The case against specialized graph engines

Jing Fan, Adalbert Gerald Soosai Raj, and Jignesh M. Patel
University of Wisconsin – Madison

Motivation

- Graph analytics is now common
- Response = new specialized graph engines

"One Size Fits All": An Idea Whose Time Has Come and Gone

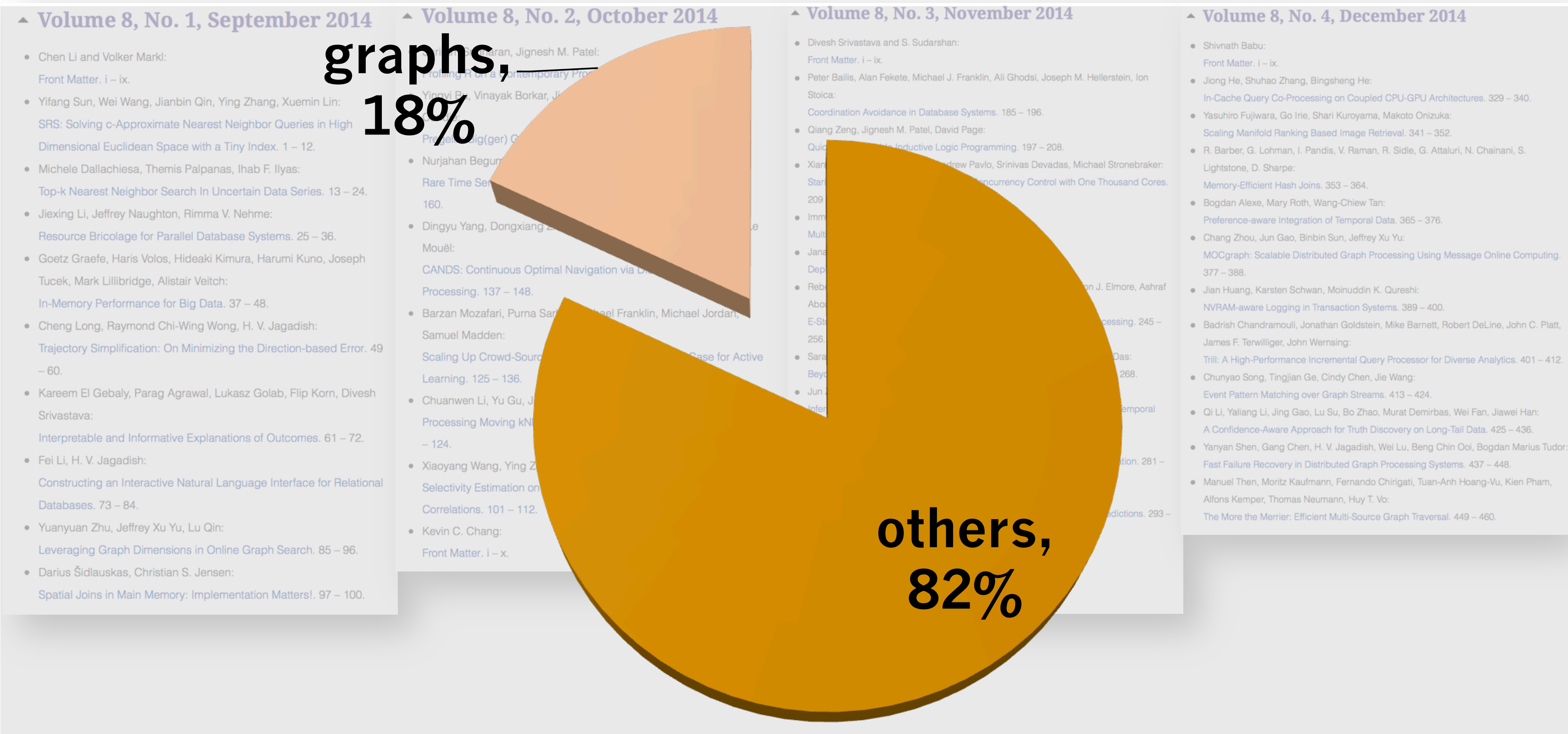
Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

ines causes va

Motivation

- Graph analytics is now common
- Response = new specialized graph engines



Motivation

- Graph analytics is now common
- Response = new specialized graph engines

Google Pregel

APACHE GIRAPH

Stanford GPS

GraphLab Unleash Data Science™

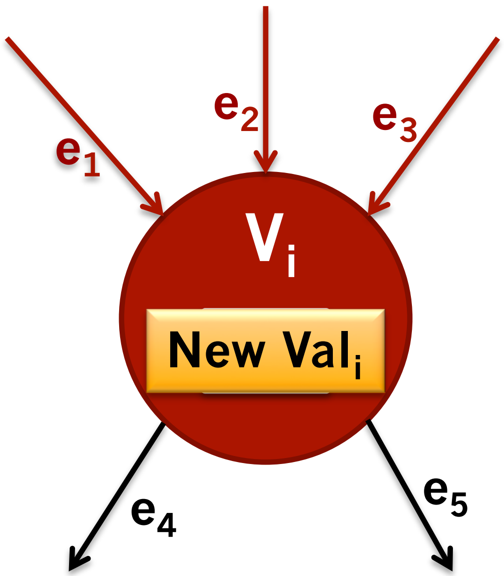
Question: Is graph processing that different from other types of data processing?

Our Answer: No. Can be subsumed by "traditional" relational processing

What is appealing about these new engines?

Vertex-centric API → Easy to write graph programs → Higher programmer productivity

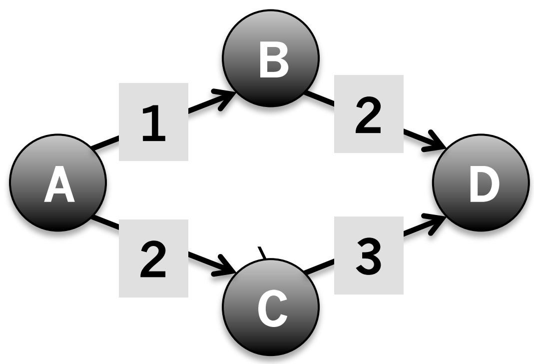
Graph API: Giraph



Vertex Centric:

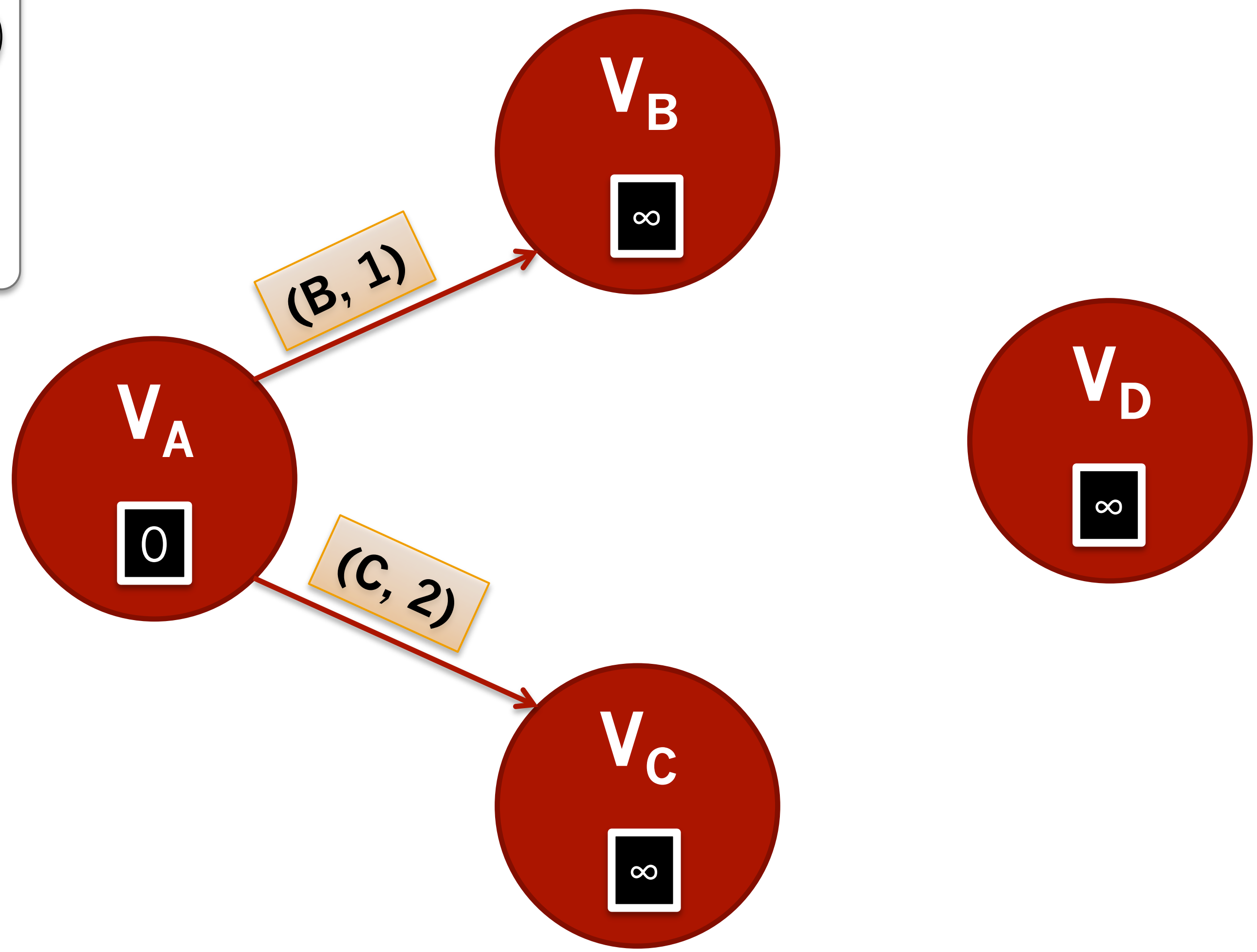
```
do {
  foreach vertex in the graph {
    receive_messages();
    mutate_vertex_value();
    if (send_to_neighbors()) {
       send_messages_to_neighbors();
    }
  }
} until (has_converged() || reached_limit())
```

Example: Shortest path



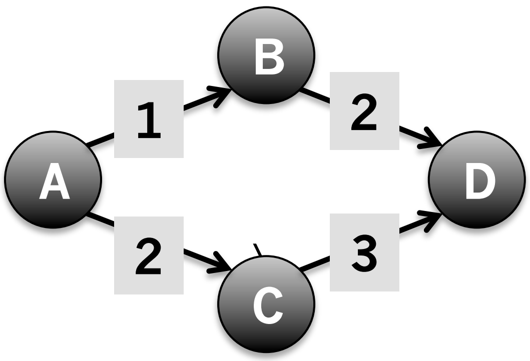
Input Graph

Computation & Communication Pattern



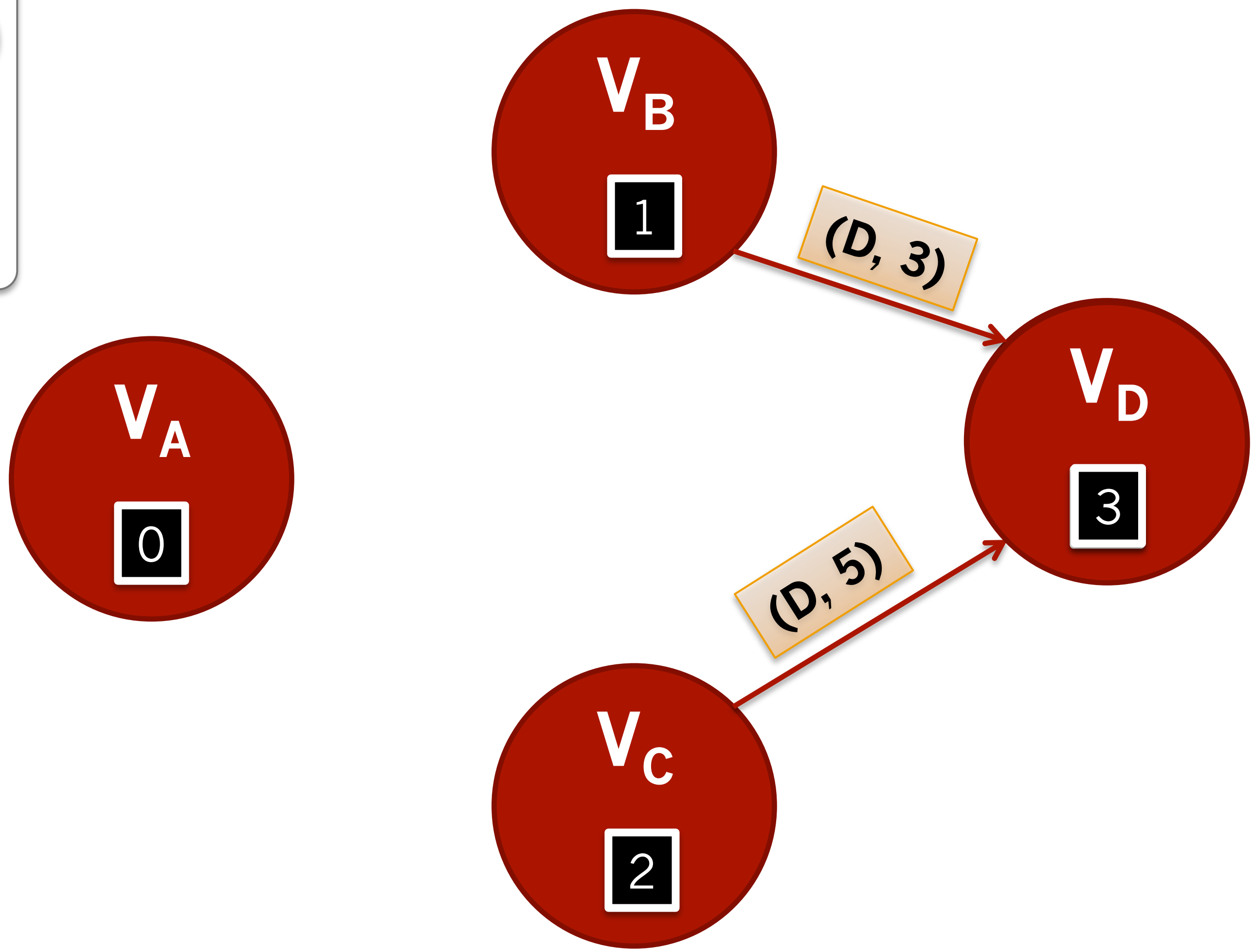
Iteration 1

Example: Shortest path



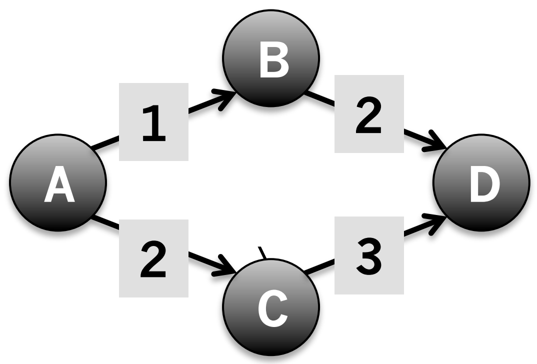
Input Graph

Computation & Communication Pattern



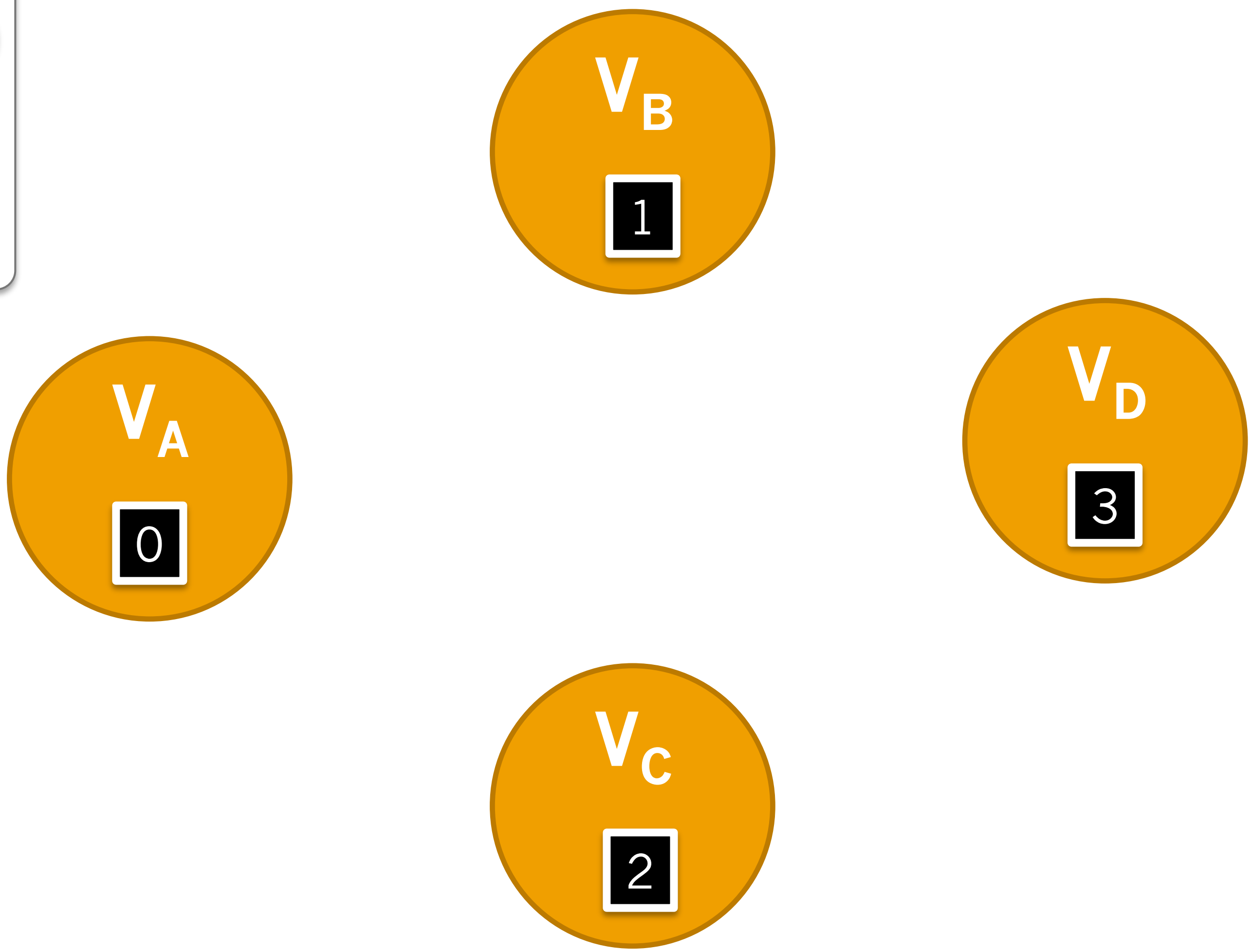
Iteration 2

Example: Shortest path



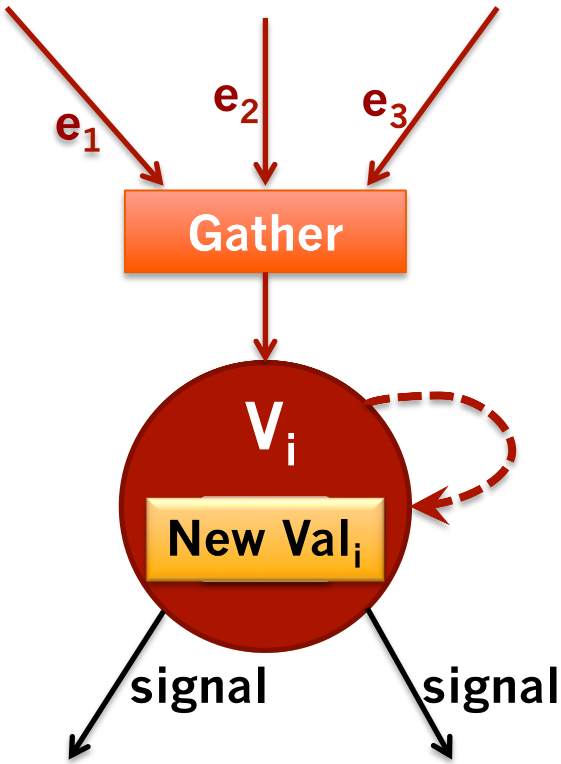
Input Graph

Computation & Communication Pattern



Iteration 3

GraphLab



1. **Gather** values (from neighbors)
2. **Apply** updates to local state
3. **Scatter** signals to your neighbors

What is appealing about these new engines?

Vertex-centric API → Easy to write graph programs → Higher programmer productivity

But …

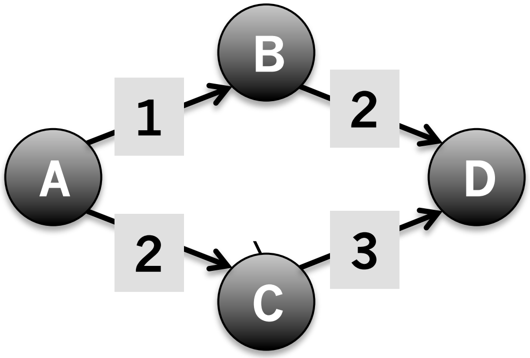
- Can we build a similar vertex-centric simple API?
- … and then map it to SQL, with good performance

The GRAIL API

Advantages:

- Already have SQL in the enterprise stack
- Huge advantage to "one size fits many"
  - $O(N^2)$ headaches when maintaining N specialized systems
  - Economies of scale

Example: Shortest path



Input Graph

Vertex

id	val
A	∞
B	∞
C	∞
D	∞

Edge

src	dest	val
A	B	1
A	C	2
B	D	2
C	D	3

Iteration 1

next

id	val
A	0
B	∞
C	∞
D	∞

message

id	val
B	1
C	2

Iteration 2

next

id	val
A	0
B	1
C	2
D	∞

message

id	val
D	3
D	5

Iteration 3

next

id	val
A	0
B	1
C	2
D	3

message

id	val

The Grail API

T-SQL Code

```
DECLARE @flag int;
SET @flag = 1;
SELECT vertex.id, 2147483647 AS val
INTO next
FROM vertex;
CREATE TABLE message(
   id int,
   val int
);
INSERT INTO message values(1,0);
WHILE (@flag != 0)
BEGIN
   SELECT message.id AS id, MIN(message.val) AS val
   INTO cur
   FROM message
   GROUP BY message.id;
   DROP TABLE message;
   SELECT cur.id AS id, cur.val AS val
   INTO update
   FROM cur, next
   WHERE cur.id = next.id AND cur.val <  next.val;
   UPDATE next
   SET next.val = update.val
   FROM next, update
   WHERE  next.id = update.id;
   SELECT edge.dest AS id, update.val + 1 AS val
   INTO message
   FROM update, edge
   WHERE edge.src = update.id;
   DROP TABLE cur;
   DROP TABLE update;
   SELECT @flag = COUNT(*) FROM message
END
```

Initialize

Initialize the message table

Aggregate the messages

Update and generate messages for the next iteration

Stop when no new msgs.

```
VertexValType: INT
MessageValType: INT
InitiateVal : INT_MAX
InitialMessage : (1, 0)
CombineMessage: MIN(message)
UpdateAndSend: update=cur.val<getVal()
        if (update) {
          setVal(cur.val)
          send(out, cur.val+1)
        }
End: NO_MESSAGE
```

T-SQL Code

```
DECLARE @flag int;
SET @flag = 1;

SELECT vertex.id, 2147483647 AS val
INTO next
FROM vertex;

CREATE TABLE message(
    id int,
    val int
);
INSERT INTO message values(1,0);

WHILE (@flag != 0)
BEGIN
    SELECT message.id AS id, MIN(message.val) AS val
    INTO cur
    FROM message
    GROUP BY message.id;

    DROP TABLE message;

    SELECT cur.id AS id, cur.val AS val
    INTO update
    FROM cur, next
    WHERE cur.id = next.id AND cur.val <  next.val;

    UPDATE next
    SET next.val = update.val
    FROM update, next
    WHERE  next.id = update.id;

    SELECT edge.dest AS id, update.val + 1 AS val
    INTO message
    FROM update, edge
    WHERE edge.src = update.id;

    DROP TABLE cur;

    DROP TABLE update;

    SELECT @flag = COUNT(*) FROM message;
END
```

Initialize

Initialize the message table

Aggregate the messages

Create an update table and only consider updated vertices

Update the next table

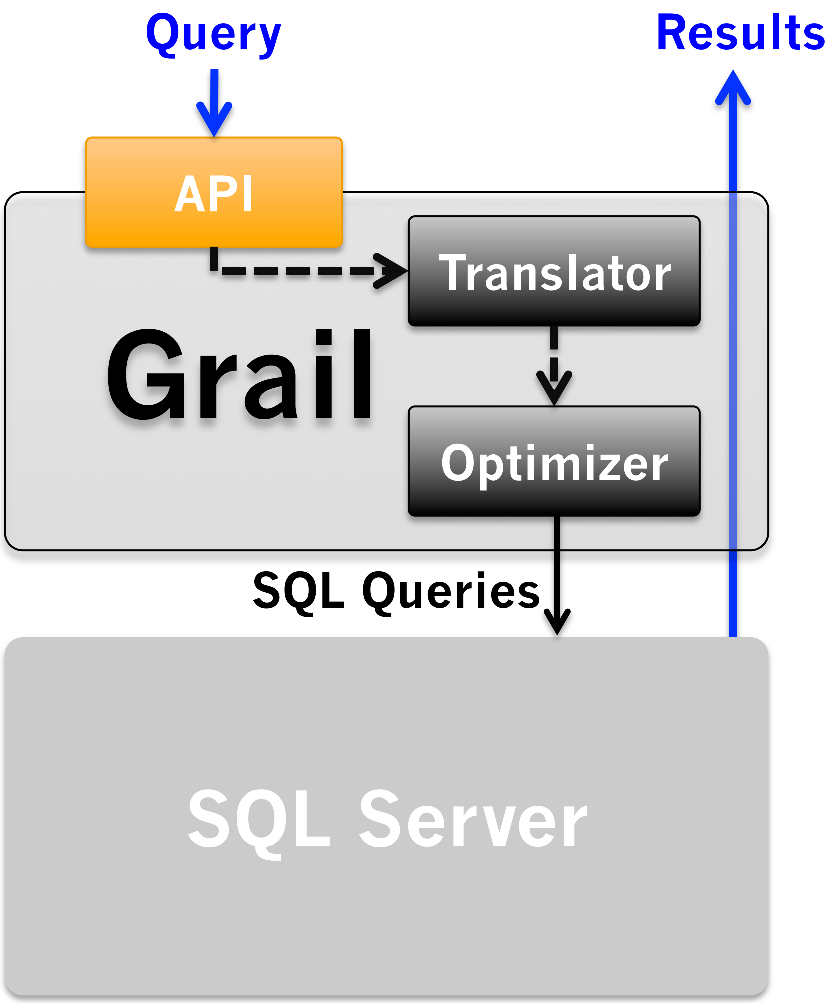
Generate the message table for the next iteration

Stop when there are no new messages

Vertex Centric	Relational Algebra
Receive messages	cur $\longleftarrow \gamma_{id,F_0(val)}(message)$
Mutate value	$next \xleftarrow{u} \pi_{next.id,F_1(other.val)} other \bowtie_{id} next$
Send messages	$\pi_{edge.B,F_2(other.val,edge.val)} other \bowtie_{other.id=edge.A} edge$

	Aggregate function (can be a UDAF)	Scalar computation (can be a UDF)	Scalar computation (can be a UDF)	Join attributes control the direction
For single source shortest path	min	sum	identity	Outgoing edges

Grail: Implementation and Evaluation



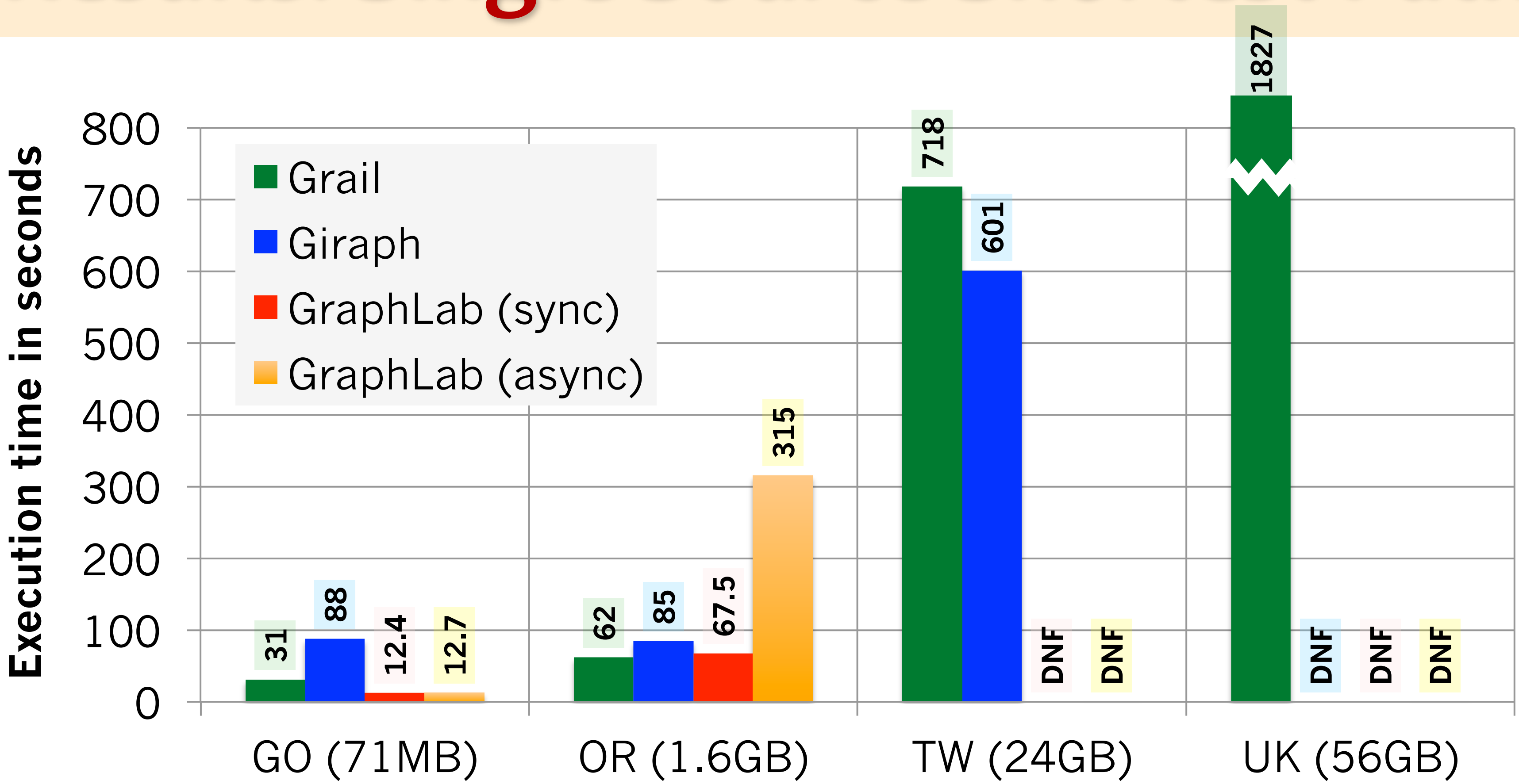
Test Machine (single node)
- Dual 1.8GHz Xeon E2450L
- 96GB of main memory

Compare with
- Giraph (v.1.1.0)
- GraphLab (v 2.2): sync and async

Dataset	#nodes	#edges	size
web-google (GO)	9K	5M	71MB
com-Orkut (OR)	3M	117M	1.6GB
Twitter-10 (TW)	41.6M	1.5B	24GB
uk-2007-05 (UK)	100M	3.3B	56GB

Queries
- Single source shortest-path
- Page Rank
- Weakly connect components

Results: Single Source Shortest Path



Grail is slower than GraphLab for the smallest datasets,
... but catches up as the dataset size grows,
... and can handle the largest datasets, while the other systems fail

Summary: Graph Analytics on RDBMS

Simple API (Grail) addresses the programmer productivity issue

Produces far more robust and deployable solutions than specialized graph engines

Interesting physical schema design and optimization issues

The general case against GraphDB Inc.

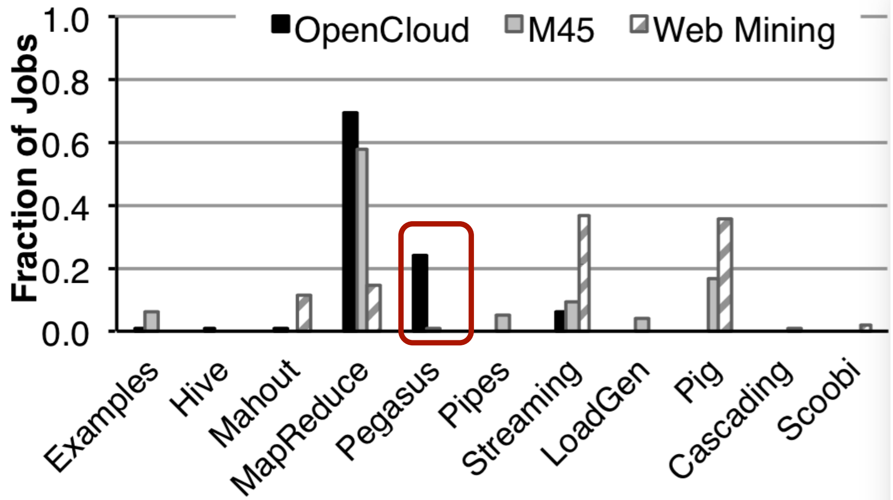


Figure 1: *Fraction of jobs per application type.*

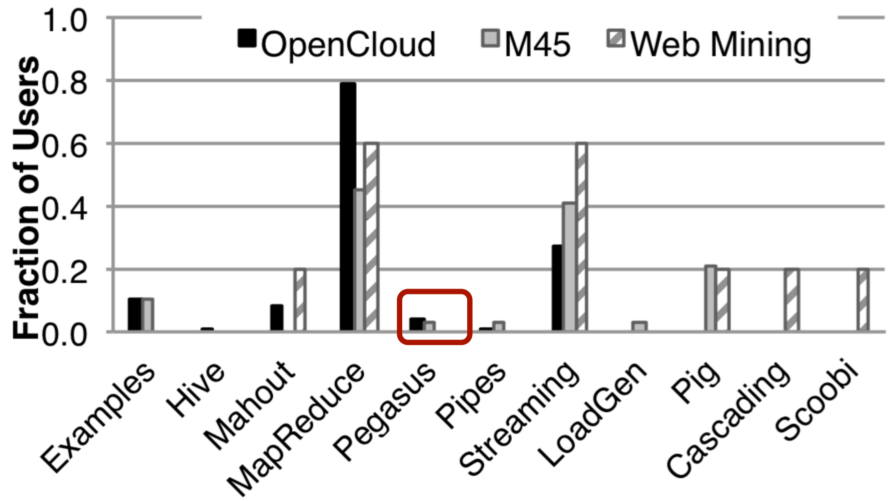


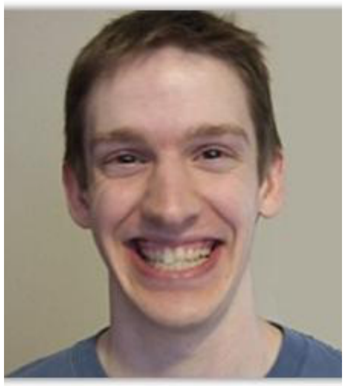
Figure 2: *Fraction of users per application type.*

Thanks!

David DeWitt

Jae Young Do

Alan Halverson



Ian Rae