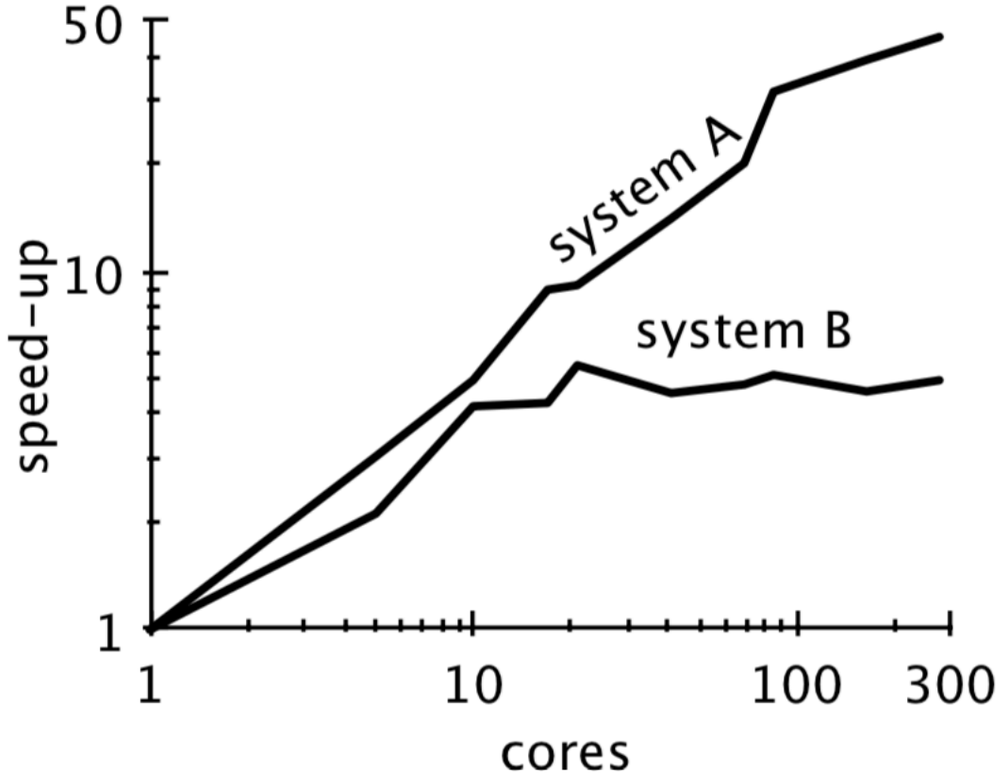
Scalability! But at what COST?

Abhinav Garg
CS 744 - Fall 2018

Outline

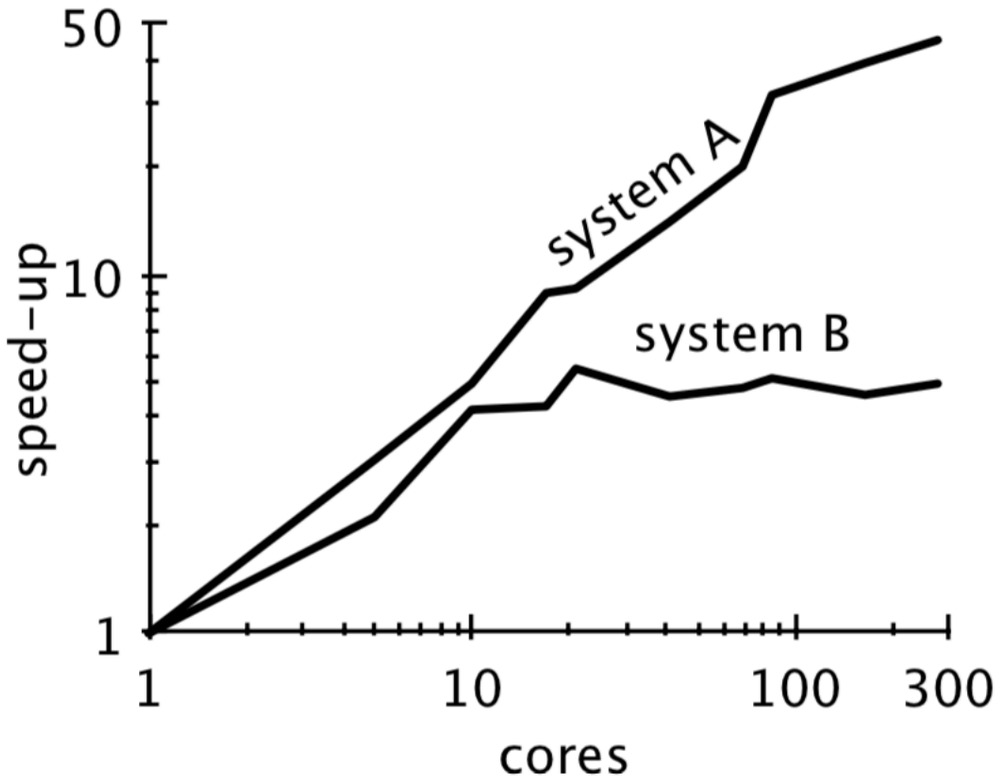
- Motivation
- Goal
- COST
- Methodology
- Baseline Measurements
- Better Baselines
- Applying COST to prior work
- Take-aways

Which system is better ?

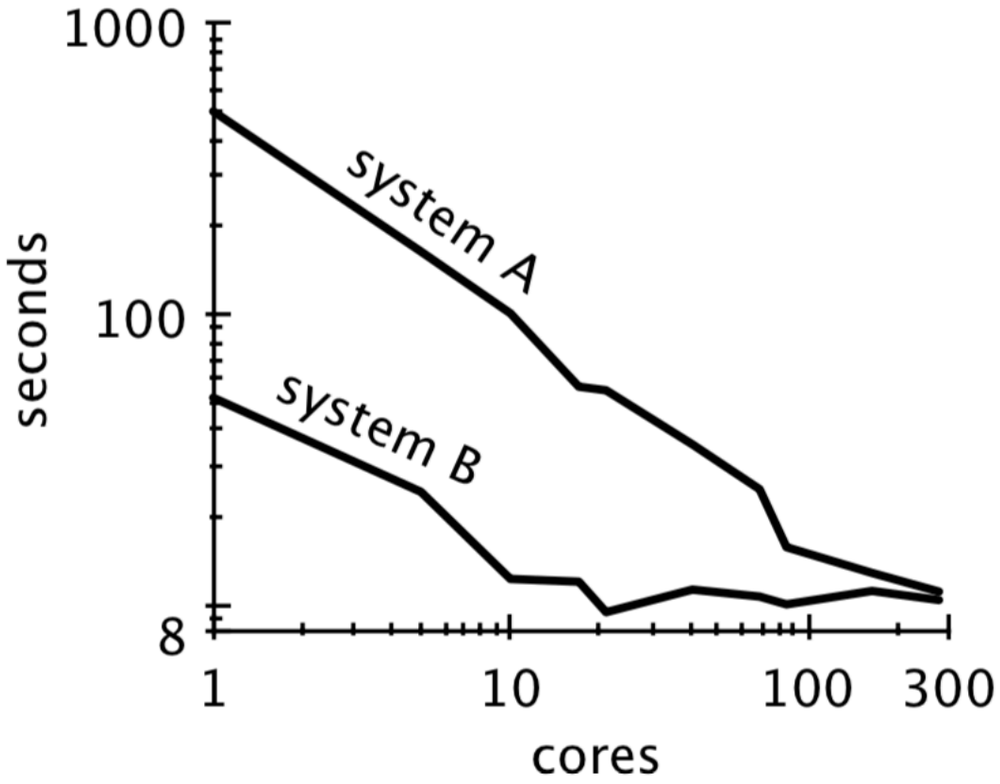


Scaling of System A and System B

Which one would you use ?



Scaling



Performance

Naiad computation before (System A) and after (System B) a performance optimization is applied

Motivation

- Scalability is considered most important feature
- Big data systems may scale well, often because they introduce a lot of overhead
- Are systems truly improving performance?

Goal

- A new performance metric for big data platforms
- Distinguish scalability from efficient use of resources
- Weight system’s scalability against overheads
- Do not reward systems with substantial but parallelizable overheads

COST

- Configuration that outperforms a single thread
- Hardware configuration required before platform outperforms competent single threaded implementation

Methodology

- Take measurements from recent graph processing publications
- Compare against simple single-threaded implementations running on a laptop
- Write competent, but not overly fancy algorithms.
- Evaluate Page Rank and Graph Connectivity on twitter_rv and uk_2007_05 graphs (GraphX)

Baseline Measurements

scalable system	cores	twitter	uk-2007-05
GraphChi [12]	2	3160s	6972s
Stratosphere [8]	16	2250s	-
X-Stream [21]	16	1488s	-
Spark [10]	128	857s	1759s
Giraph [10]	128	596s	1235s
GraphLab [10]	128	249s	833s
GraphX [10]	128	419s	462s
Single thread (SSD)	1	300s	651s
Single thread (RAM)	1	275s	-

Elapsed time for 20 Page Rank iterations

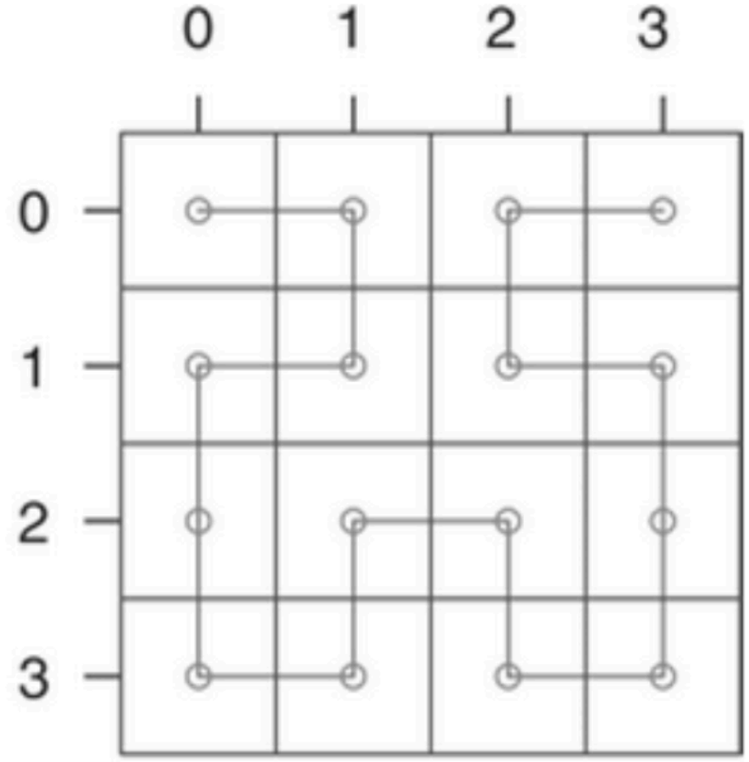
Baseline Measurements

scalable system	cores	twitter	uk-2007-05
Stratosphere [8]	16	950s	-
X-Stream [21]	16	1159s	-
Spark [10]	128	1784s	$\geq$ 8000s
Giraph [10]	128	200s	$\geq$ 8000s
GraphLab [10]	128	242s	714s
GraphX [10]	128	251s	800s
Single thread (SSD)	1	153s	417s

Elapsed time for Graph Connectivity (using label propagation)

Better Baselines

- Improve graph layout
- Hilbert Order instead of Vertex Order
- (good, good) locality instead of (great, poor)
- Reduces TLB misses and page walks



Better Baselines

- Improve algorithms
- Label propagation scales due to algorithms sub-optimality
- Label propagation does more work than better algorithms
- Use Union-Find algorithm

```rust
fn label_propagation<G: EdgeMapper>(graph: &G, nodes: u32) {
    let mut label: Vec<u32> = (0..nodes).collect();
    let mut done = false;
    while !done {
        done = true;
        graph.map_edges(|x, y| {
            if label[x] != label[y] {
                done = false;
                label[x] = min(label[x], label[y]);
                label[y] = min(label[x], label[y]);
            }
        });
    }
}
fn union_find<G: EdgeMapper>(graph: &G, nodes: u32) {
    let mut root: Vec<u32> = (0..nodes).collect();
    let mut rank: Vec<u8> = (0..nodes).map(|_| 0u8).collect();
    graph.map_edges(|mut x, mut y| {
        while x != root[x] { x = root[x]; }
        while y != root[y] { y = root[y]; }
        if x != y {
            match rank[x].cmp(&rank[y]) {
                Less    => root[x] = y,
                Greater => root[y] = x,
                Equal   => { root[y] = x; rank[x] += 1 },
            }
        }
    });
}
```

Better Baselines

Page Rank

scalable system	cores	twitter	uk-2007-05
GraphLab	128	249s	833s
GraphX	128	419s	462s
Vertex order (SSD)	1	300s	651s
Vertex order (RAM)	1	275s	-
Hilbert order (SSD)	1	242s	256s
Hilbert order (RAM)	1	110s	-

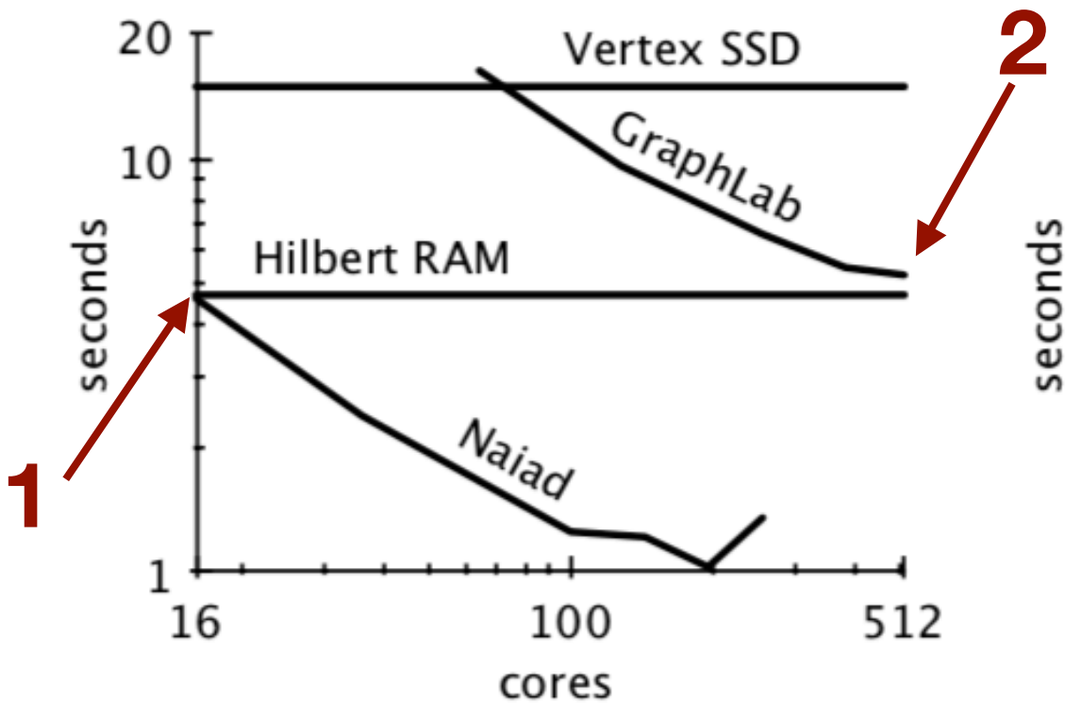
179 sec to convert →

Graph Connectivity

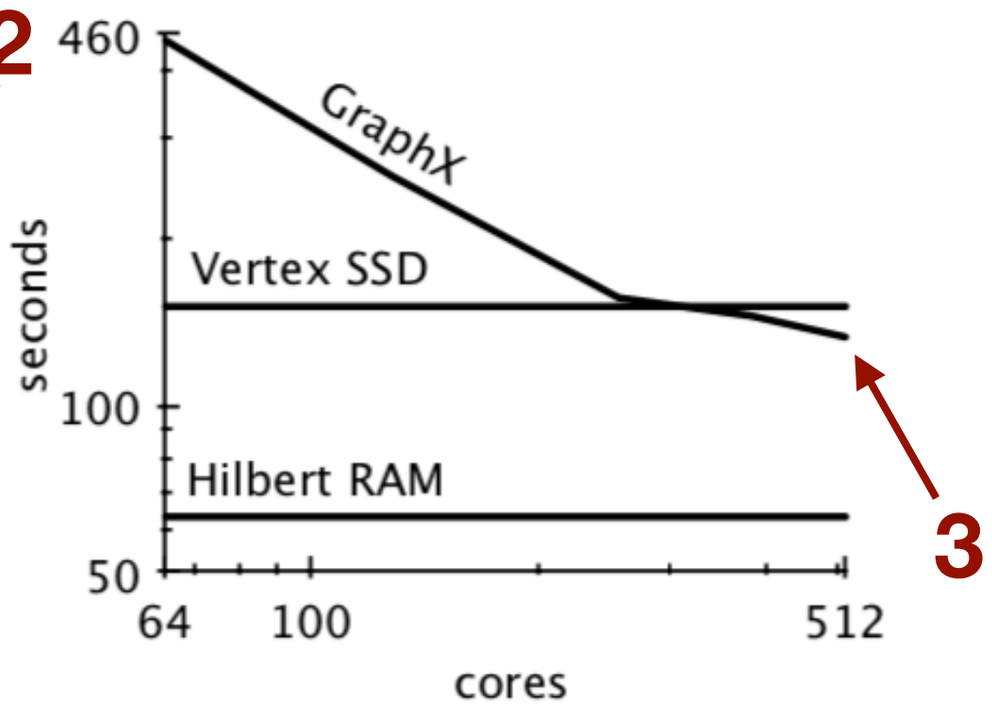
scalable system	cores	twitter	uk-2007-05
GraphLab	128	242s	714s
GraphX	128	251s	800s
Single thread (SSD)	1	153s	417s
Union-Find (SSD)	1	15s	30s

Does not ‘think like a vertex’, but parallelizable →

Applying COST to prior work



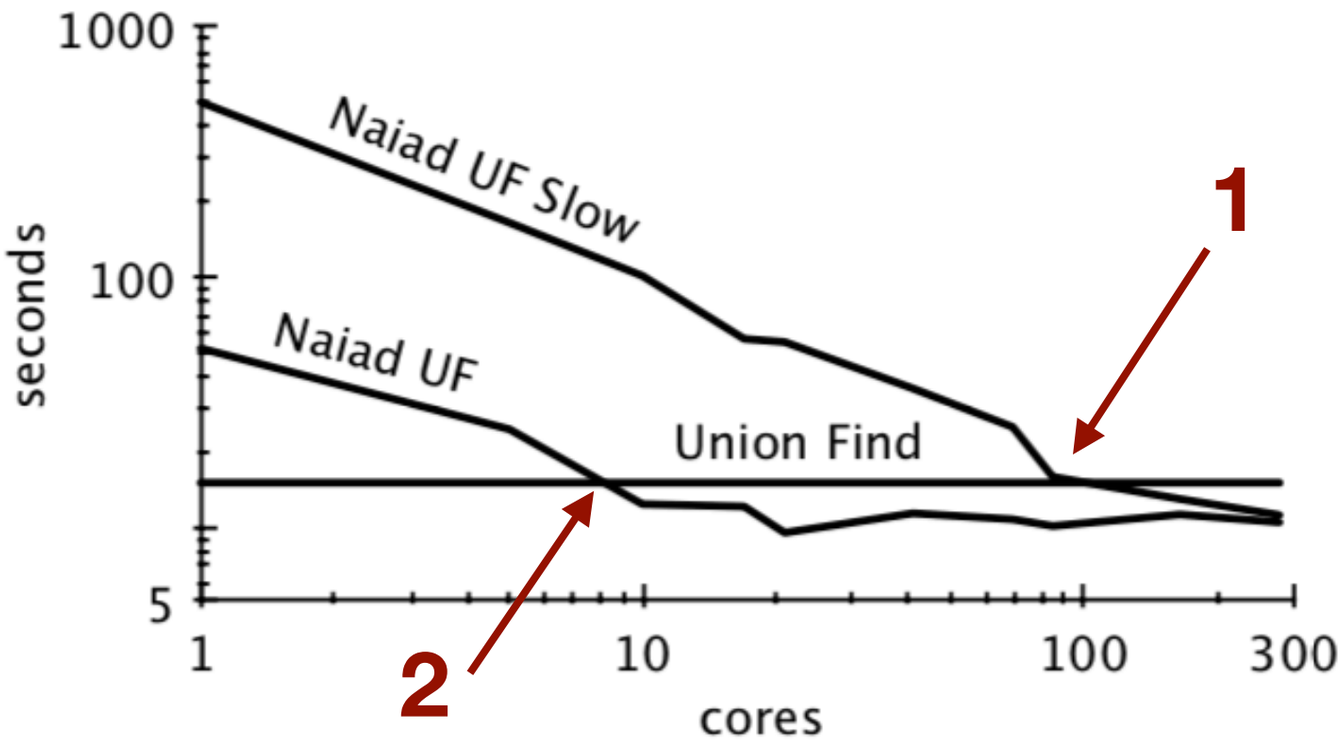
Time per warm iteration



Time for 10 iterations from a cold start

Scaling measurements for Page Rank on Twitter Graph

Applying COST to prior work



Two Naiad implementations of parallel union-find for graph connectivity

- 1- Hash Table based
- 2- Array based
- Makes trade-off clearer

Reasons to tolerate high COST

- Integration with existing ecosystem
- Target variety of problems
- High availability, fault tolerance, or security
- Technical expertise of the team

Think: Do you really need the high COST system?

Take-aways

- Understanding overheads is important
- Most scalable systems might not be most efficient
- Consider alternative hardware and algorithms
- Important to evaluate COST - to explain if high COST is intrinsic, to highlight avoidable inefficiencies

Questions ?

References

- Frank McSherry, Michael Isard, Derek Murray. *Scalability! But at what COST?* HotOS, 2015

- http://www.frankmcsherry.org/graph/scalability/cost/2015/01/15/COST.html

- https://www.youtube.com/watch?v=6bWBEJBMNG0