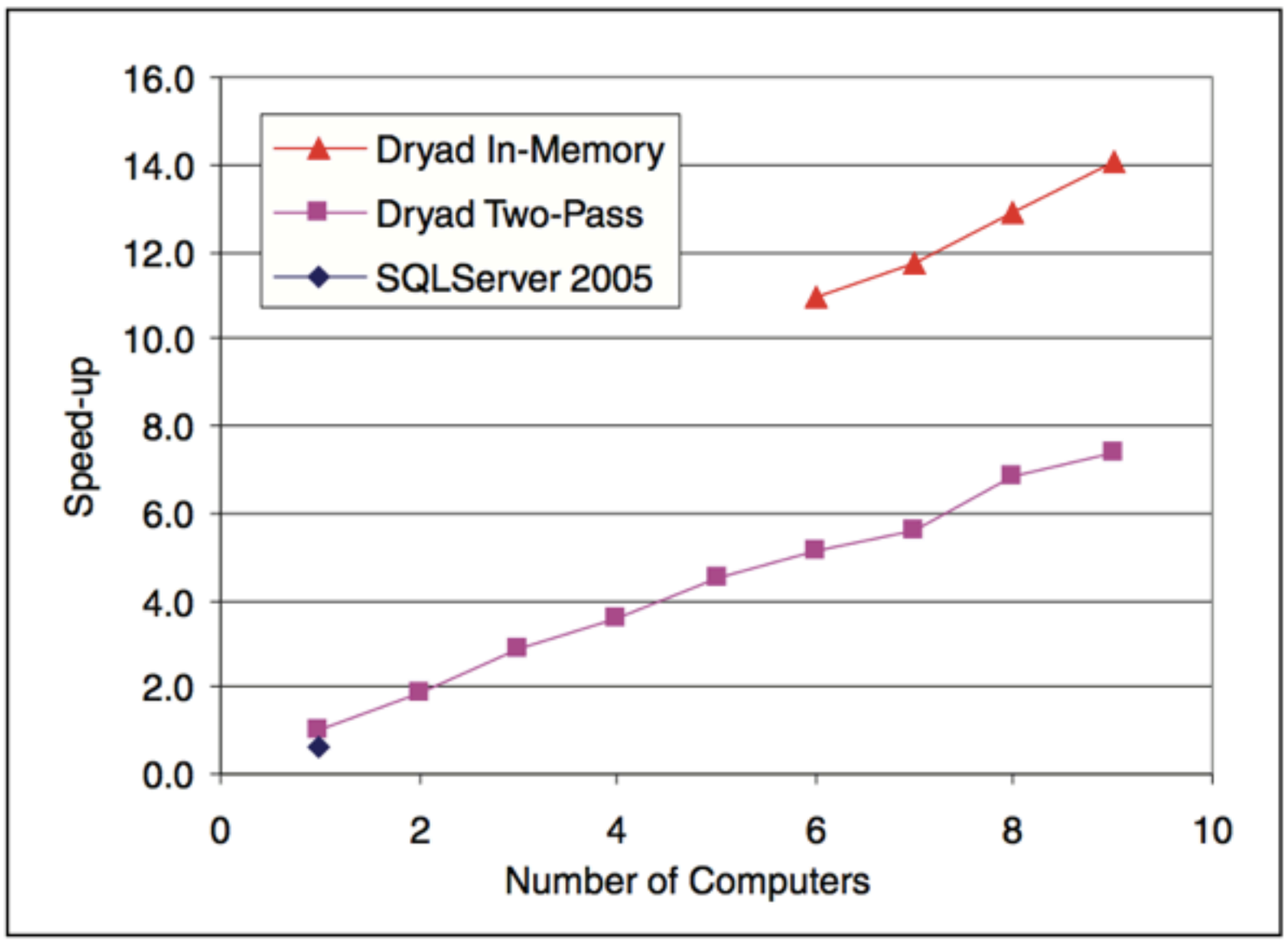


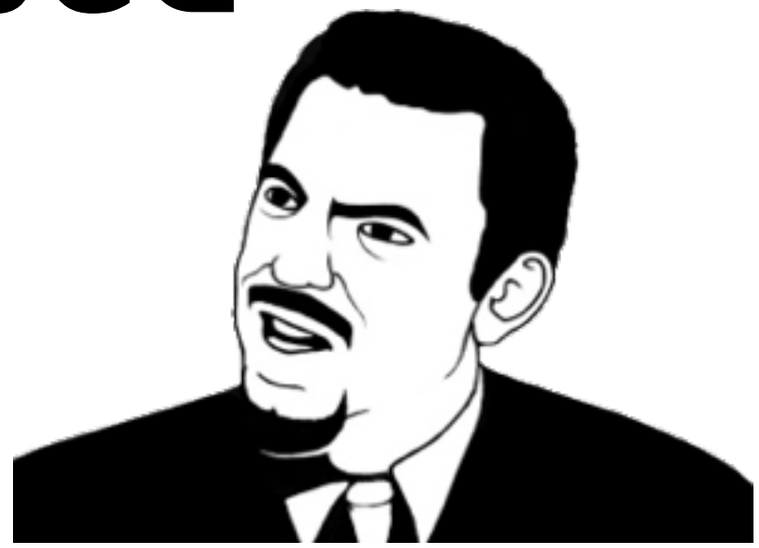
Evaluation: a missed opportunity





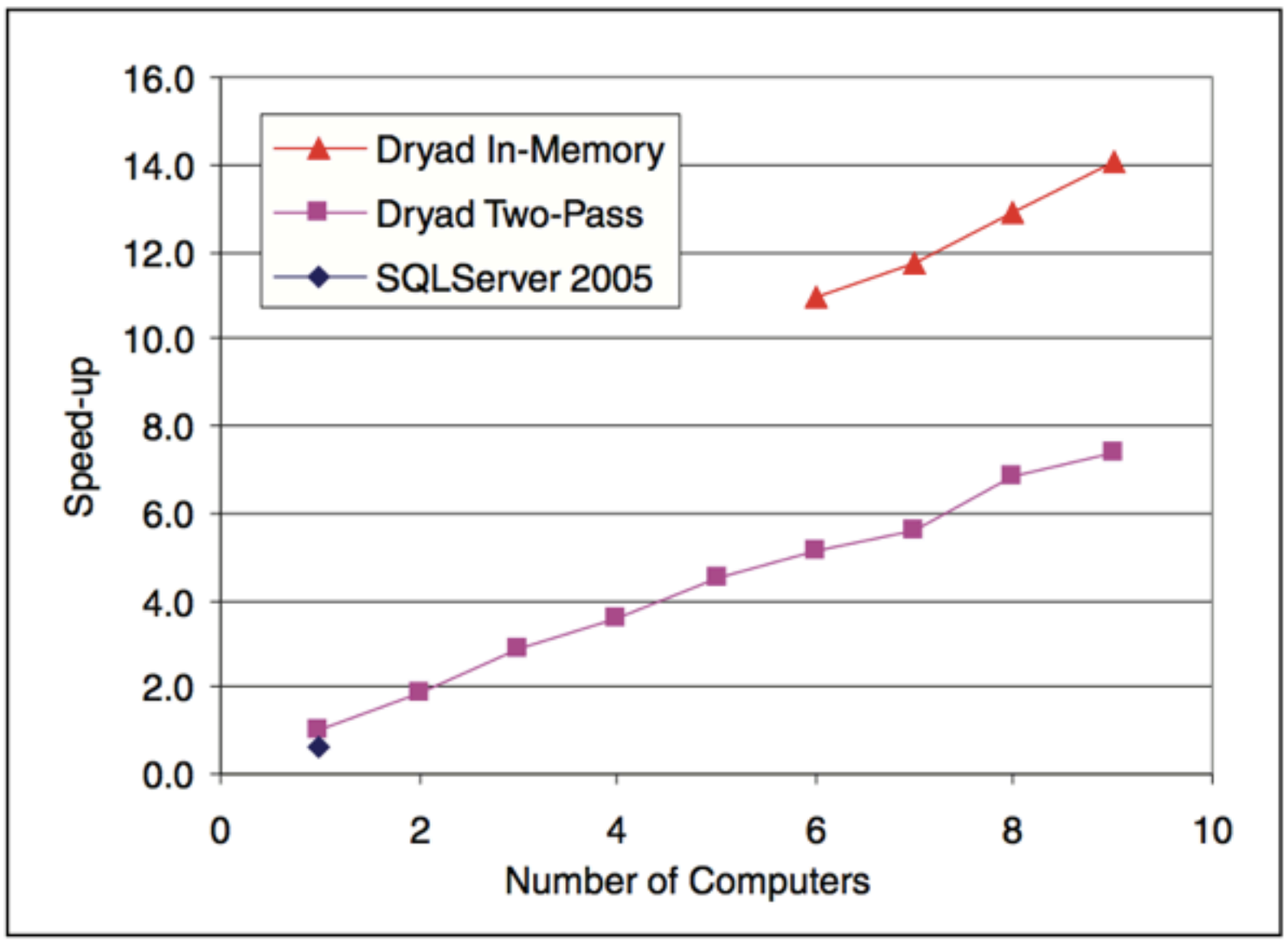
Evaluation: a missed opportunity

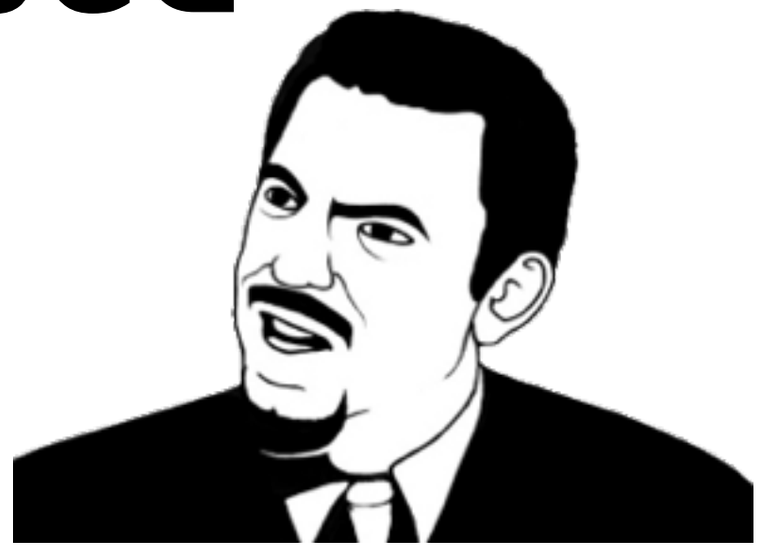




Where's the comparison to a more restrictive framework?

Evaluation: a missed opportunity

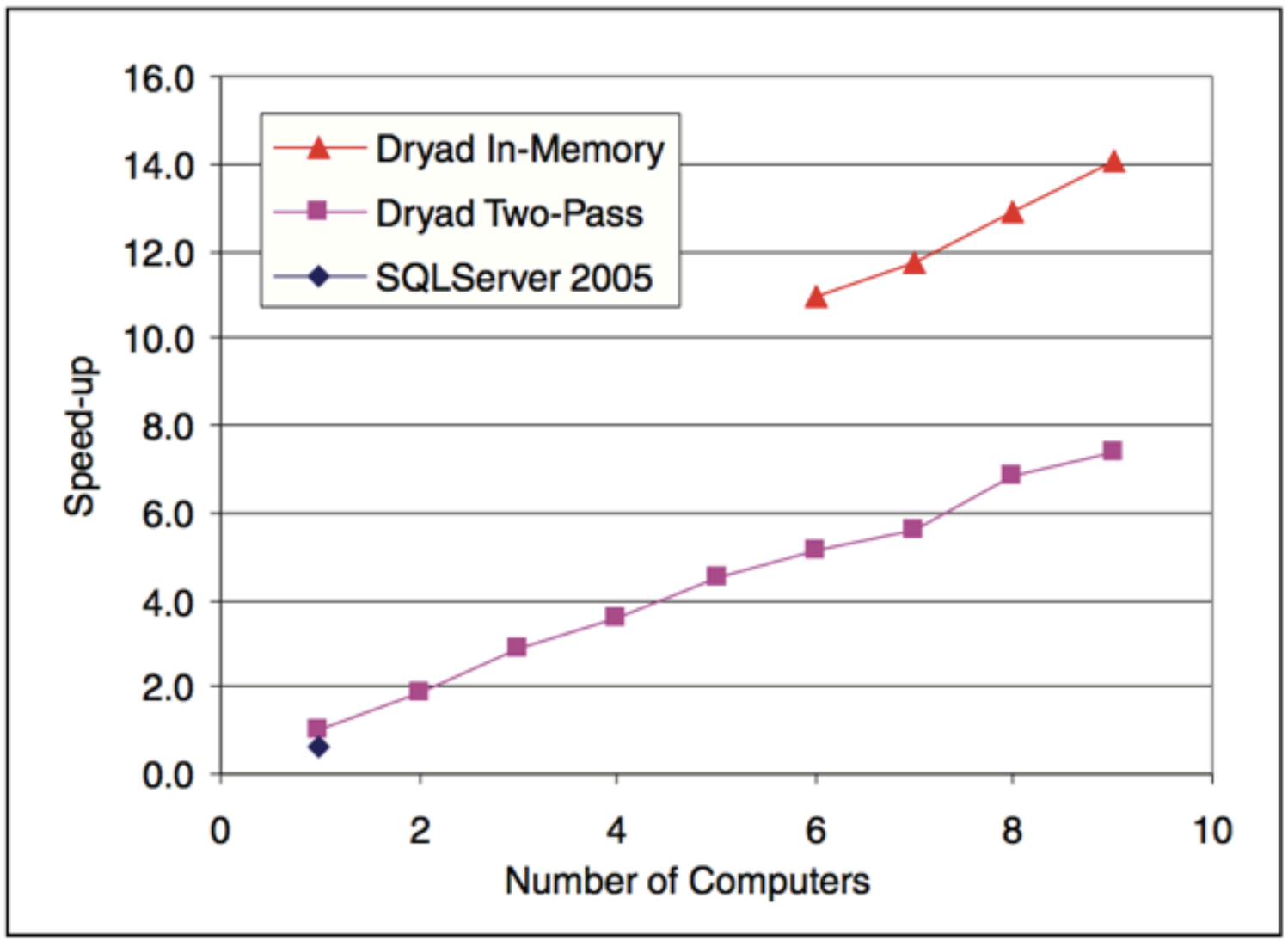


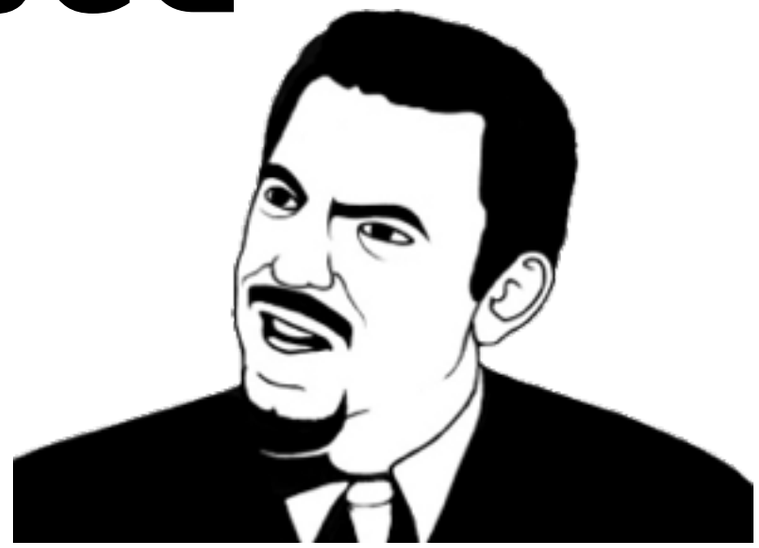


Where's the comparison to a more restrictive framework?

You beat single-node M$ SQL Server! Congratulations! <pat on back>

Evaluation: a missed opportunity





Where's the comparison to a more restrictive framework?

You beat single-node M$ SQL Server! Congratulations! <pat on back>

How does that scaling graph change when we go to 100 computers? 1000?

Evaluation: a missed opportunity

- They couldn't compare to a MR implementation, but they could try to approximate one...
- What about lines of code/program complexity?
- What about demonstrating fault-tolerance?
- What about comparing against a parallel RDBMS?
  - Probably makes M$ $QL $erver look bad
  - Science & truth versus commercial expediency?

There are these funny things called gigabytes...

There are these funny things called gigabytes...

10,160,519,065,748 Bytes

9462 GB

153,703,445,725 Bytes

143 GB

118,364,131,628 Bytes

110 GB

33,375,616,713 Bytes.

31 GB

Even if the authors didn't show me what I wanted, can I get something out of the paper? Is there a lesson here?

Design

What was surprising here?

What would you have done differently?

Design

What was surprising here?

What would you have done differently?

Is this obvious?

Design

What was surprising here?

What would you have done differently?

Is this obvious?

Caveat: sometimes the best solutions are obvious in retrospect.

Is this one of them?

Systems gurus: what do you think about Dryad versus the dataflow programming of the 70s, 80s, and 90s?

Adoption

- e-Science =?= “no one at M$ even uses this stuff”
  - Data mining uses logs, but isn’t necessarily in use
- Scheduler assumes job “is the only job running on the cluster”
- Nebula == “very popular ‘front end’”

Adoption

- e-Science =?= “no one at M$ even uses this stuff”
  - Data mining uses logs, but isn’t necessarily in use
- Scheduler assumes job “is the only job running on the cluster”
- Nebula == “very popular ‘front end’”

MapReduce: designed for production use
M$R: designed as research (?)
(not a bad thing, but worth acknowledging)

Adoption

No freely available Dryad implementation

(also, runs on M$ stack)

Future prospects

- This seems useful and could likely beat MapReduce
- Higher-level languages key
  - Plug in Dryad as Pig backend--screaming perf?
- Good idea, bad implementation?
  - M$ platform != FLOSSy goodness that makes Hadoop so popular
- Still waiting for killer “here’s when MR sucks” paper...

Future prospects

Props to authors for “Building on Dryad” section #futureworkmeansbrokenpromises

There is light in the darkness!!!!

Progress!

- "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language", OSDI 2008
- "SCOPE: Easy and efficient parallel processing of massive data sets", VLDB 2008
- "Distributed Data-Parallel Computing Using a High-Level Programming Language", SIGMOD 2009
- "Distributed Aggregation for Data-Parallel Computing: Interfaces and Implementations", SOSP 2009

Progress!

These researchers (and their colleagues) **actually** followed through

Maybe this wasn't the paper I'd hoped for, but at least they developed the system further...

Any experts in audience care to comment?


PROMISES HIGH-LEVEL
LANGUAGES AS “FUTURE WORK”
WRITES 4+ TOP-TIER
PUBS ON THEM

Good Guy MSR

Meanwhile, in Mountain View...

Meanwhile, in Mountain View...

Scumbag Google

Edit: actually, Pregel, Dremel, etc.

End