OPTiC: Opportunistic Graph Processing in Multi-Tenant Clusters

Muntasir Raihan Rahman, Nokia Bell Labs

Indranil Gupta, University of Illinois Urbana-Champaign

Akash Kapoor, Princeton University

Haozhen Ding, Airbnb

Distributed Protocols Research Group (DPRG)

http://dprg.cs.uiuc.edu/

OPTiC: Opportunistic graph Processing on Multi-Tenant Clusters

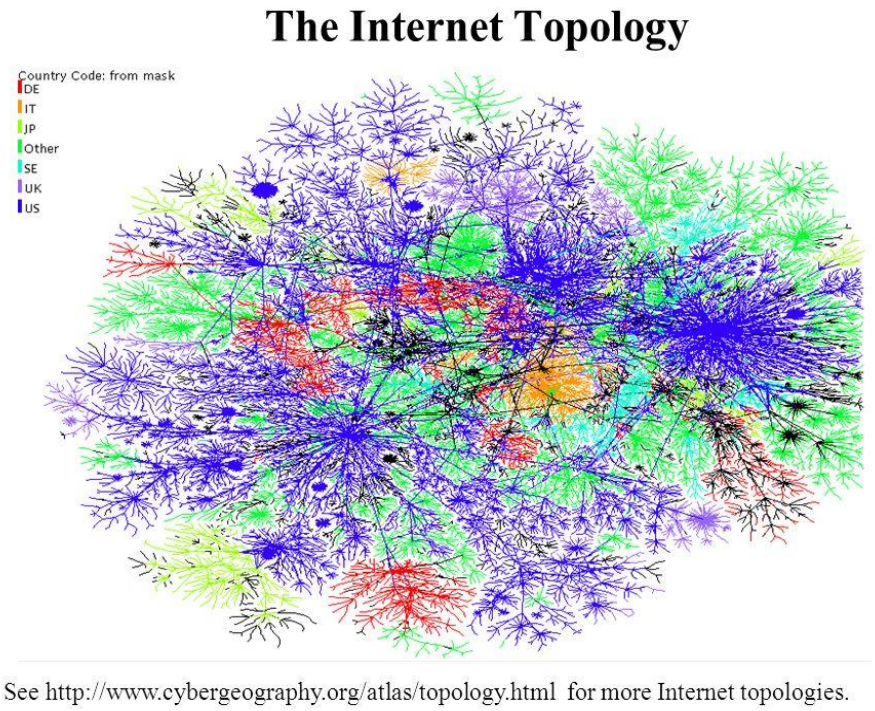
- OPTiC is the first multi-tenant system for graph processing
- OPTiC bridges the gap between graph processing layer and cluster scheduler layer
- Key techniques
  - New algorithm for graph computation progress estimation
  - Smart prefetching of resources
- We implemented our system on top of Apache Giraph + YARN stack
- We obtain 20-82% improvement in job completion time for realistic workloads under realistic network conditions

Graphs are Ubiquitous

Biological
- Food Web
- Protein Interaction Network
- Metabolic Network

Man-made
- Online Social Network (OSN)
- Web Graph
- The Internet

Protein Interaction Network



The Internet Graph

Graphs are Massive Scale: Facebook Graph: |V|=1.1B, |E|=150B (May 2013)

Distributed Graph Processing

Apache Giraph

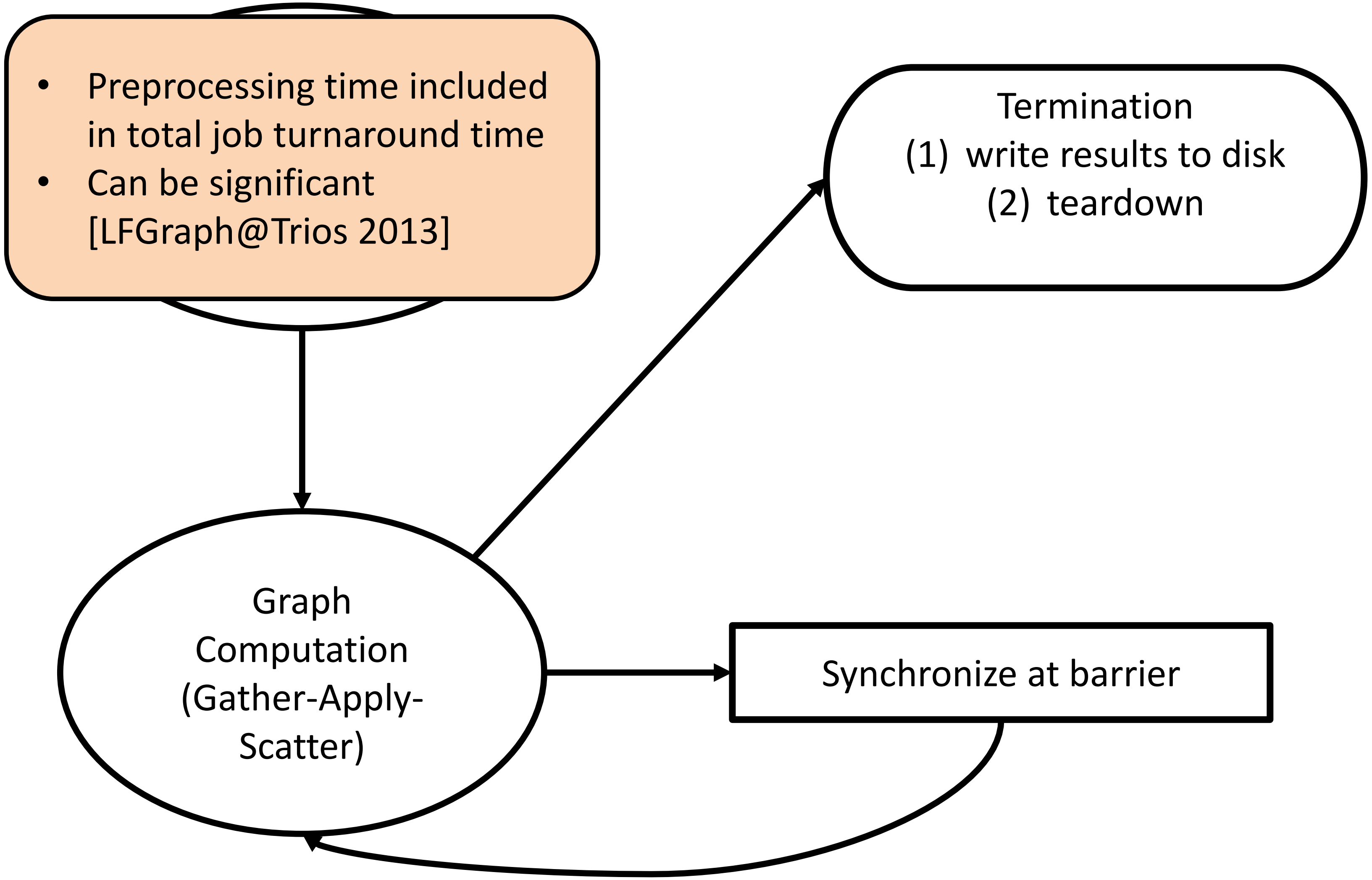
Dato PowerGraph

PowerLyra

Databricks GraphX

Google Pregel

Anatomy of a Graph Processing Job



Graph Processing on Multi-tenant Clusters

Graph Processing Engines do not take advantage of multi-tenancy in cluster scheduler

GAP

Cluster Schedulers un-aware of graph nature of jobs

- Only assume map-reduce or similar abstractions

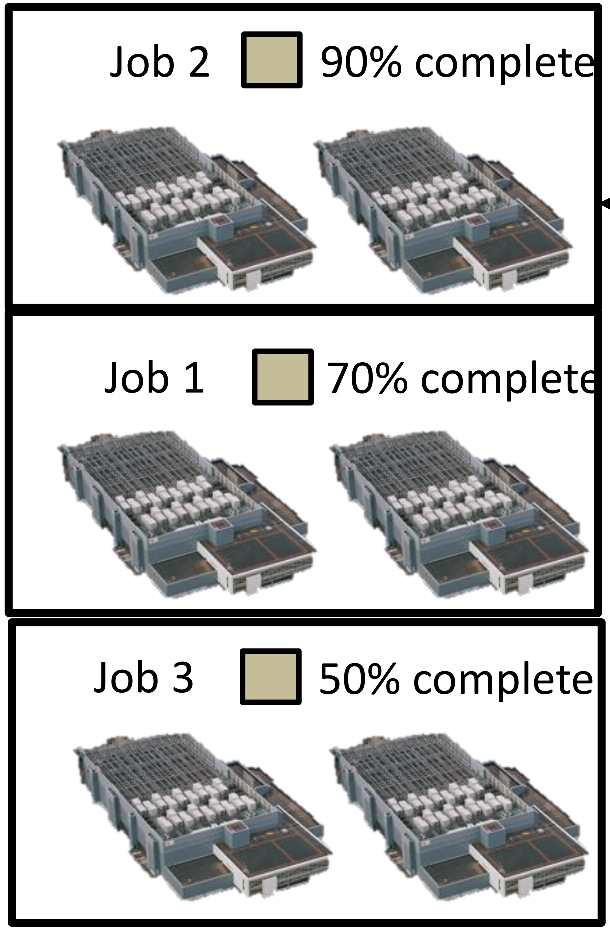
OPTiC: Opportunistic Graph Processing on Multi-Tenant Clusters

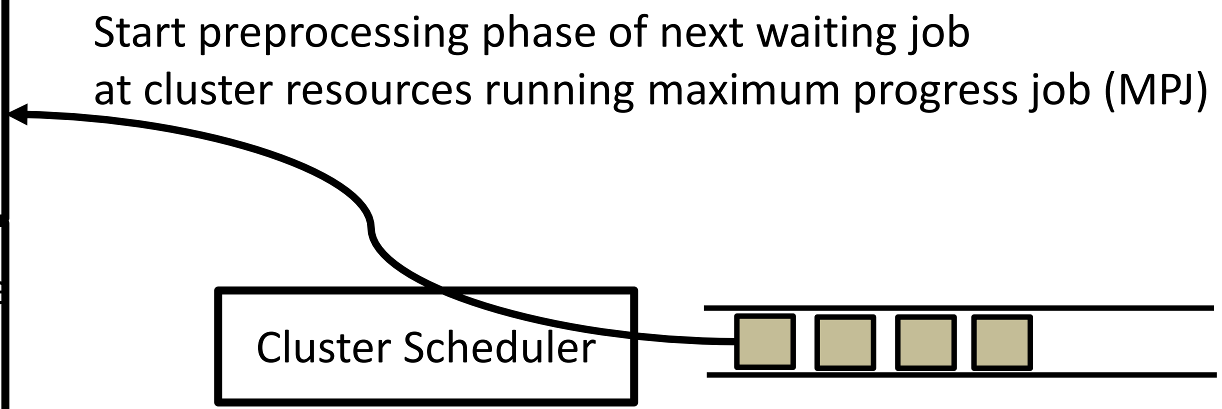
Key Idea: Opportunistic Overlapping of
(1) Graph Preprocessing Phase of Waiting Jobs with
(2) Graph Computation Phase of Current Jobs

System Assumptions

- Synchronous graph processing (workers sync periodically)
- Over-subscribed cluster (always a waiting job)
- No pre-emption
- All input graphs stored in Distributed File System (e.g., HDFS)
- Disk locality matters

Key Idea, Simplified: Opportunistic Overlapping





Benefits:

- MPJ most likely to free up cluster resources first
- When the next waiting job is scheduled, preprocessing phase is already underway

Challenges

1# Prefetching Resources
2# Estimating Progress

Challenge # 1: How to Prefetch

Desired Feature: Minimal Interference on Current Running Jobs

Progress-Aware Memory Prefetching

- Prefetch graph of waiting job directly into memory of MPJ server(s)
- MPJ server memory being used to store and compute on MPJ graph
- Interferes with MPJ, potentially increase MPJ run-time

Progress-Aware Disk Prefetching (PADP)

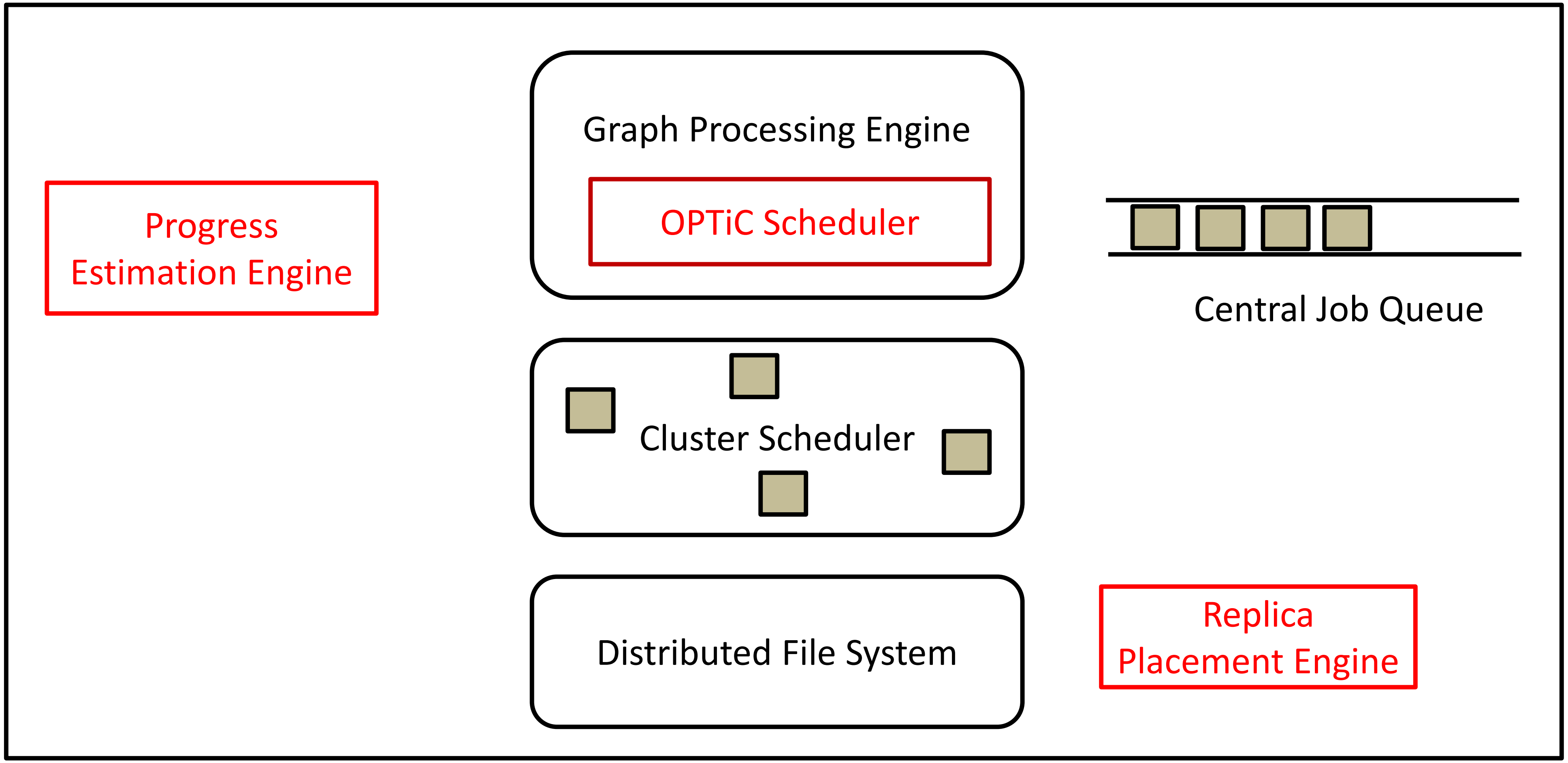
- Prefetch graph of waiting job into disk of MPJ server(s)

✓

- Local disk fetch avoids network contention
- DARE@IEEE Cluster data (Amazon 20 server virtual cluster)
  - Amazon EC2 disk bandwidth mean 141.5 MB/s
  - Amazon EC2 network bandwidth mean 73.2 MB/s
- Cheaper to fetch from local disk than from network

MPJ=Max Progress Job

Architecture: OPTiC with PADP



OPTiC-PADP Scheduling Algorithm

OPTiC scheduler
- For next waiting job in queue
  - Fetch progress information of running jobs
  - Determine server(s) ***S***

Running Job

1. Creating additional replicas in disk increases the (non-zero) storage performance cost
2. But there is a lot of available space on disks, which are mostly under-utilized
3. So the actual dollar cost of the system is close to zero

Cluster Scheduler
- Scheduled next waiting job when ***MPJ*** finishes

Next Waiting Job
- Scheduled on ***S***
- Fetch graph from local disk instead of remote disk in ***DFS***

Challenge # 2: Estimating Progress of Graph Computation

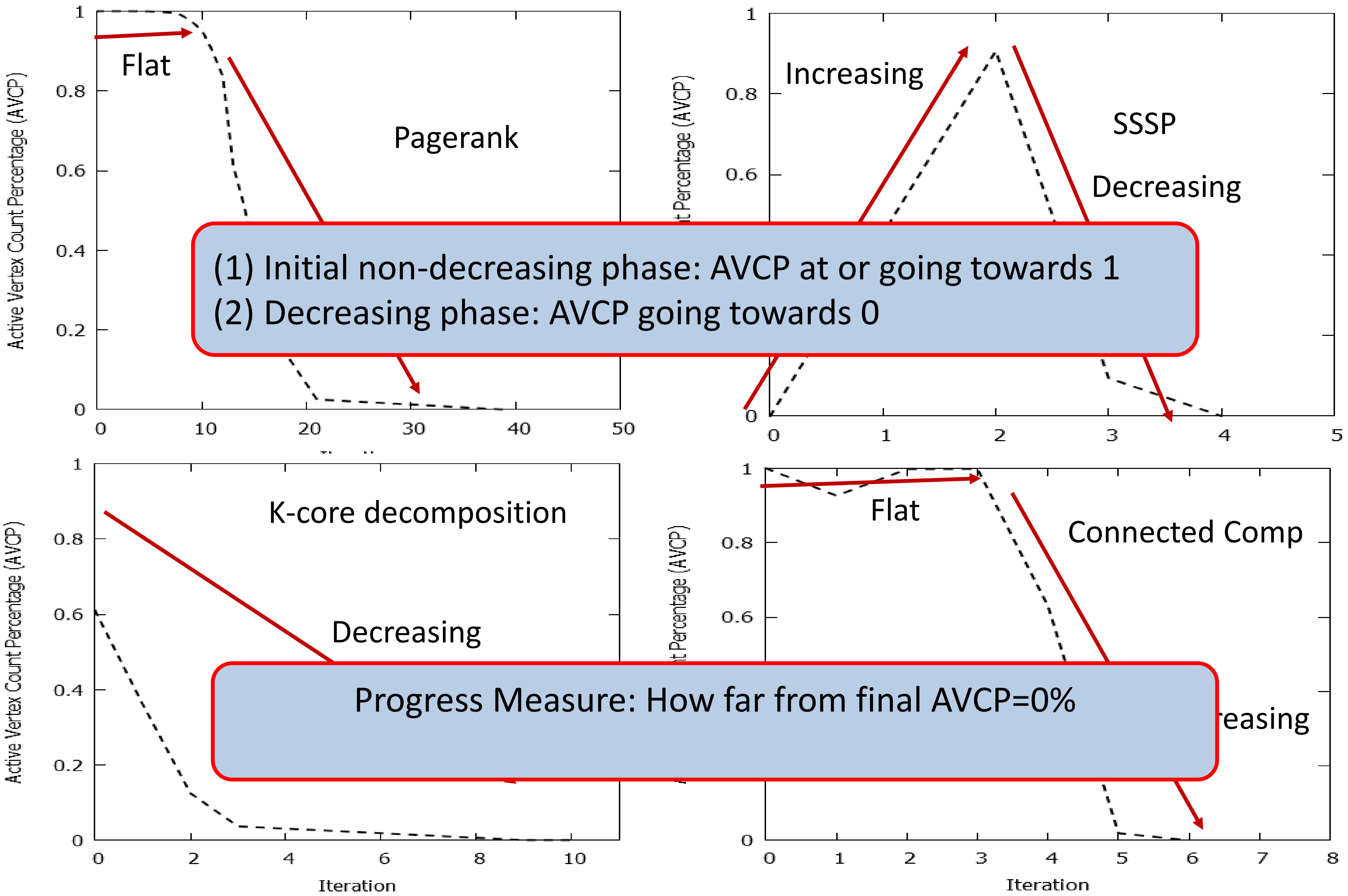
1. Profiling:
   – Profile the run-time of various graph algorithms on different cluster configurations for different graph sizes
   – Huge overhead, job details dependent (-)
2. Use Cluster Scheduler Progress Estimator:
   – For example Giraph programs are mapped to map-reduce programs
   – Use cluster map-reduce progress estimator to estimate graph computation progress
   – Cluster dependent (-)

Profile-free, Cluster-agnostic Progress Estimation

Use Graph Processing Layer Metrics:

- Track the evolution of active vertex count (AVC)
  - A vertex is active as long as there are some incoming messages from previous iteration
- At termination AVC = 0
- Profile-independent, Cluster-agnostic (+)

Evolution of AVCP=AVC/N

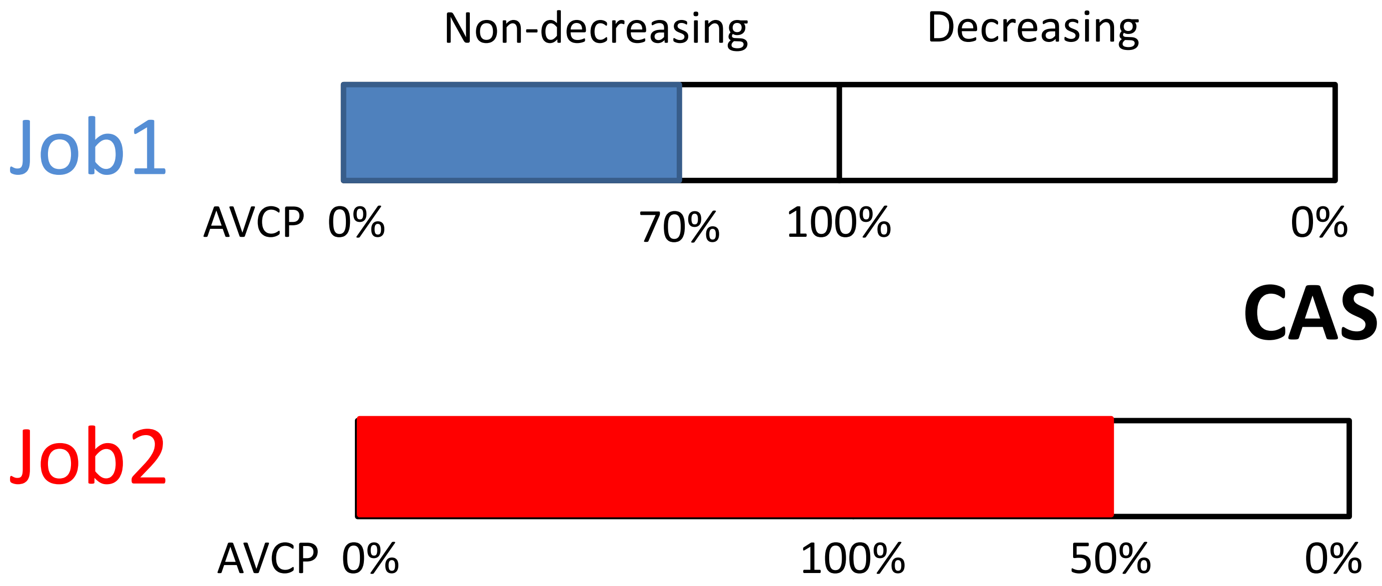


(1) Initial non-decreasing phase: AVCP at or going towards 1
(2) Decreasing phase: AVCP going towards 0

Progress Measure: How far from final AVCP=0%

Progress Comparator Algorithm

MPJ = Max Progress Job

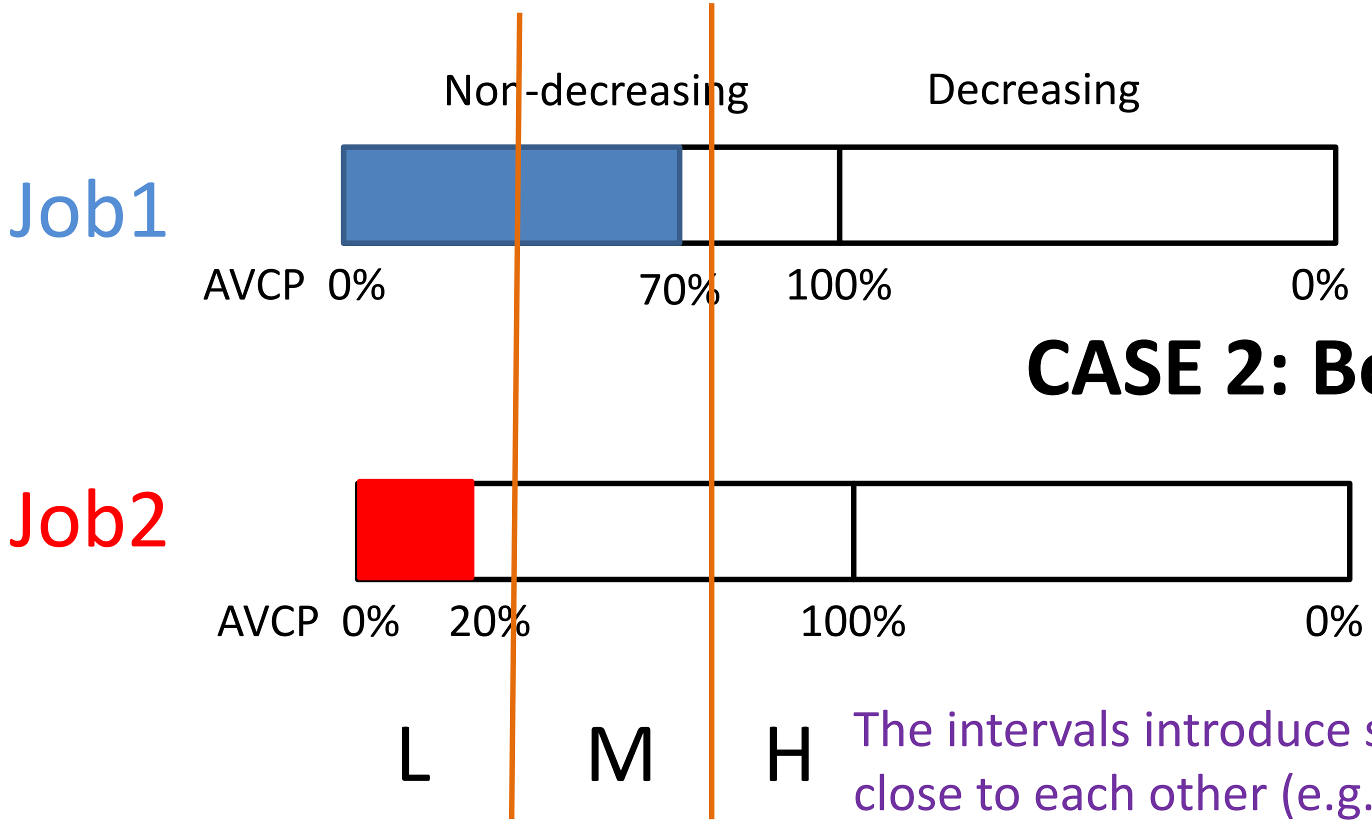


CASE 1: Jobs in different phases

Job2 in 2^{nd} Decreasing Phase: MPJ

Progress Comparator Algorithm (2)

MPJ = Max Progress Job



CASE 2: Both jobs in Non-dec phase

The intervals introduce some randomness for jobs with AVCP close to each other (e.g., if Job 2 was at 60% (M) instead)

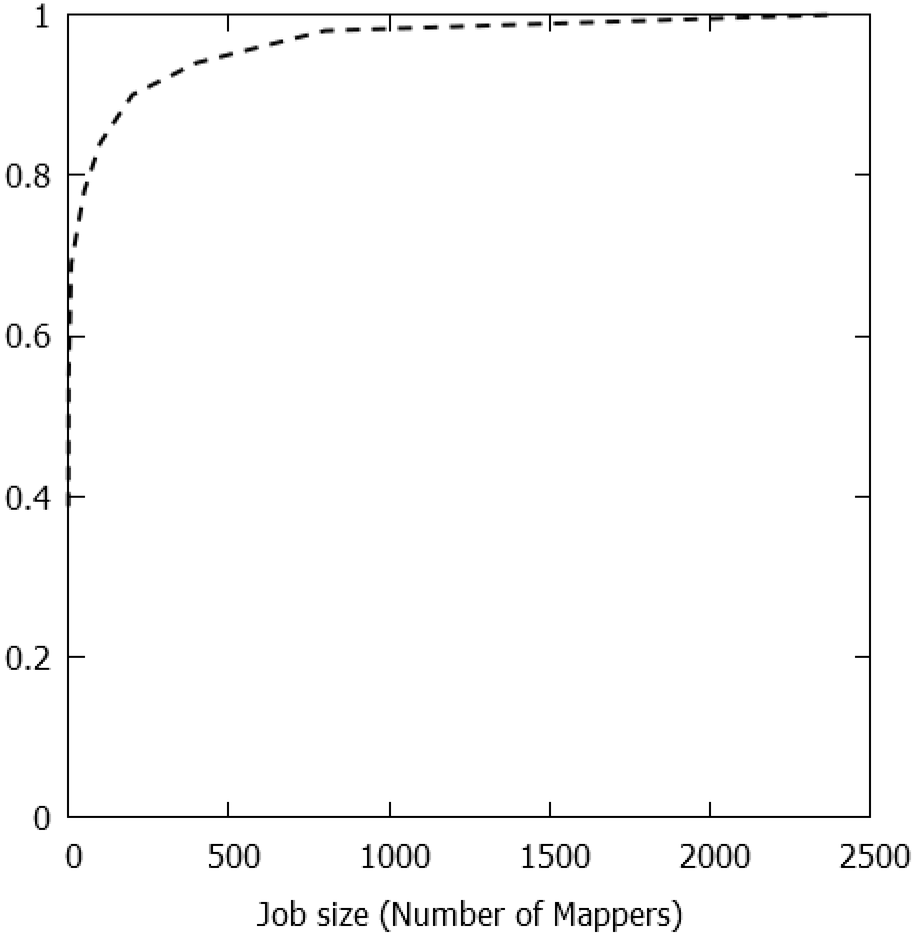
Job1 closer to 100% in first phase: MPJ

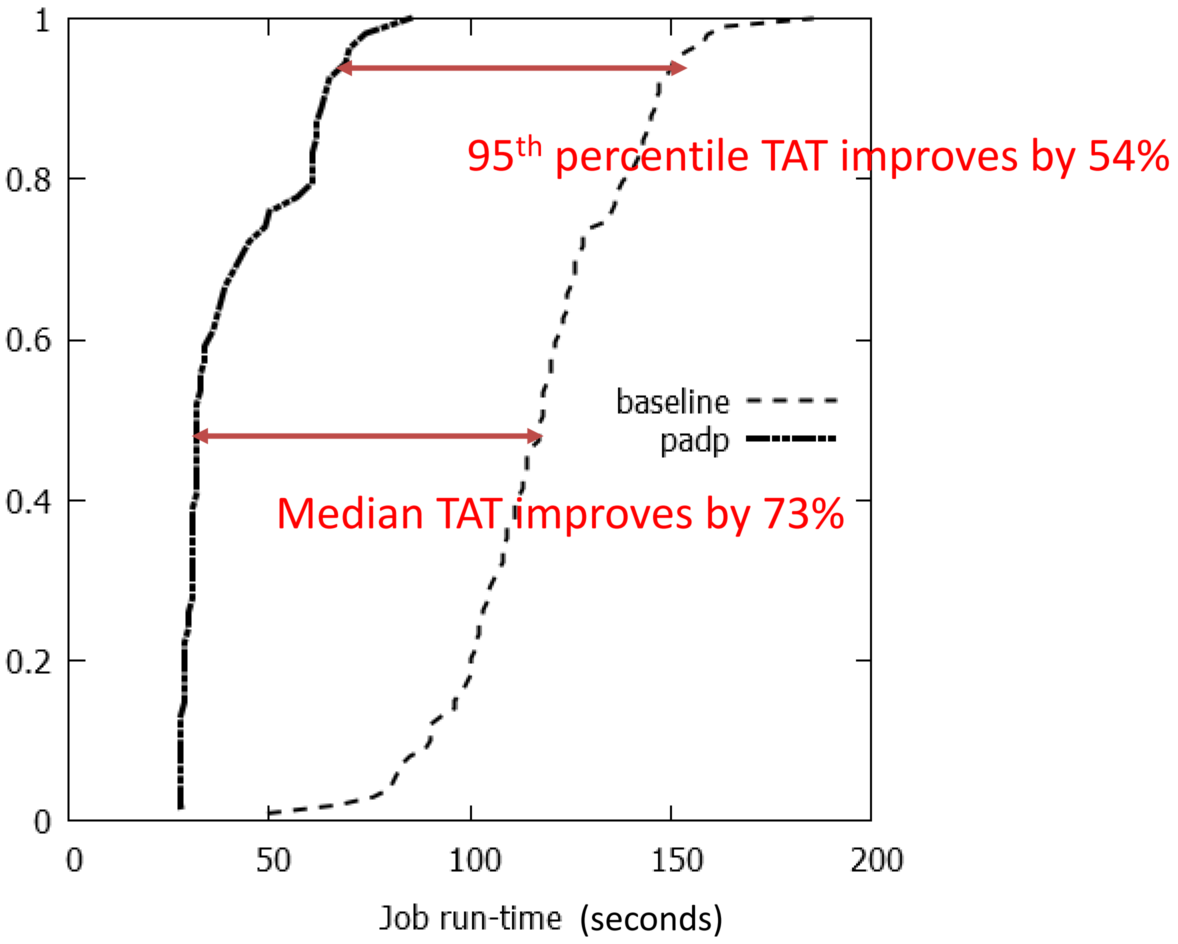
CASE 3: Both jobs in Dec phase (similar)

Evaluation Setup

- Testbed
  - 9 Quad-core servers with 64GB memory, 200GB disks, running Ubuntu 14.04
- Test Algorithms: Single source shortest path (SSSP), K-core decomposition (KC), Page-rank (PR)
- Graphs: Uniform Randomly Generated Synthetic graphs
- Performance Metric: Job completion time
- Compared Scheduling Algorithms:
  - Baseline (B): default YARN FIFO policy (RF=3)
  - PADP (P): OPTiC PADP policy (RF=3 + opportunistically created replica (at-most 1))

Facebook Production Trace Workload

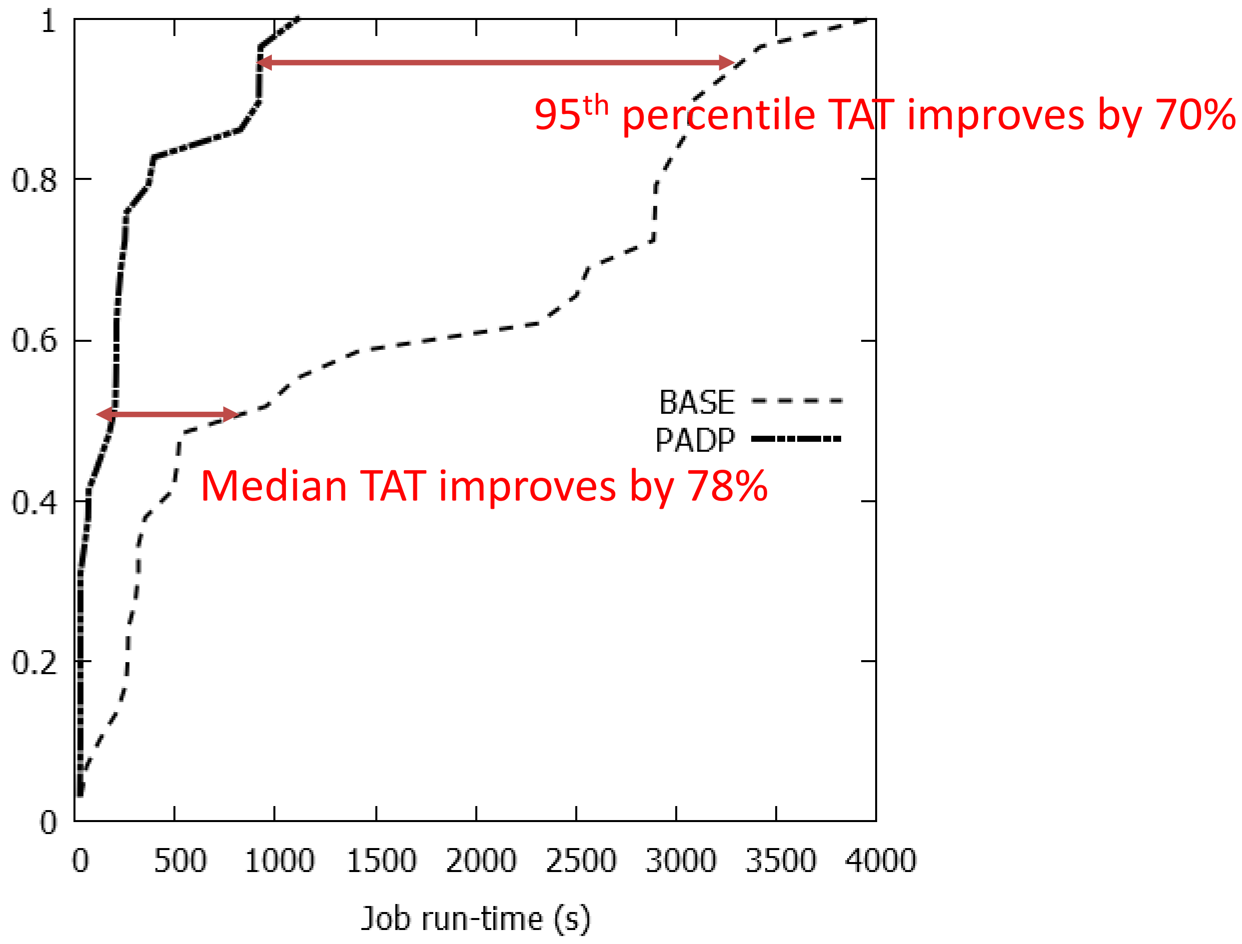




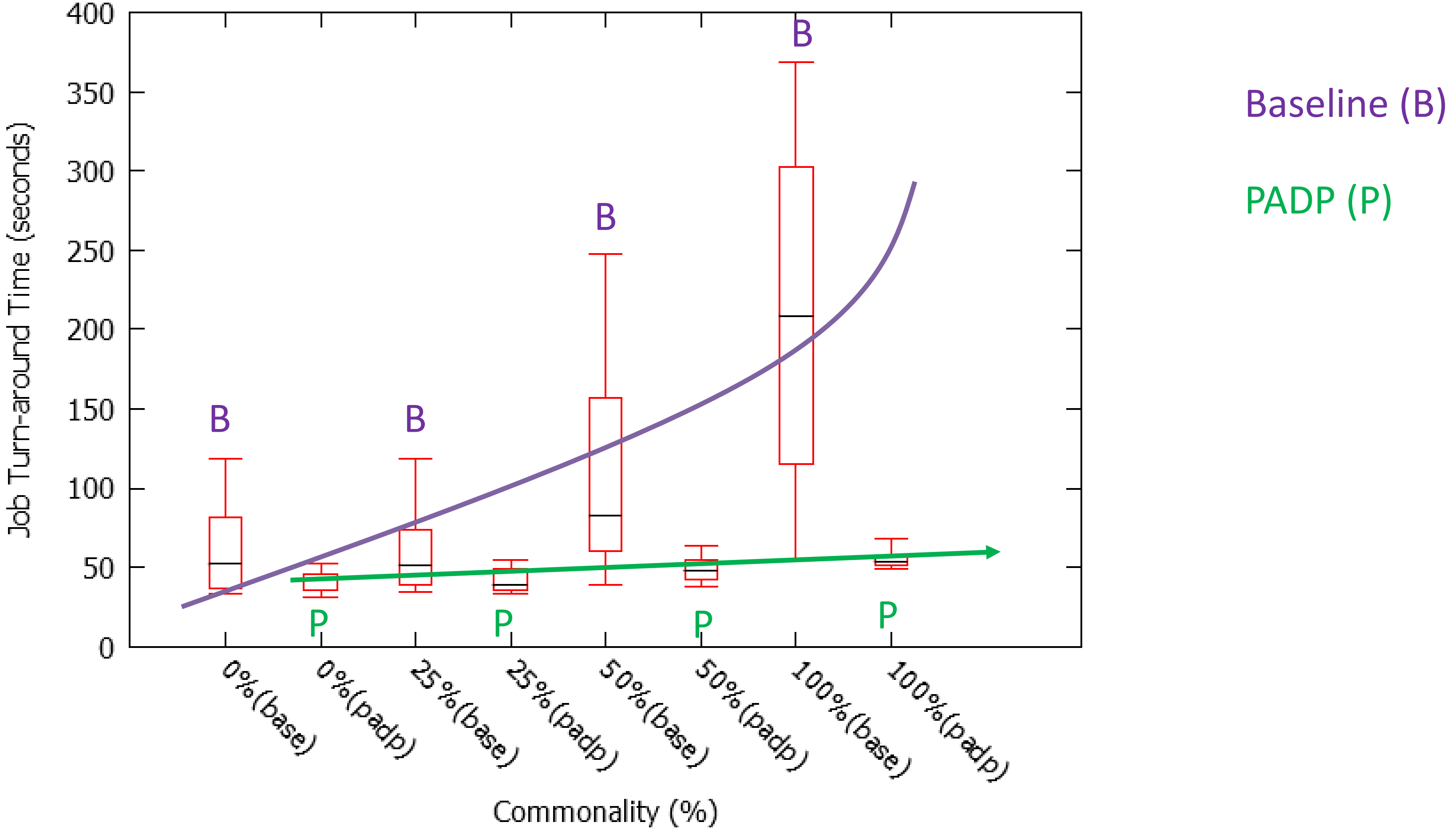
- Job size distribution from Facebook Trace (Vertex count proportional to map count)
- Most jobs in cluster are small
- Poisson arrival process with mean 7s, Network delay LN(3ms)

Yahoo! Production Trace Workload

- Map-reduce job trace from Yahoo! Production cluster of several hundreds of servers
- Trace has 300 jobs with job size and job arrival times
- Bursty arrival process
- Heterogeneous jobs: mixture of SSSP, KC, PR



Scale and Graph Commonality Experiment



- Graph commonality (degree of graph sharing among jobs) increases left to right
- Average graph size also increases from left to right

Related Work

- **Cluster Schedulers (Map-reduce abstraction, multi-tenant)**
  - **YARN**, Fair Scheduler
  - **Mesos**, Dominant Resource Fairness
  - Multi-tenancy with fairness for sharing cluster resources
  - **OPTiC** scheduler aware of graph computation progress
- **Graph Processing (Single-tenant)**
  - **Pregel**, first message passing system based on BSP
  - **GraphLab** proposes shared memory computation
  - **PowerGraph** optimizes for power-law graphs
  - **LFGraph** improves performance with cheap partitioning and publish-subscribe message flow
  - **OPTiC** improves performance for multi-tenant graph processing
- **Progress Estimation**
  - Many systems for estimating progress of map-reduce jobs, e.g., **KAMD**
  - SQL Progress Estimators, e.g., **DNE** (Driver Node Estimator), **TGN** (Total Get Next)
  - **OPTiC** progress estimator based on graph processing level metrics

Summary of OPTiC

- OPTiC is the first multi-tenant graph processing system
- Key techniques
  - Prefetching: we overlap graph pre-processing phase of waiting jobs with computation phase of running jobs
  - Progress Estimation: we propose a new algorithm for estimating progress of graph processing jobs using a graph level metric independent of the underlying cluster and job details
- We obtain 20-82% improvement in job completion time for realistic workloads under realistic network conditions
  - Cost of increased replication of input graph in DFS (3 to 3 + opportunistically created replica (at-most 1))