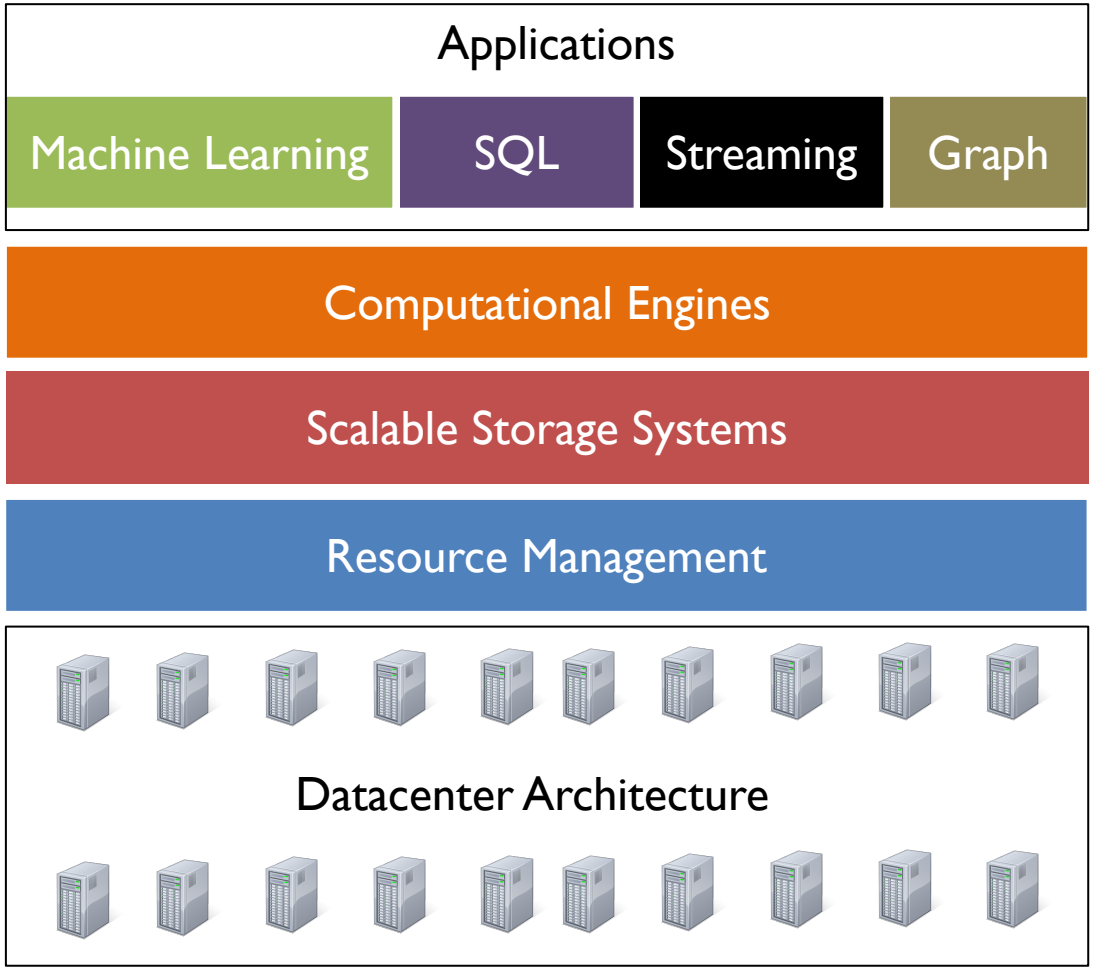
CS 744: GRAPHX

Shivaram Venkataraman

Fall 2021

ADMINISTRIVIA

- Midterm grades today?
- Course Project: Check in by Nov 30th


Applications
Machine Learning
SQL
Streaming
Graph
Computational Engines
Scalable Storage Systems
Resource Management
Datacenter Architecture

POWERGRAPH

What is different from dataflow system e.g., Spark?

Programming Model:
Gather-Apply-Scatter

Better Graph Partitioning
with vertex cuts

What are some shortcomings?

Distributed execution
(Sync, Async)

THIS CLASS

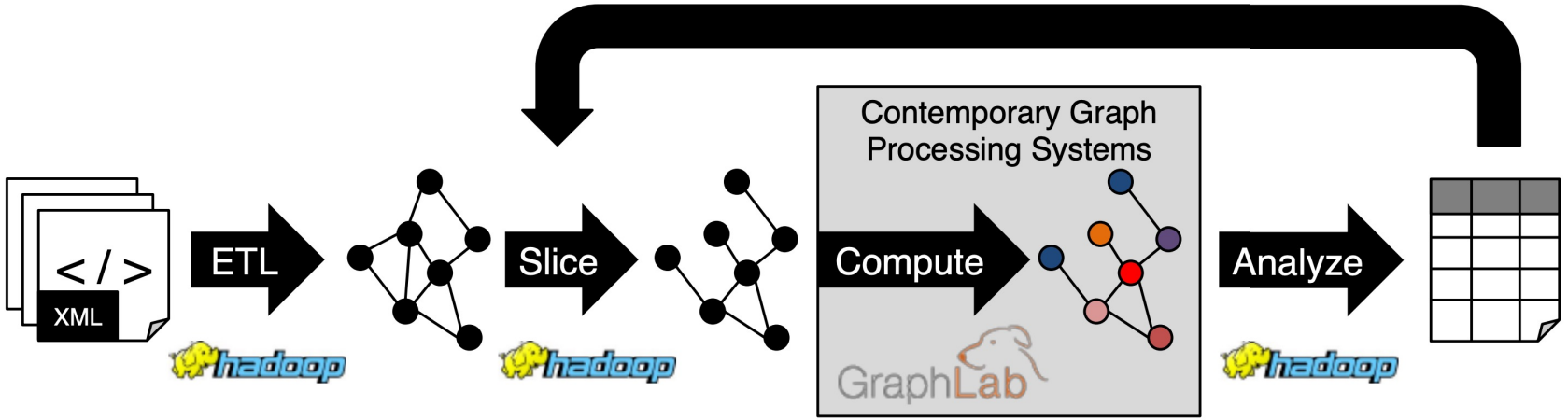
GraphX

Can we efficiently map graph abstractions to dataflow engines?

Scalability! But at what COST?

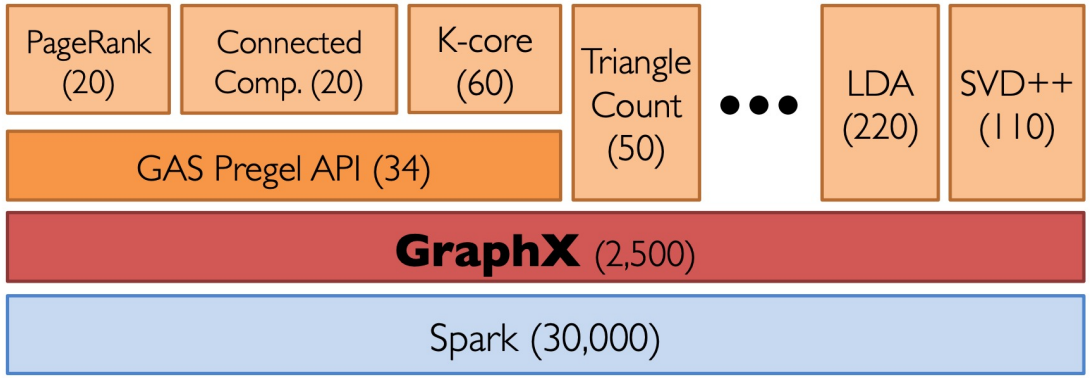
When should we distribute graph processing?

MOTIVATION


XML
ETL
hadoop
Slice
hadoop
Contemporary Graph
Processing Systems
Compute
GraphLab
Analyze
hadoop

SYSTEM OVERVIEW

Advantages?



PROGRAMMING MODEL

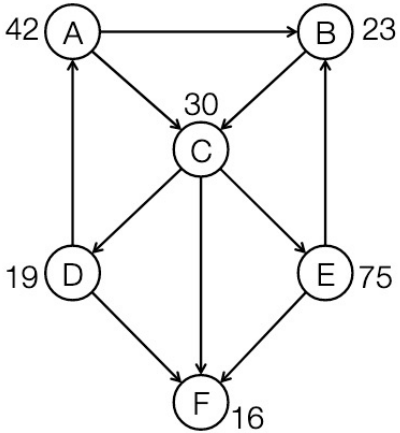
```
class Graph[V, E] {
  // Constructor
  def Graph(v: Collection[(Id, V)],
            e: Collection[(Id, Id, E)])
  // Collection views
  def vertices: Collection[(Id, V)]
  def edges: Collection[(Id, Id, E)]
  def triplets: Collection[Triplet]
  // Graph-parallel computation
  def mrTriplets(f: (Triplet) => M,
      sum: (M, M) => M): Collection[(Id, M)]
  // Convenience functions
  def mapV(f: (Id, V) => V): Graph[V, E]
  def mapE(f: (Id, Id, E) => E): Graph[V, E]
  def leftJoinV(v: Collection[(Id, V)],
      f: (Id, V, V) => V): Graph[V, E]
  def leftJoinE(e: Collection[(Id, Id, E)],
      f: (Id, Id, E, E) => E): Graph[V, E]
  def subgraph(vPred: (Id, V) => Boolean,
      ePred: (Triplet) => Boolean)
    : Graph[V, E]
  def reverse: Graph[V, E]
}
```

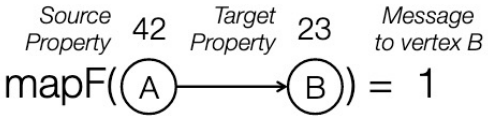
Constructor

Triplets

MR TRIPLETS

```
mrTriplets(f: (Triplet) => M, sum: (M, M) => M): Collection[(Id, M)]
```





Resulting Vertices

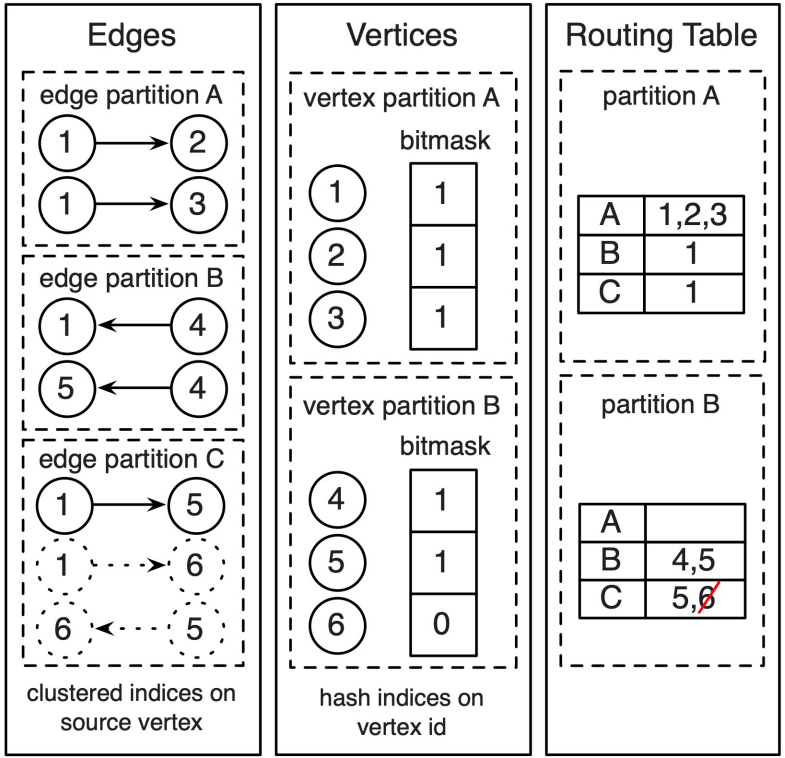
Vertex Id	Property
A	0
B	2
C	1
D	1
E	0
F	3

```
val graph: Graph[User, Double]
def mapUDF(t: Triplet[User, Double]) =
  if (t.src.age > t.dst.age) 1 else 0
def reduceUDF(a: Int, b: Int): Int = a + b
val seniors: Collection[(Id, Int)] =
  graph.mrTriplets(mapUDF, reduceUDF)
```

PREGEL USING GRAPHX

```
def Pregel(g: Graph[V, E],
      vprog: (Id, V, M) => V,
      sendMsg: (Triplet) => M,
      gather: (M, M) => M): = {

  g.mapV((id, v) => (v, halt=false))

  while (g.vertices.exists(v => !v.halt)) {
    val msgs: Collection[(Id, M)] =
        g.subgraph(ePred=(s,d,sP,eP,dP)=>!sP.halt)
         .mrTriplets(sendMsg, gather)

    g = g.leftJoinV(msgs).mapV(vprog)
  }

  return g.vertices
}
```

IMPLEMENTING TRIPLETS VIEW



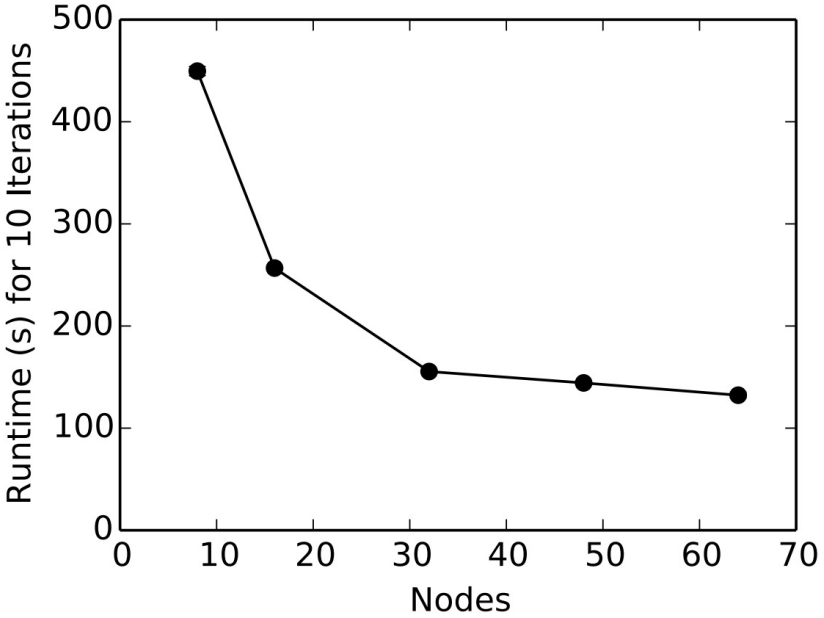
Join strategy

Send vertices to the edge site

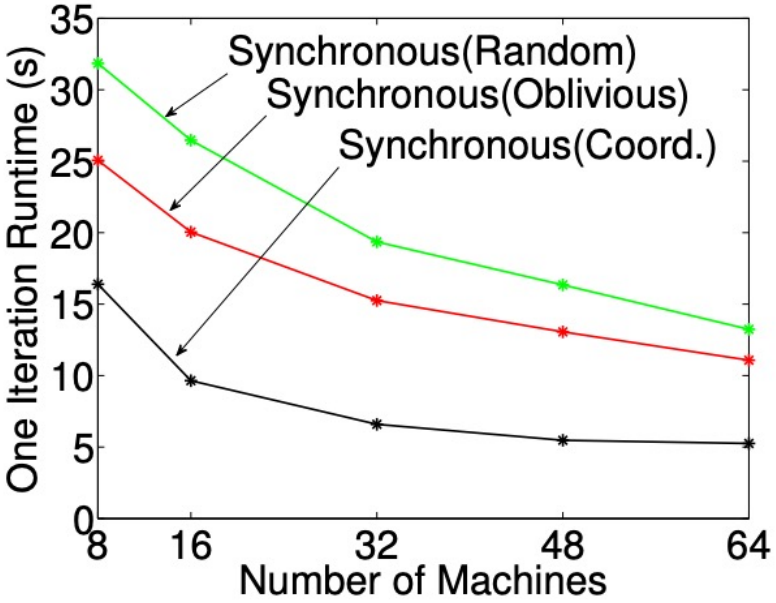
Multicast join

Using routing table

SCALABILITY VS. ABSOLUTE PERFORMANCE



GraphX
3x from 8 to 32 machines



PowerGraph
2.6x from 8 to 32

COST: CONFIGURATION THAT OUT-PERFORMS SINGLE THREAD

```
fn PageRank20(graph: GraphIterator, alpha: f32) {
  let mut a = vec![0f32; graph.nodes()];
  let mut b = vec![0f32; graph.nodes()];
  let mut d = vec![0f32; graph.nodes()];

  graph.map_edges(|x, y| { d[x] += 1; });

  for iter in 0..20 {
    for i in 0..graph.nodes() {
      b[i] = alpha * a[i] / d[i];
      a[i] = 1f32 - alpha;
    }

    graph.map_edges(|x, y| { a[y] += b[x]; });
  }
}
```

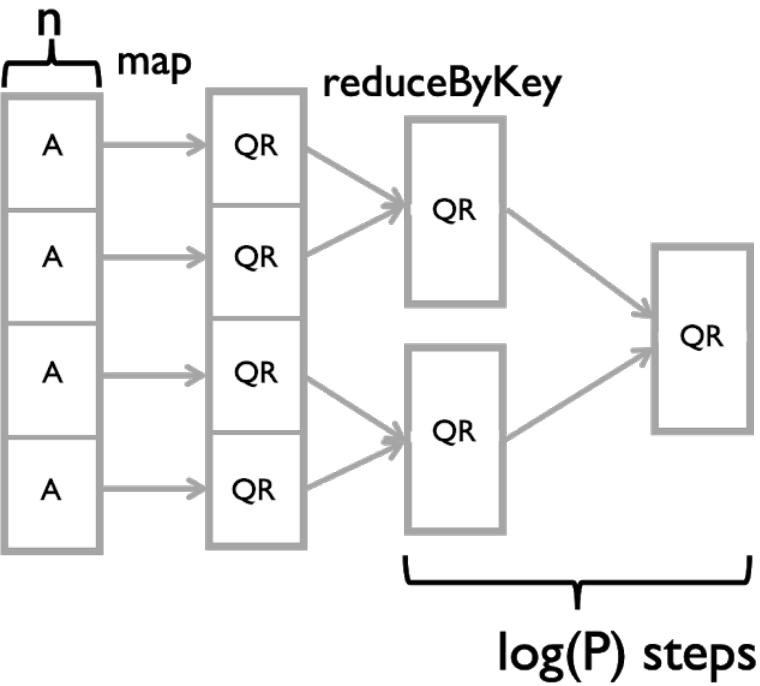
scalable system	cores	twitter
GraphLab [10]	128	249s
GraphX [10]	128	419s
Single thread (SSD)	1	300s
Single thread (RAM)	1	275s

DISCUSSION

https://forms.gle/u4TvMumnH7yBHd3b8

What are some reasons why GraphX or GraphLab or Naiad might be slower than a single thread implementation of PageRank?

How would you expect a single-thread QR implementation to perform?



Configuration: 100K rows/core, 24 cores

Matrix: 2.4M x 1024, Dense matrix

	1st Stage	TSQR tree	Total
EC2+OpenBLAS	23.604 (s)	1.080 (s)	24.752 (s)

SUMMARY

GraphX: Combine graph processing with relational model

COST

- Configuration that outperforms single-thread
- Measure scalability AND absolute performance
    - Computation model of scalable frameworks might be limited
    - Hardware efficiency matters
    - System/Language overheads

NEXT STEPS

Next class: Marius

Project check-ins by Nov 20th

OPTIMIZING MR TRIPLETS

Filtered Index Scanning

- Store edges clustered on source vertex id
- Filter triplets using user-defined predicate

Automatic Join Elimination

- Some UDFs don't access source or dest properties
- Inspect JVM byte code to avoid joins