TigerGraph's Computation Model

Alin Deutsch
Professor, UC San Diego
Chief Scientist, TigerGraph
Vertex types:
  • Product (name, category, price)
  • Customer (ssn, name, address)

Edge types:
  • Bought (discount, quantity)

Customer c bought 100 units of product p at discount 5%: modeled by edge

  (c) --[Bought {discount=5%, quantity=100}]--> (p)
Example: Customer Buys Product

- Customer
- Product
- Bought
- Name
- Discount
- Quantity
- Price
- Node
- Edge
- Property
Sales Data as Property Graph

- customers
- sales
- products
Map-Reduce Adapted to Graphs

• parallel processing

• computation starts from the "active vertex set"

• **Map** same computation over
  – active vertices, or over
  – edges incident to active vertex set
  and compute new active vertex set

• **Reduce** map results into aggregating containers called “accumulators”
Active Vertex Set
Map
Vertex Map

apply same computation to all active vertices
apply to all edges incident on active vertices
Compute New Active Vertex Set

apply same computation to all active vertices
Reduce

• The results of maps are aggregated by writing into containers called “accumulators”

"Edge-Map, Vertex-Reduce"

paradigm
Accumulators

• An Edge-Map-Vertex-Reduce step collects and aggregates data by writing it into *accumulators*

• Accumulators are containers (data types) that
  – hold a data value
  – accept inputs
  – aggregate inputs into the data value using a binary operation

• May be built-in (sum, max, min, etc.) or user-defined

• May be
  – global (a single container instance for the query)
  – vertex-attached (one container instance per vertex)
Vertex-Attached Accumulator Example: Revenue per Customer and per Product

@cSales

customer

bought

discount

quantity

thisSaleRevenue

price

@pSales
Reduce Into Vertex-Attached Accumulator: Revenue per Customer and per Product
Expressed in GSQL

• Edge Map maximizes opportunities for parallel evaluation

SumAccum<float> @cSales, @pSales;

SELECT  
c  
FROM    
(c:Customer) -[b:Bought]-> (p:Product)

ACCUM  
float thisSaleRevenue = b.quantity*(1-b.discount)*p.price,
    c.@cSales += thisSaleRevenue,
    p.@pSales += thisSaleRevenue;
Vertex-Attached Accumulator Example: Revenue per Customer and per Product

- Edge Map maximizes opportunities for parallel evaluation

```c
SumAccum<float> @cSales, @pSales;

Products =
SELECT p
FROM (c:Customer) -[b:Bought]-> (p:Product)
ACCUM float thisSaleRevenue = b.quantity*(1-b.discount)*p.price,
    c.@cSales += thisSaleRevenue,
    p.@pSales += thisSaleRevenue;
```

Benefits of Accumulator-based Aggregation
(Transcend Graph Model)

• It subsumes SQL-style aggregation
  – implemented SQL’s GROUP BY clause in GSQL as syntactic sugar

• Specifies queries whose evaluation is naturally parallelizable
  → performance!

• Facilitates specification of single-pass multi-aggregation (by different grouping criteria)
  – only partially supported even in SQL:
    – SQL’s most sophisticated aggregation primitives result in **wasteful aggregation** (may compute more aggregates than user needs)
    – Experiments show up to 3x speedup of accumulator-based over conventional (SQL-style) aggregation (see SIGMOD’20 paper)
Thank You