Modeling and Optimization of Big Data Systems

Li Zhang

System Analysis & Optimization Group

IBM T. J. Watson Research Center

Collaborations with many colleagues, students at IBM and many academic collaborators.

Big Data Systems

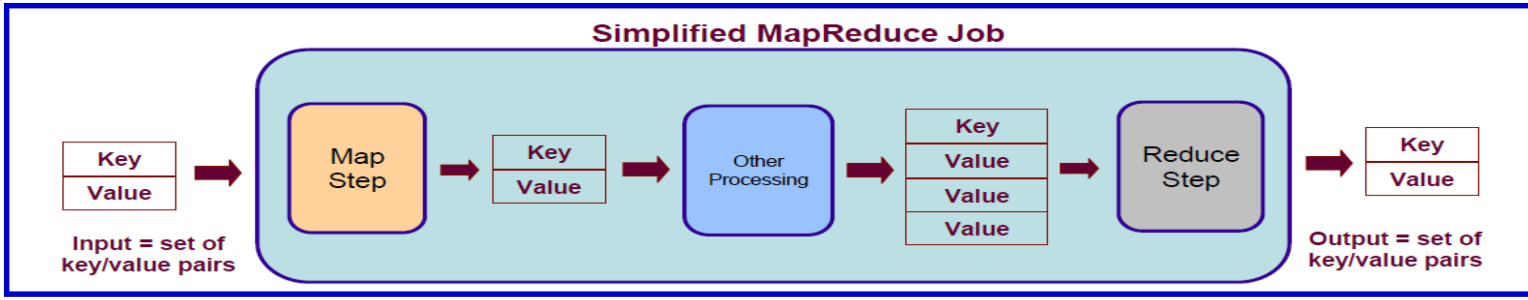
- **Characteristics of Big Data Systems**
  - **Volume**
  - **Variety**
  - **Velocity**
  - **Variability**
  - **Veracity**
  - **Complexity**
- **Examples**
  - **Storage: HDFS, GFS, …**
  - **Processing: MapReduce, Spark, Hive, …**
  - **NoSQL stores:**
    - **Column: Cassandra, Hbase, …**
    - **Document: CouchDB, DocumentDB, MongoDB, …**
    - **Key-value: MemcacheDB, Redis, Aerospike, …**
    - **Graph: Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog, …**
    - **Multi-model: Alchemy Database, CortexDB, …**

MapReduce Systems

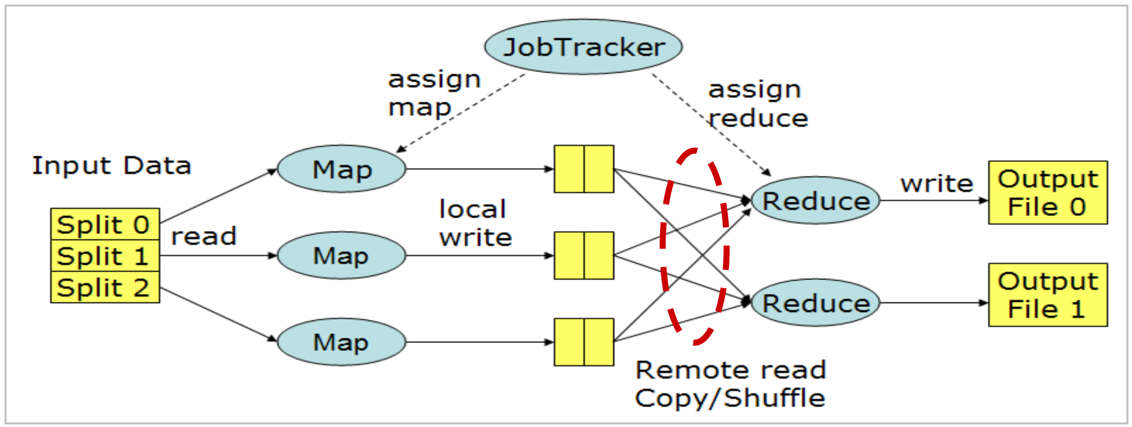
- **Commonly used in Big Data analytics**
  - **By Facebook, Yahoo, Google, TaoBao, …**
- **Simple workloads**
  - **Word count, grep, sort, sampling, …**
- **Complex workloads**
  - **NutchIndexing, PageRank, Bayesian classification, K-means clustering, log analyzer, simulation, …**
  - **SQL like queries (Hive, Jaql, …) compiled to DAG of MapReduce jobs, …**
- **Map heavy**
  - **Word count, grep, sampling, …**
- **Reduce heavy**
  - **Sort, queries, …**
- **Data (I/O) heavy**
  - **Sort**

MapReduce overview

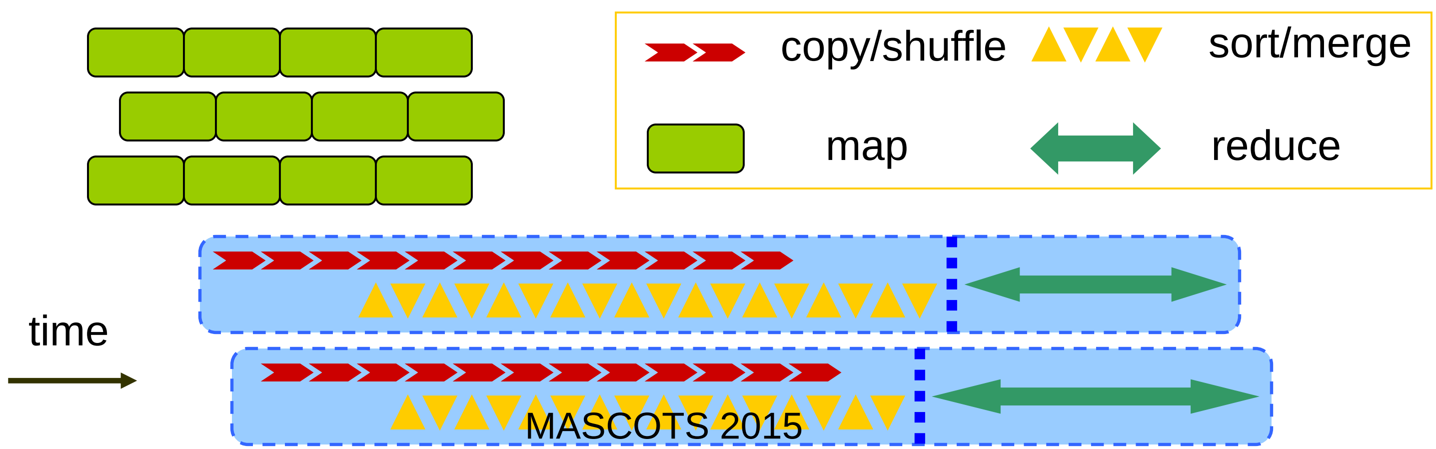
Function View



System View



Process View



Map Reduce Modeling & System Optimization

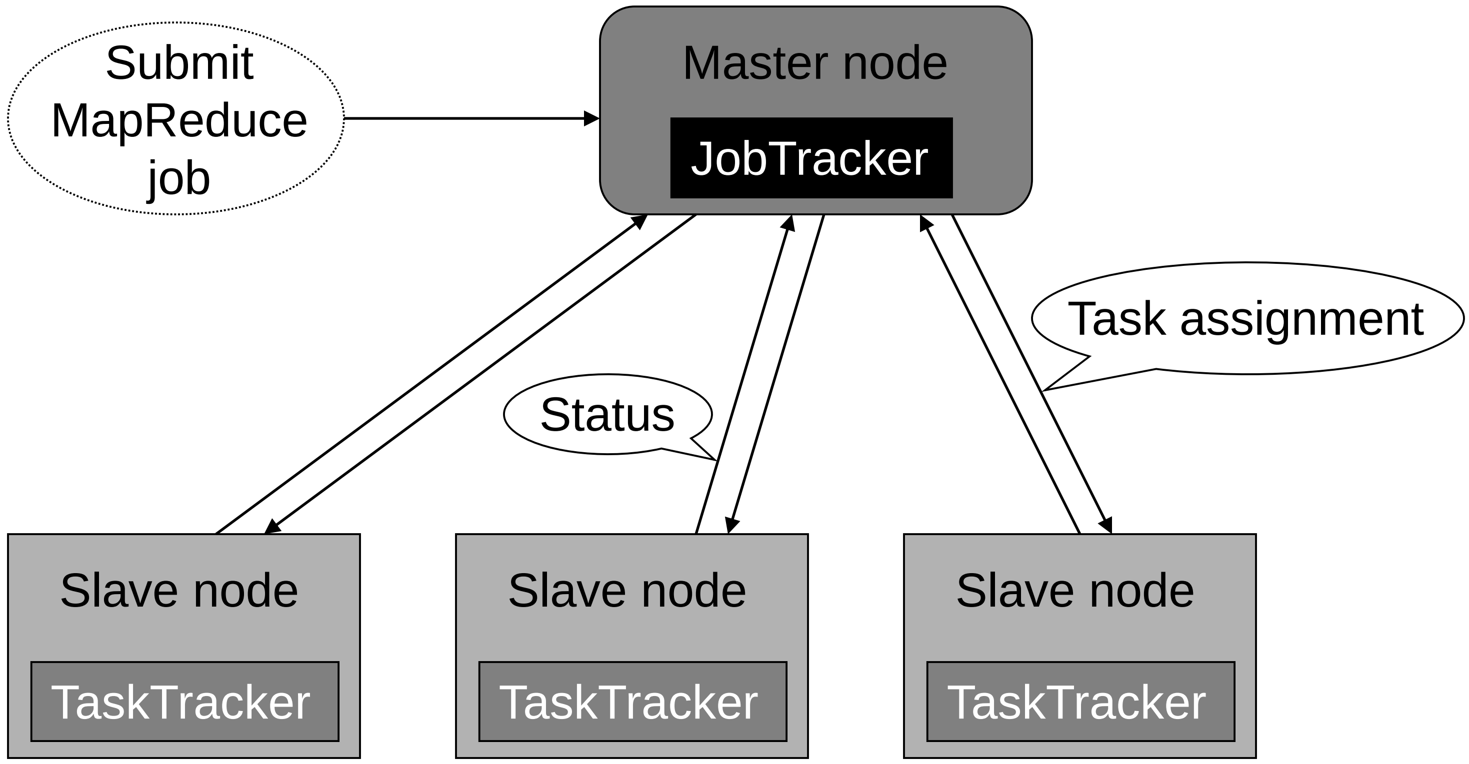
Goal

- **Identify inefficiencies in MapReduce mechanisms & fix them**
- **Improve the scheduling mechanism**
- **Performance modeling based approach for capacity planning of MapReduce applications**

Our past work

- **Worked on both Hadoop & IBM Platform Symphony**
- **Inefficiencies:**
  - Reduce starvation, improve data locality, avoid scheduling delay
- **Better scheduling & memory management**
  - Pause/resume for reducers, task interleaving
- **Performance modeling and capacity planning**
  - Benchmarking on Symphony clusters for representative benchmarks
  - Gray box performance models to
    - capture perf metrics (e.g. completion time) as a function of job & system parameters (e.g. data size, cluster size, # of map tasks & reduce tasks, …)
  - Help users determine required capacity setting for a target level SLA
- **Deliver capabilities to IBM Cloud products and services**

Scheduling: heartbeat mechanism



A scheduler is critical for good performance in presence of multiple jobs

1) >25,000 MapReduce jobs/day (Facebook 2010)
2) Short jobs after large ones (trace study)

Scheduling is Not Easy!

- **Complexities**
  - Multiple phases for each job
  - Multiple resources (cpu, or I/O, or network) may be stressed
  - Fork and join feature for map/copy/shuffle phase
  - Jobs with different characteristics
    - Map heavy, reduce heavy, …
  - Move computation vs move large amount of data
  - May not be work conserving
- **Many Existing Schedulers (e.g. Fair)**
  - Lack of cooperation between map and reduce tasks
    - Most existing work only optimizes the scheduling of map tasks or reducers independently
  - Dependence between map and reduce causes starvation
  - No consideration of Reduce task locality (which depends on map task allocations)

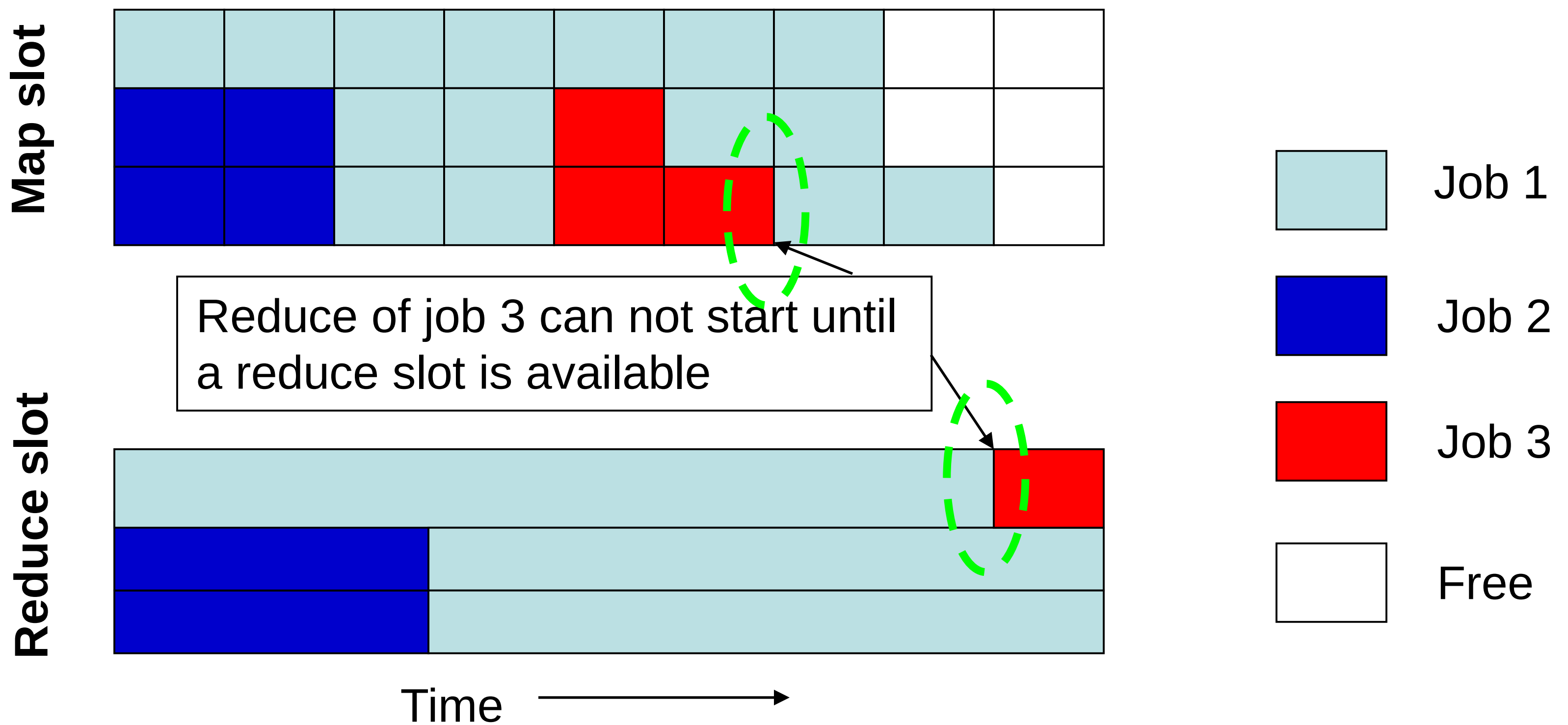
Existing work

- **Fair (Facebook)**
  - **Ensure a minimum number of slots to a job (fair for maps)**
- **Quincy and Mantri for Dryad (Microsoft)**
  - **Support a graph represented data processing model (DAG – direct acyclic graph)**
- **Capacity (Yahoo)**
  - **Support for multiple queues each with a fraction of capacity; a job is submitted to a queue**
- **Others**
  - **LATE – scheduling speculative tasks**
  - **Delay – improve data locality**
  - **Deadline oriented schedulers**
- **Research at IBM**
  - **FLEX – add-on module to Fair to optimize a number of metrics**
  - **M3R – Main Memory MapReduce engine in X10**
  - **Platform Computing**

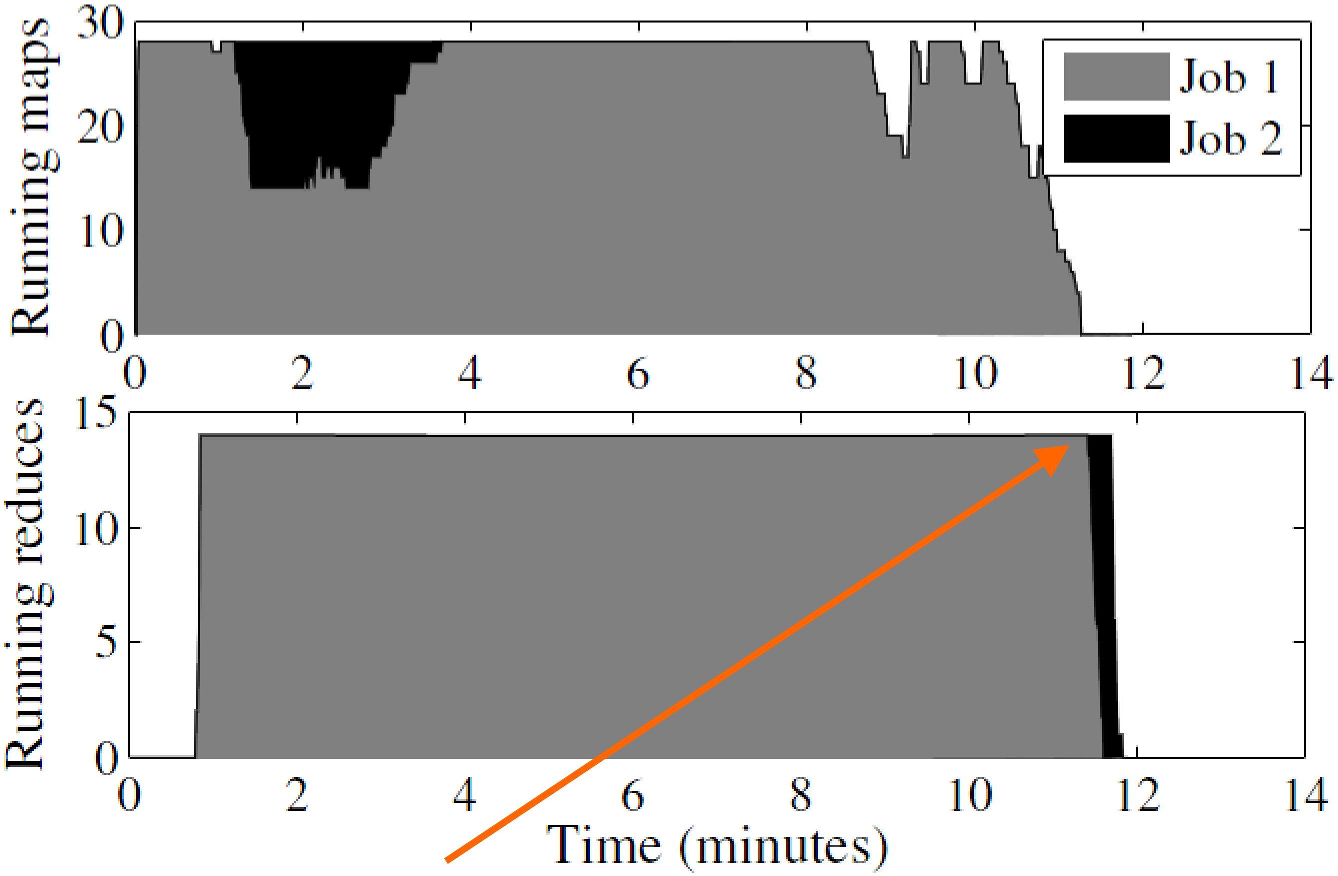
Starvation problem

Difference between map and reduce

- Map: small and independent, run in parallel
- Reduce: long (fetch/shuffle → sort/merge → reduce)
  - Launched in a greedy manner



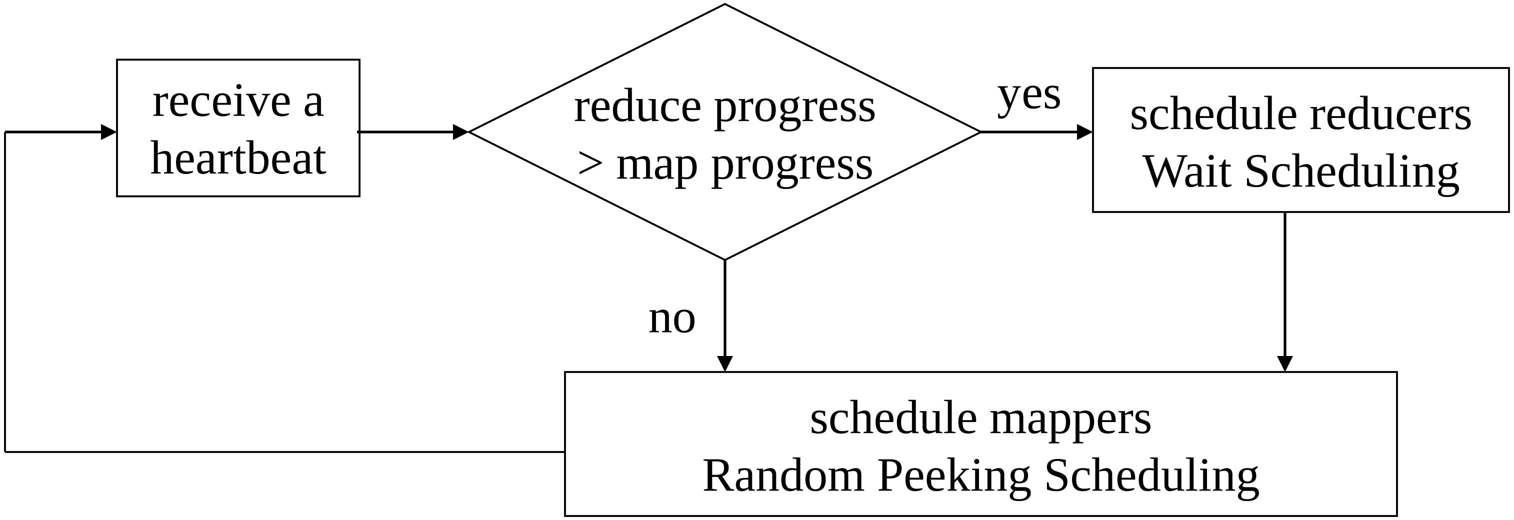
Real experiment



▶ **If only Job 2's reduce could start earlier …**

Scheduler design

- **Coupling: launch reduce tasks according to the progress of finished map tasks**
  - **Reduce starvation**
- **Wait scheduling for reduce: place reducers close to the "centrality" of the intermediate data on the tree topology by skipping some received heartbeats**
  - **Reduce data movement (improve data locality for reduce)**
- **Random peeking for map: allow launch map task on remote node**
  - **Avoid scheduling delay (in large clusters)**

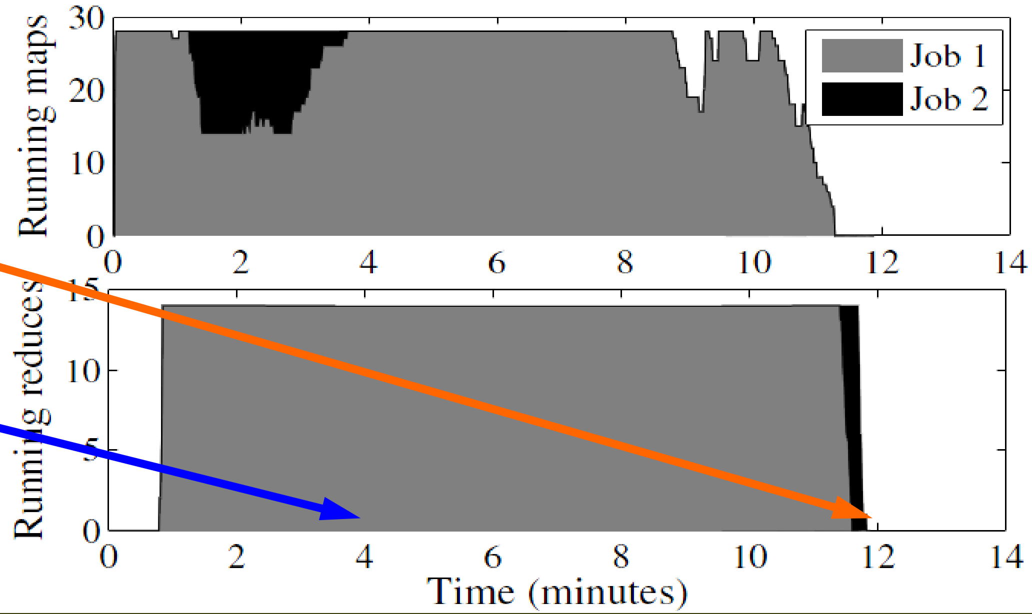


Coupling to mitigate starvation

Fair

Job 2 finishes late

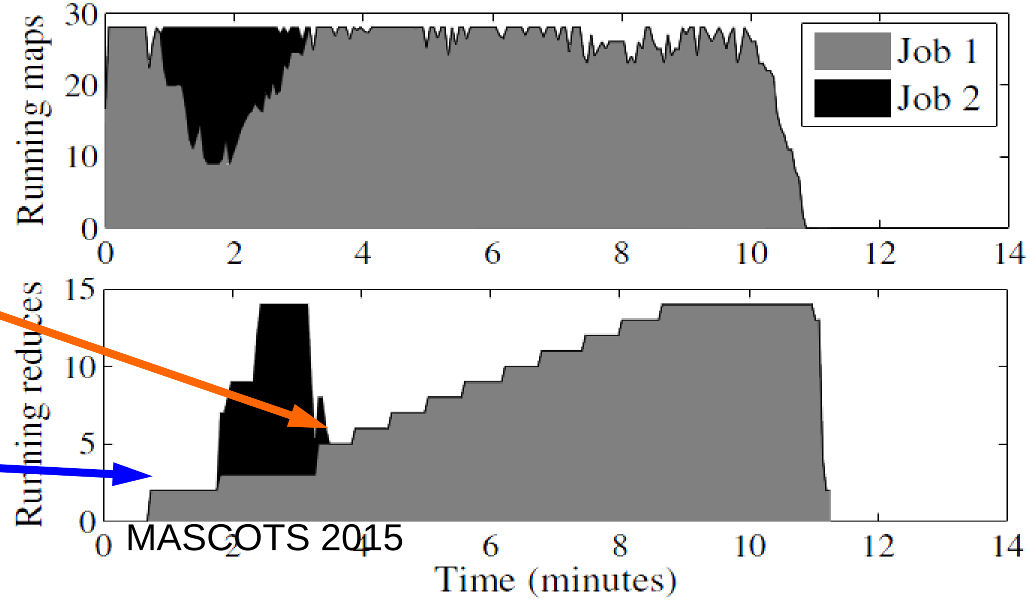
Job 2 starts to wait
For reduce slots



Coupling

Job 2 finishes early

Gradually launch
reduce tasks

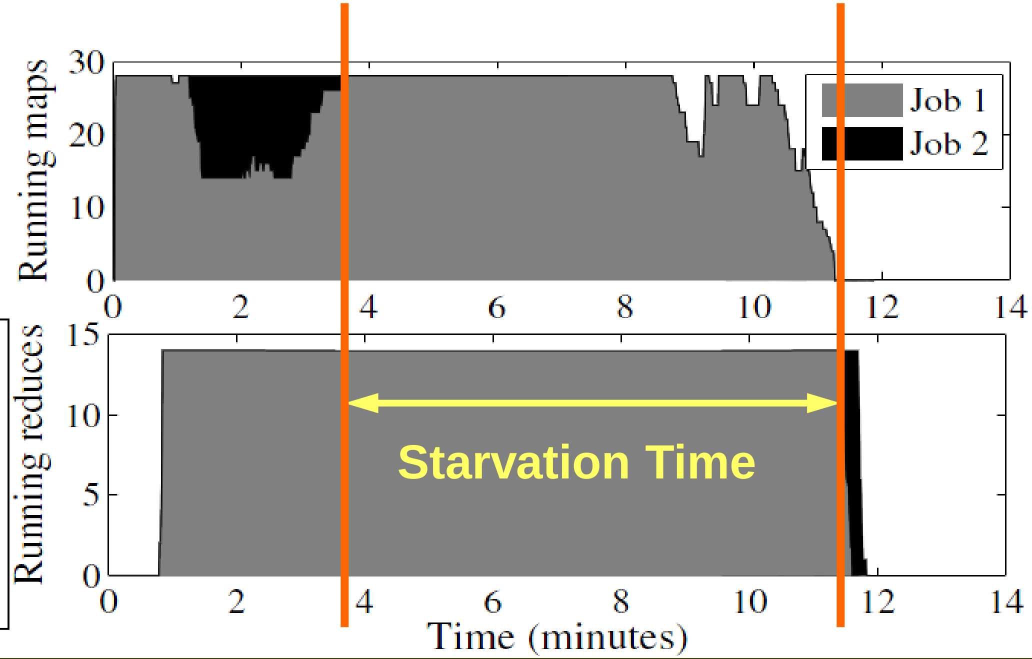


Starvation time

Fair

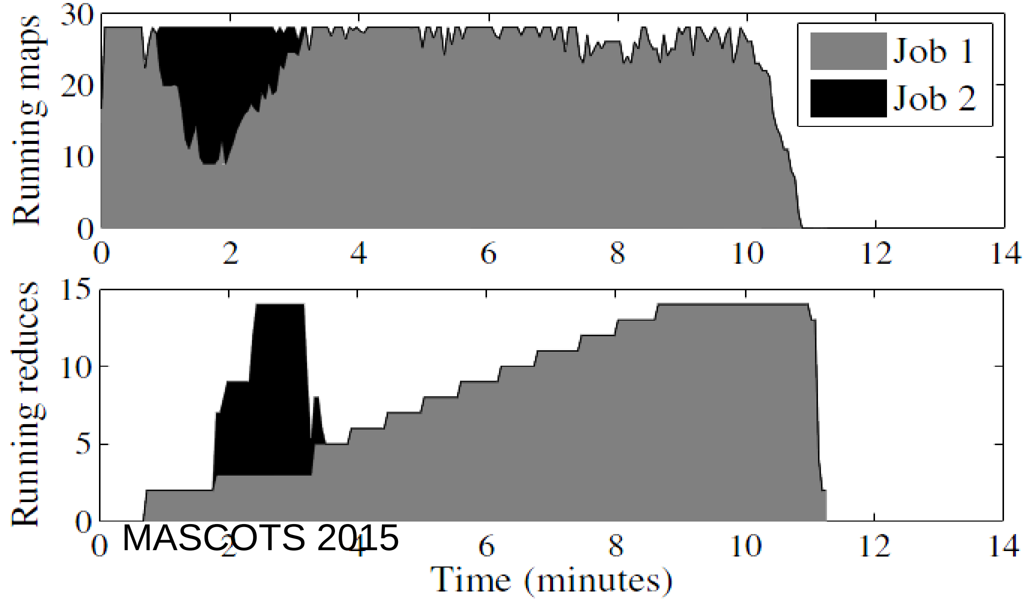
Long Starvation Time

Starvation Time :=
Average time between completion of last map task and start of each reduce task



Coupling

0 Starvation Time



Reduced starvation

- **Experiment**
  - 22 jobs: map heavy (Grep, QuasiMonteCarlo), reduce heavy (sort), small, large …
  - Repeat 5 times
  - Coupling scheduler reduces starvation time

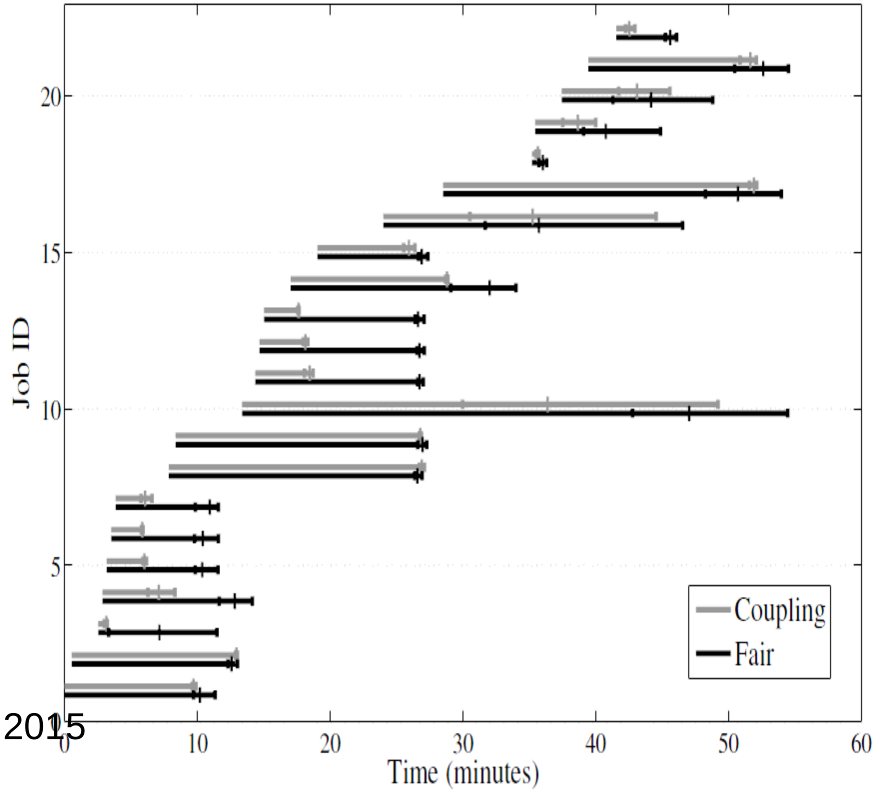
Starvation for Fair

Starvation for Coupling

JOB SEQUENCE

JobID	Job	Time	M	R	S_F	S_C
01	Grep [1-5]* randomInput	0	148	15	0.0	0.0
02	Grep [5-9]* randomInput	30	148	15	0.0	0.0
03	QuasiMonteCarlo	150	5	1	6.1	0.0
04	WordCount randomInput05	170	8	1	4.0	0.0
05	Grep [2-6]* randomInput05	190	8	2	3.9	0.0
06	Grep [3-6]* randomInput05	210	8	2	2.8	0.0
07	Grep [4-6]* randomInput05	230	8	3	2.8	0.0
08	Grep [a-h][a-z]* wikiInput	470	427	15	0.0	0.2
09	Grep [a-g][a-z]* wikiInput	500	427	15	0.0	0.0
10	Sort randomPair1	800	224	27	4.9	4.5
11	Grep [1-2]* randomInput10	860	15	5	8.3	0.9
12	Grep [1-5]* randomInput05	880	8	3	8.6	0.6
13	Grep [6-9]* randomInput05	900	8	2	8.4	0.2
14	Sort randomPair3	1020	64	27	5.3	3.5
15	Grep [3-8]* randomInput20	1140	30	2	1.0	0.0
16	WordCount randomInput10	1440	15	1	0.1	0.0
17	Sort randomPair2	1710	352	27	0.3	5.0
18	QuasiMonteCarlo	2110	15	1	0.3	0.0
19	Grep [1-5]* randomInput05	2125	8	3	0.0	0.0
20	Sort randomPair3	2245	64	27	0.5	1.1
21	RandomWriter	2365	150	0	0.0	0.0
22	QuasiMonteCarlo	2485	10	1	0.0	0.0

Processing Time (40% reduction)



Larger, more realistic experiment

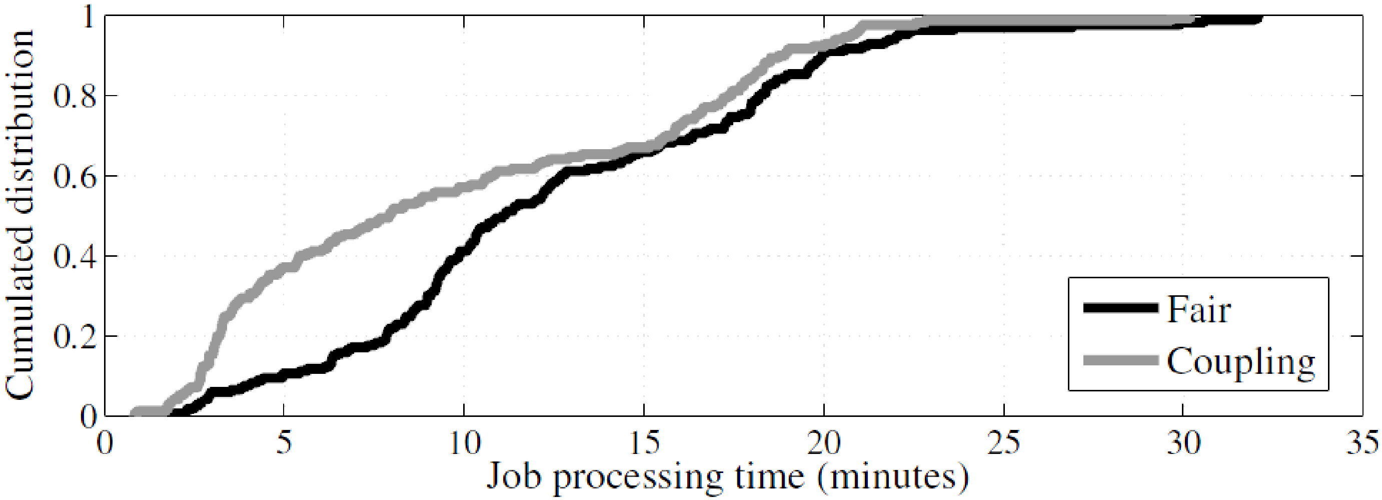
Test bed

One master node and 62 slave nodes; each node has 4 cores (2933MHz, 32KB cache size),

6GB of memory and 72GB of disk.

Set 4 map slots and 2 reduce slots for each node.

Compare job processing times distribution (200 jobs)



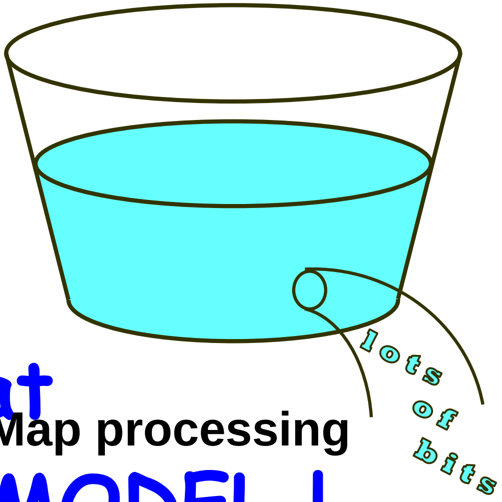
21.3% improvement in average job processing time

It's all good and cool and dry so far.

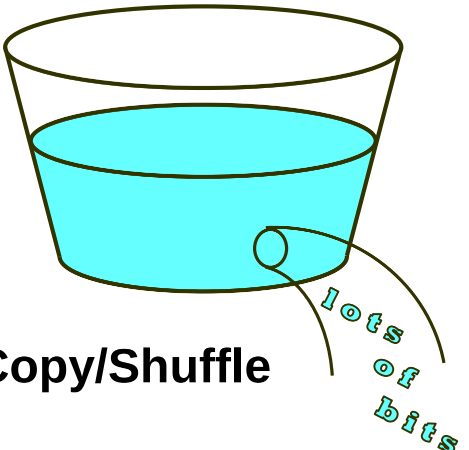
But, what about a MODEL?

One does not get to present at MASCOTS without showing a MODEL !

Let's get wet !

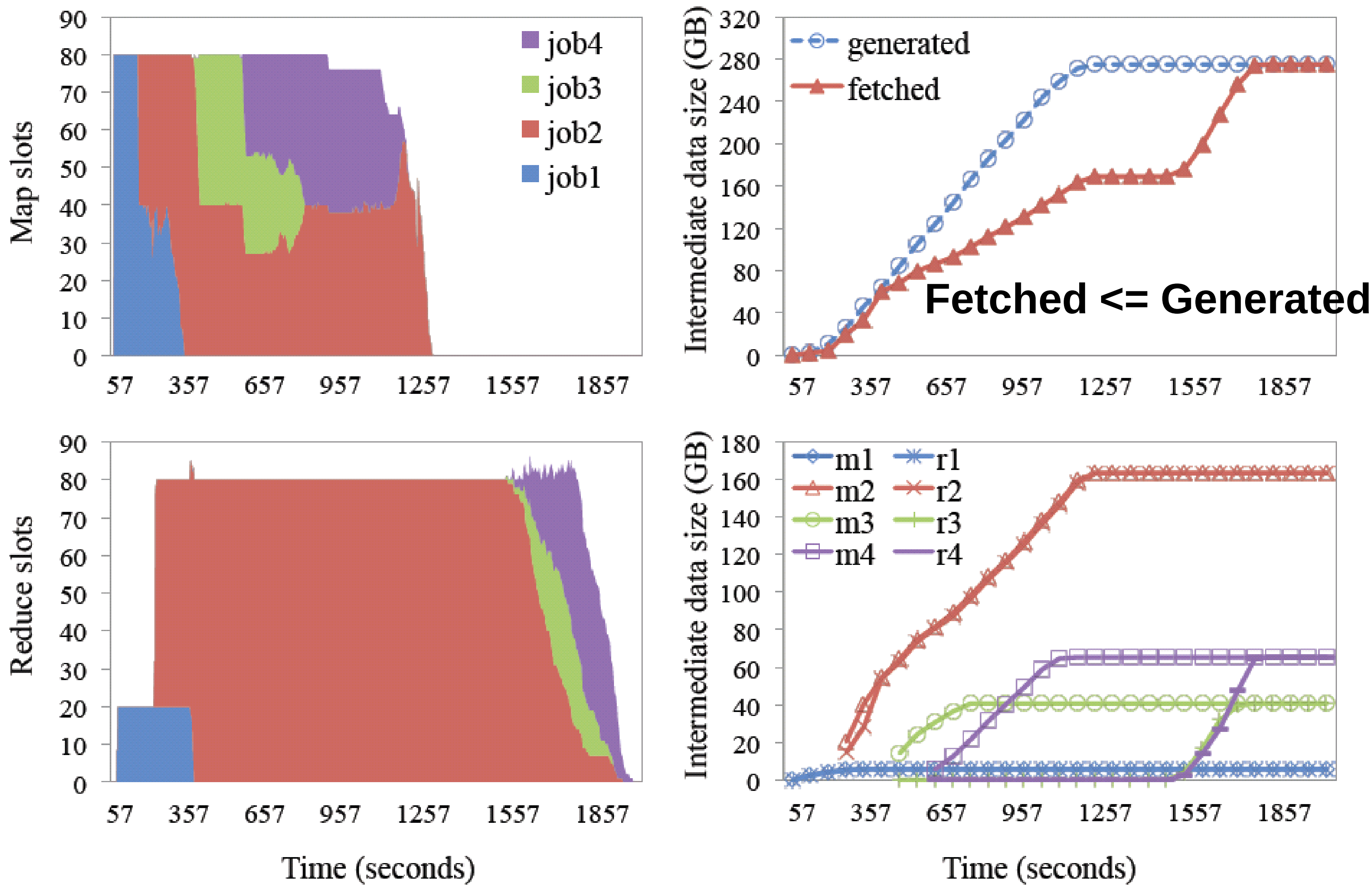


Map processing

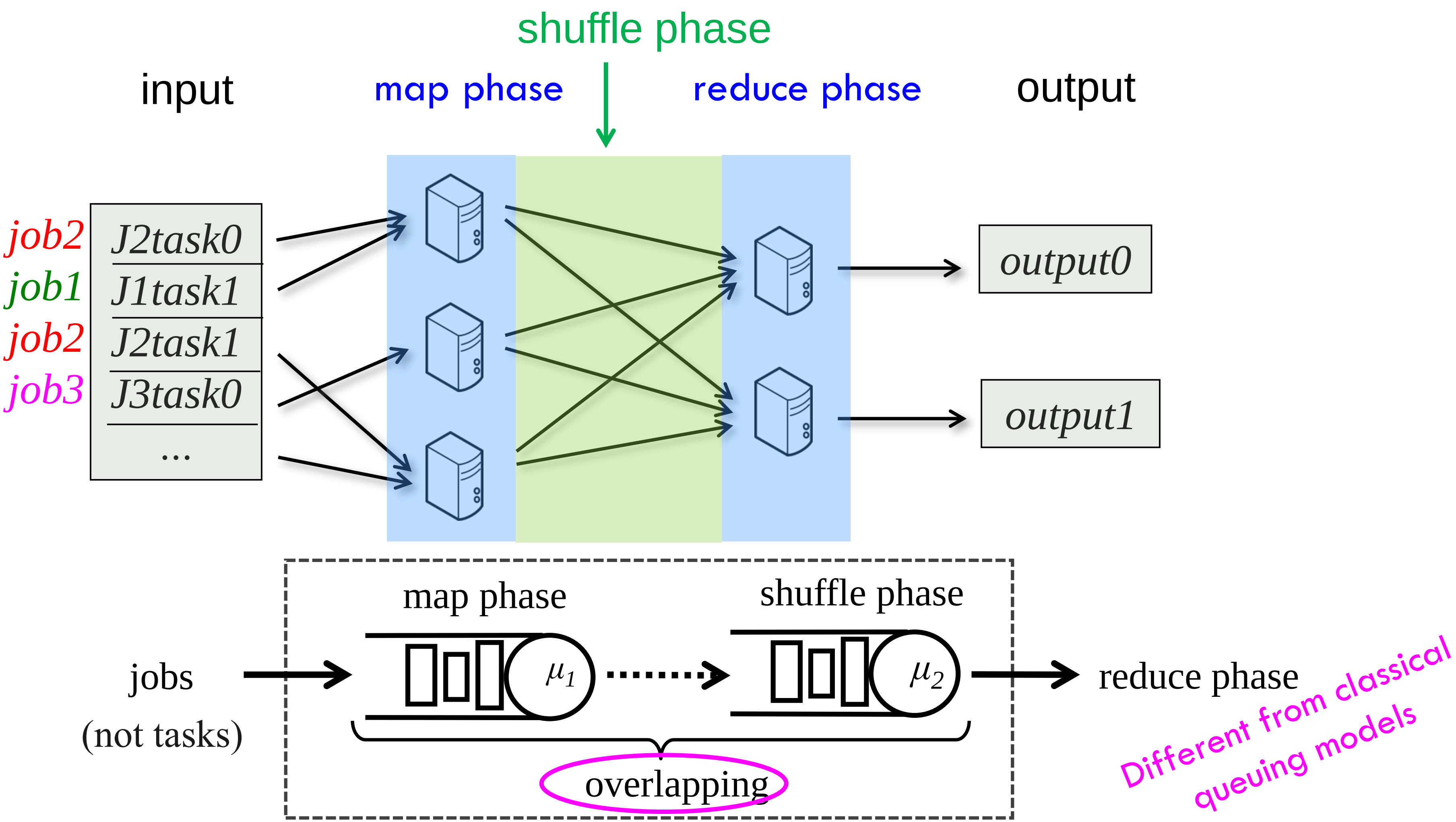


Copy/Shuffle

Queueing model & analysis



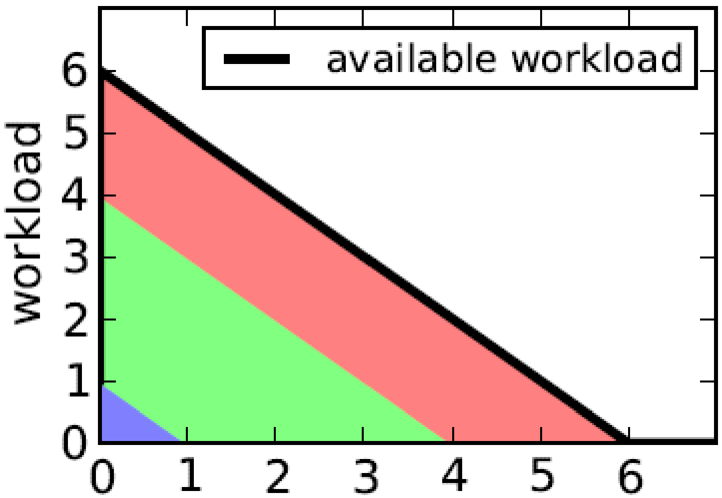
Overlapping Tandem Queue



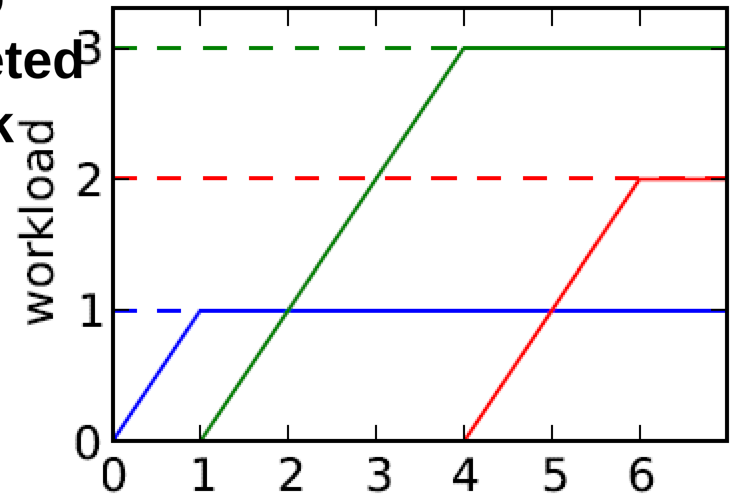
s.t. $\text{progress}_1(\text{job-i}) \geq \text{progress}_2(\text{job-i})$

Overlapping Tandem Queue

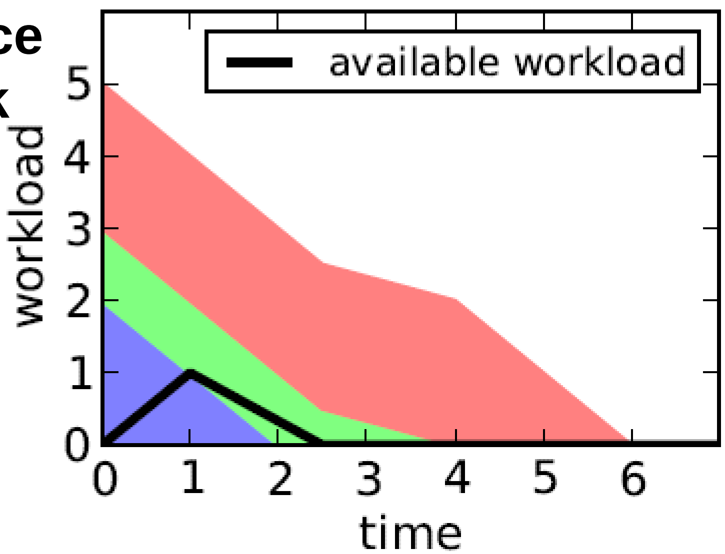
Map work



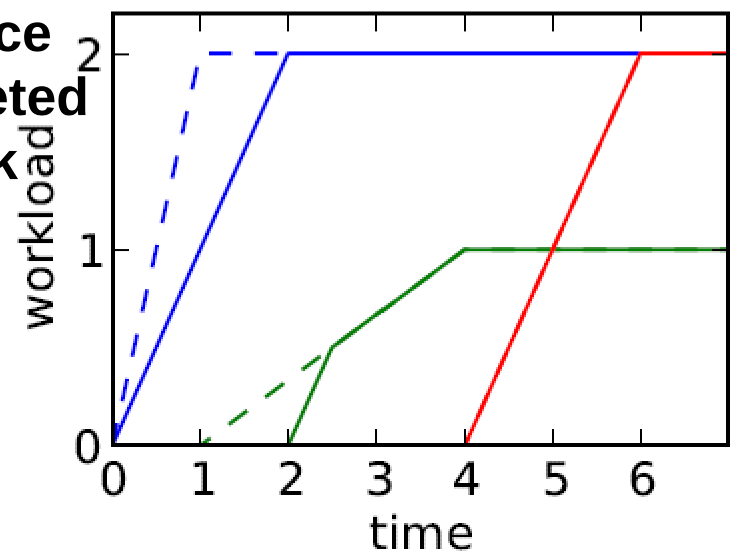
Map completed work



Reduce work

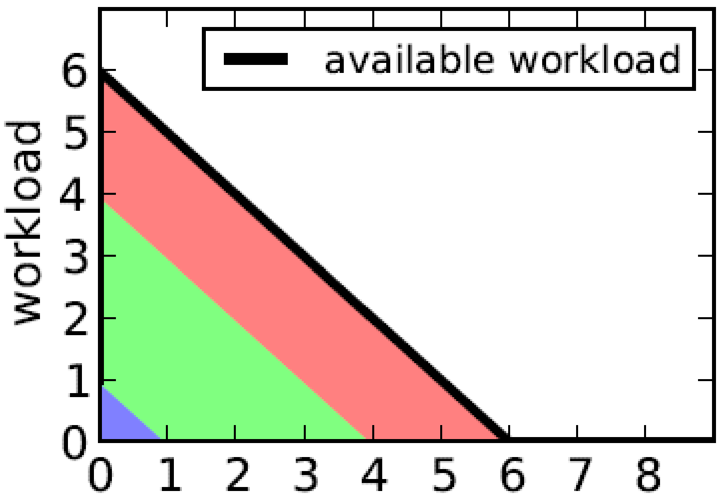


Reduce completed work

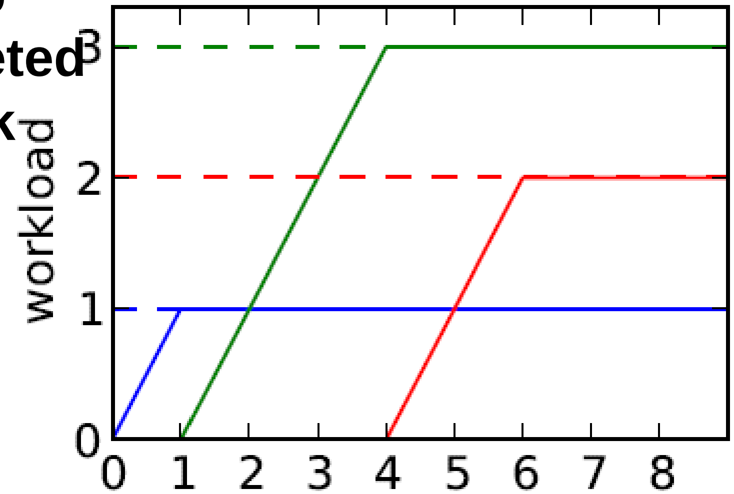


Traditional Tandem Queue

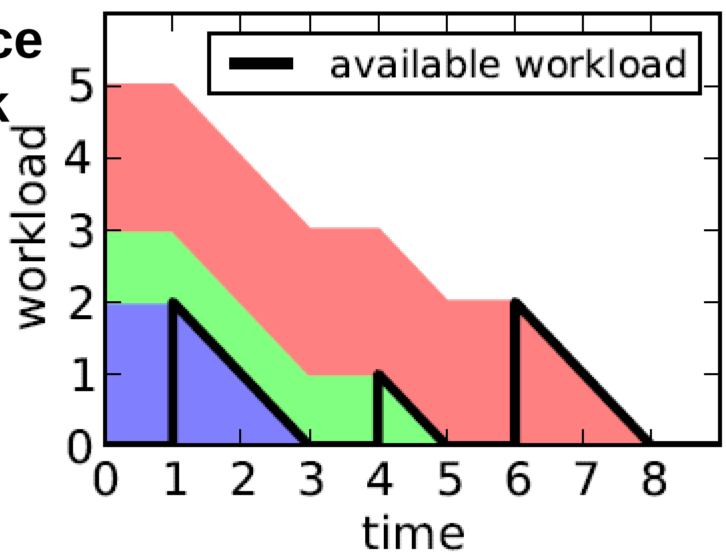
Map work



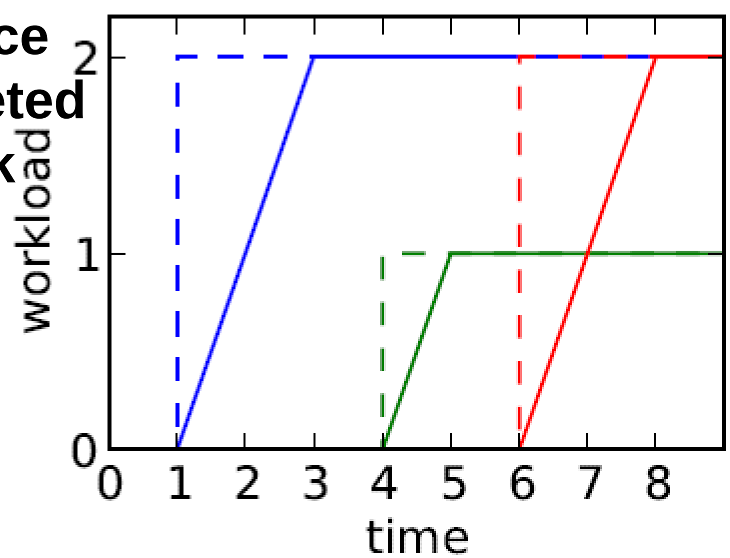
Map completed work



Reduce work



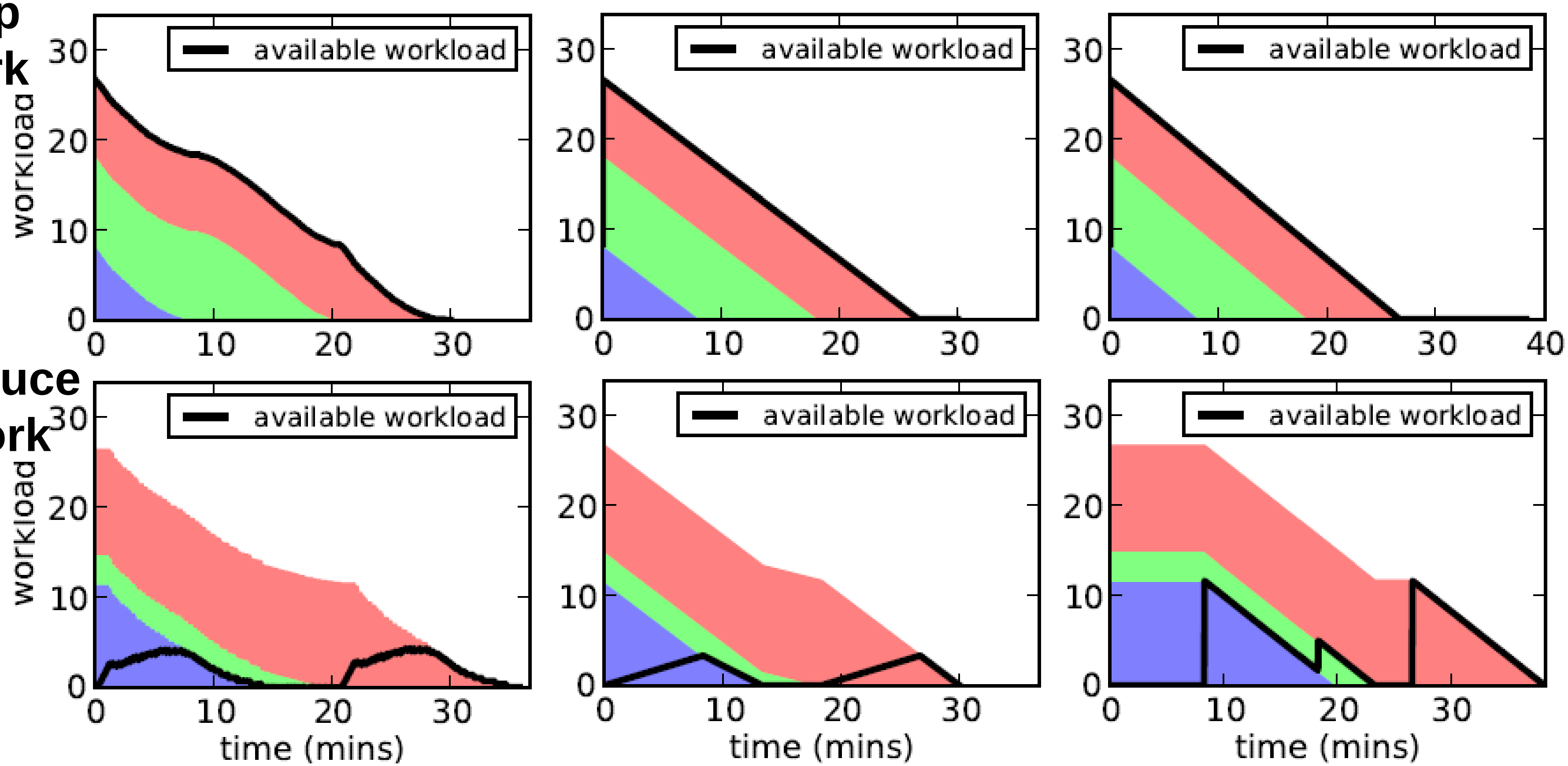
Reduce completed work



Which model fits better?

Map work

Reduce work



Measurement from real system

Overlapping Tandem Queue

Traditional Tandem Queue

Difference from classical models

- **Overlapping tandem queue**
  - **Fluid model: big data is like fluid!**

- **Tandem queueing model**
  - **No overlapping**
  - **Usually Poisson arrivals**
  - **Usually independent exponential service at each station**

- **Flow shop model**
  - **No overlapping**
  - **Batch jobs**
  - **Focus on non-preemptive scheduling and makespan criterion**

Model parameters

Reasonable assumption when each job has more tasks than machines.

μ_1= total processing capacity of all map slots

μ_2= total network capacity between map slots and reduce slots

normalizing $\mu_1=1, \mu_2=1.$

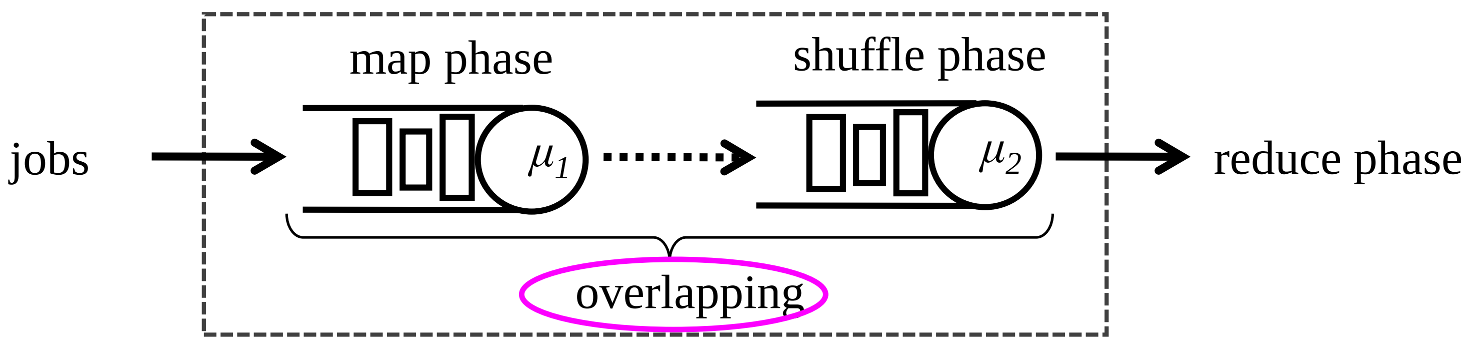
size of *job-i*: x_i y_i

service time: x_i for map phase,
y_i for shuffle phase

can be estimated in practice:

(a) Run a few map tasks of *job-i*.

(b) Linear prediction based on task time and intermediate data size.



s.t. $\text{progress}_1(\text{job-i}) \geq \text{progress}_2(\text{job-i})$

Online scheduling algorithm: MaxSRPT

Focus on "finishing small jobs early".

time to go through an idle system

Algorithm. *Both stations work on the jobs using SRPT based on* $\max(x_i(t), y_i(t))$ *subject to data availability.*

How good is it?

Not more than 2

Theorem. *Denote* $\alpha = \max_i \max(x_i/y_i, y_i/x_i)$. *The algorithm is* $2\alpha/(1+\alpha)$-*speed optimal.*

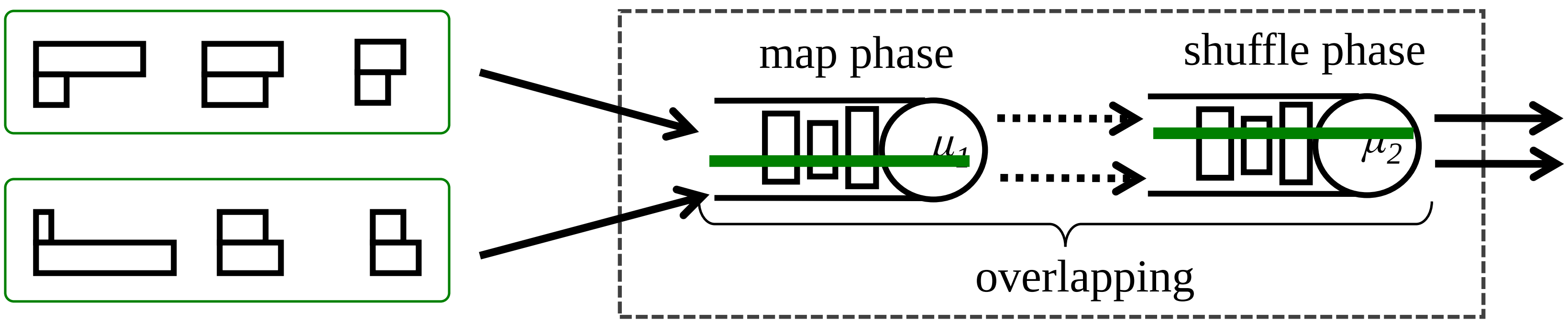
→ **Better for "balanced jobs"**

Online scheduling algorithm: SplitSRPT

Focus on "keeping the shuffle phase busy".

→ **Mix map-heavy job and shuffle-heavy job.**

Algorithm. *Denote* $\beta = \min_i \max(x_i/y_i, y_i/x_i)$. *Split the capacity of the map station so that* $\mu_{m1} : \mu_{m2} = 1 : \beta$. *Split the capacity of the shuffle station so that* $\mu_{s1} : \mu_{s2} = \beta : 1$. *For the new arrival* J_k, *update* $S_1 = S_1 \cup \{J_k\}$ *if* $x_k/y_k \geq 1$ *and* $S_2 = S_2 \cup \{J_k\}$ *if* $x_k/y_k < 1$. *Run jobs in* S_1 *by SRPT (map size) using* μ_{m2} *and* μ_{s2}. *Run jobs in* S_2 *by SRPT (shuffle size) using* μ_{m1} *and* μ_{s1}.

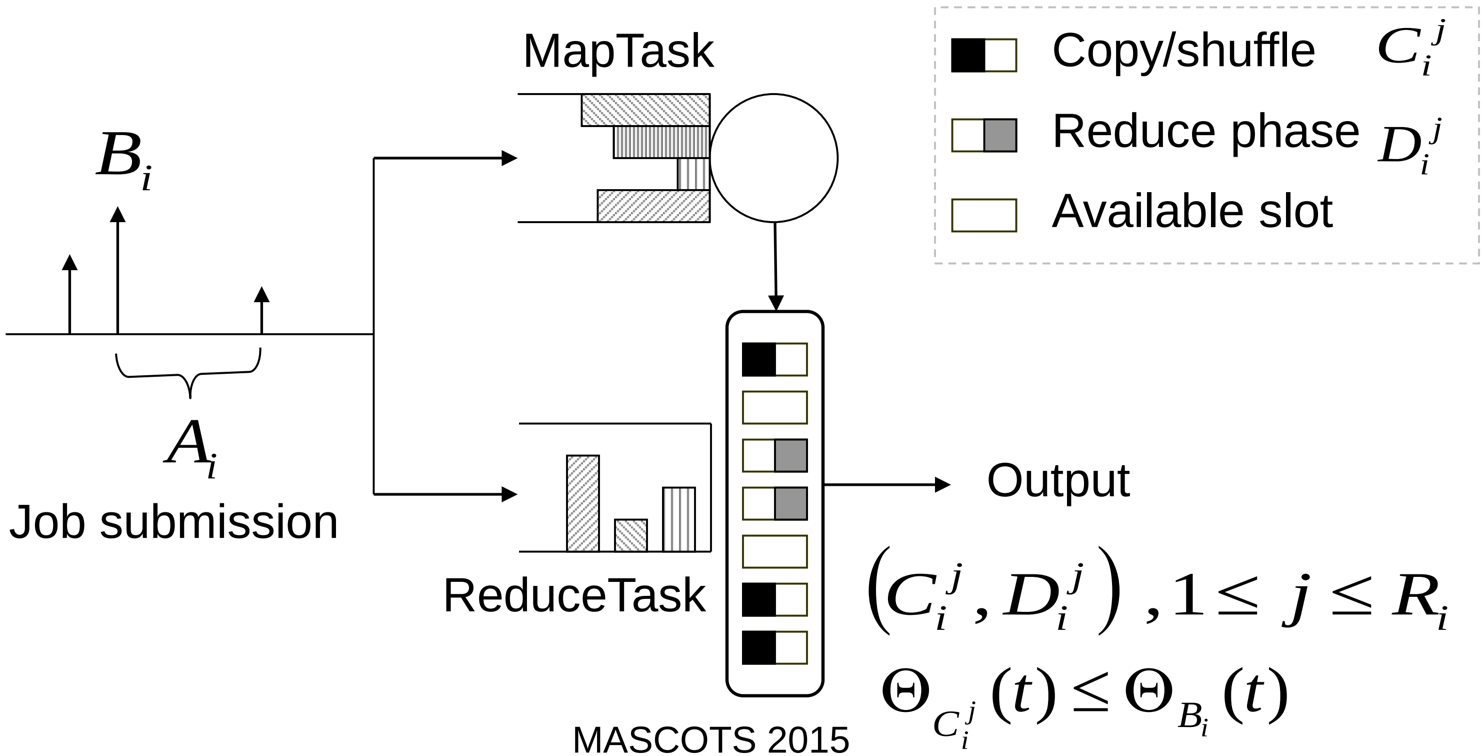


Theorem. *Denote* $\beta = \min_i \max(x_i/y_i, y_i/x_i)$. *The algorithm is* $(1+1/\beta)$*-speed optimal.*

→ **Better for "unbalanced jobs"** **Not more than 2**

Queueing model

- ***Ai* – arrival interval**
  Poisson process
- ***Bi* – map service of job i**
  Power law
- ***Ri* – # of reducers of job i**
  specified by users
- **Task progress constraint**
  $$\Theta_{C_i^j}(t) \le \Theta_{B_i}(t)$$



Heavy-tailed workload characteristics

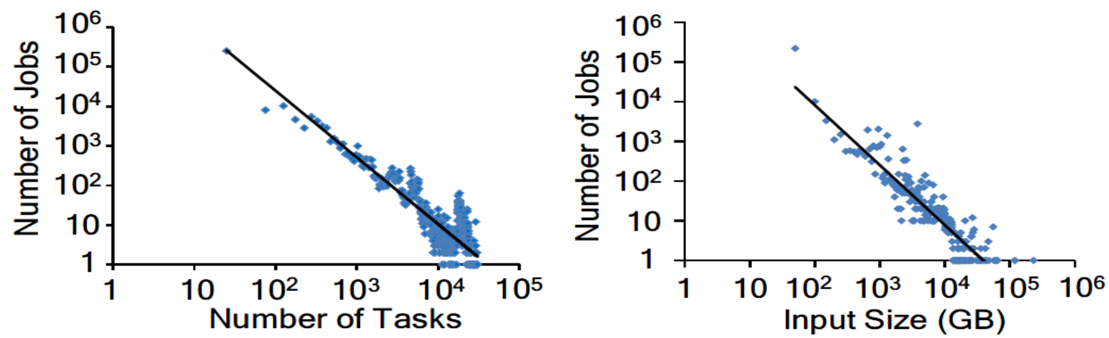
S. Kavulya, et al. *An analysis of traces from a production mapreduce cluster*, CCGRID '10

B_i - i.i.d. map service is regularly varying with index α

$$P[B_i > x] = l(x)/x^{\alpha}, x \geq 0, \alpha > 1$$

l(x) is slowly varying

$$\lim_{x \to \infty} l(\lambda x)/l(x) = 1, \lambda > 0$$



(a) Number of tasks (b) Input Size

Figure : **Power-law distribution of jobs (Facebook) in the number of tasks and input sizes. Power-law exponents are 1.9 and 1.6 when fitted with least squares regression.**

$$\frac{\log P[B > t]}{\log t} \to -\alpha.$$

G. Ananthanarayanan, et al. *PACMan: Coordinated Memory Caching for Parallel Jobs*, NSDI '12

Criticality phenomenon in heavy tails

If $\lambda E[B] < 1$, $\lambda E[R]E[C] < r$, $\alpha > 1$, then, for Fair Scheduler

1. *If* $\mathbb{P}[R > r - \lambda\mathbb{E}[R]\mathbb{E}[C]] > 0$, *then*

$$\lim_{x\to\infty} \frac{\log \mathbb{P}\left[T^f > x\right]}{\log x} = \boxed{-\alpha + 1.}$$

A large job temporarily blocks the reduce queue
input rate > service rate

2. *If* $\mathbb{P}\left[R < r - \lambda\mathbb{E}[R]\mathbb{E}[C]\right] = 1$ *and* $\alpha > 3$, *then*

$$\mathbb{P}\left[T^f > x\right] \sim \mathbb{P}[B > (1-\rho)x],$$

implying

$$\lim_{x\to\infty} \frac{\log \mathbb{P}\left[T^f > x\right]}{\log x} = \boxed{-\alpha.}$$

input rate < service rate

3. *If* $1 < \alpha \le k^*/(k^*-1)$, $k^* \ge 2$ *and* $\mathbb{P}[r/(k^*-1) > R \ge r/k^*] = 1$, $C \equiv 0$, *then*

$$\lim_{x\to\infty} \frac{\log \mathbb{P}\left[T^f > x\right]}{\log x} = \boxed{-k^*(\alpha - 1)\,.}$$

k^* large jobs temporarily block the reduce queue

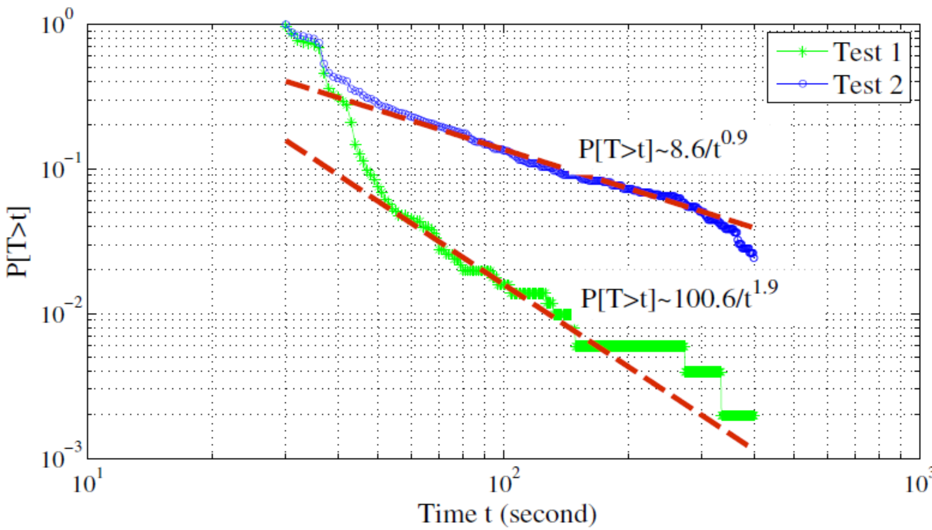
Validation of criticality

- Test bed
  - 24 nodes – 4 map + 2 reduce slots
  - Linux 22.6.18-194.17.4.el5 kernel.
  - Four 2.67GHz hex-core per node
  - Intel Xeon X5650 CPUs with Hyper-threading capability
  - 24GB memory + two 500GB Western Digital SATA hard drivers.
  - All nodes on the same Top-of-Rack 1Gigabit Ethernet switch.

Table : Composition of the job flow that is similar to the Facebook workload

Group	Benchmark	Input Size	Job (#)	ReduceTasks	
				Test-1	Test-2
1	Wordcount	64MB	330	1	1
2	Termvector	128MB	109	4	4
3	Invertedindex	256MB	36	8	18
4	Termvector	512MB	16	12	24
5	Invertedindex	1GB	5	12	32
6	Terasort	2GB	4	16	46
7	Adjancylist	4GB	3	16	46
8	Sequencecount	8GB	2	20	46
9	Sequencecount	16GB	1	20	46
		Total Jobs	506		

- Results match strikingly well with analysis
  - Each job under Test 2 runs faster than under Test 1 in a stand-alone environment
  - On contrary, job execution times under Test 2 are much worse than Test 1 in a shared environment



DynMR for IBM Platform Symphony

Performance issues

Macroscopic

- Difficulty in selecting optimal performance parameters (reduce #, MR ratio, slow-start); Auto-tuning (similar to star-fish) is also difficult
- No flow control for fetching data (cause under/over utilization of the fetch threads)
- Selfish users can monopolize the cluster by running many long reduce tasks; Unfairness to small jobs
- Unfair (YARN emphasizes fairness by assuming a single task type, MapReduce has both map and reduce)

Microscopic

- Long-tailed reduce tasks caused by data skew or heterogeneous computing nodes
- Reducer bundle several functional phases together and can only process the data of one partition; No pipelining between fetchers and mergers
- No pause-resume mechanism for ReduceTasks

How can we do better?

- **Wish list:**
  - **Preemption**
    ➔ Efficient context switching
  - **Balance Map & Reduce resources**
    ➔ Flow control

DynMR Design

- ★ **Multiple ReduceTasks in a progressive queue share a single JVM**

Guiding principle

1. **Use** fine-grained **reduce tasks (smaller partitions)**
2. **Delicately** schedule **tasks in refined time scales**
3. **Efficient task** context switching

DynMR adaptively interleaves partially-completed ReduceTasks and backfills MapTasks

When?
1. Detect-and-yield**: identify best time points to switch tasks**

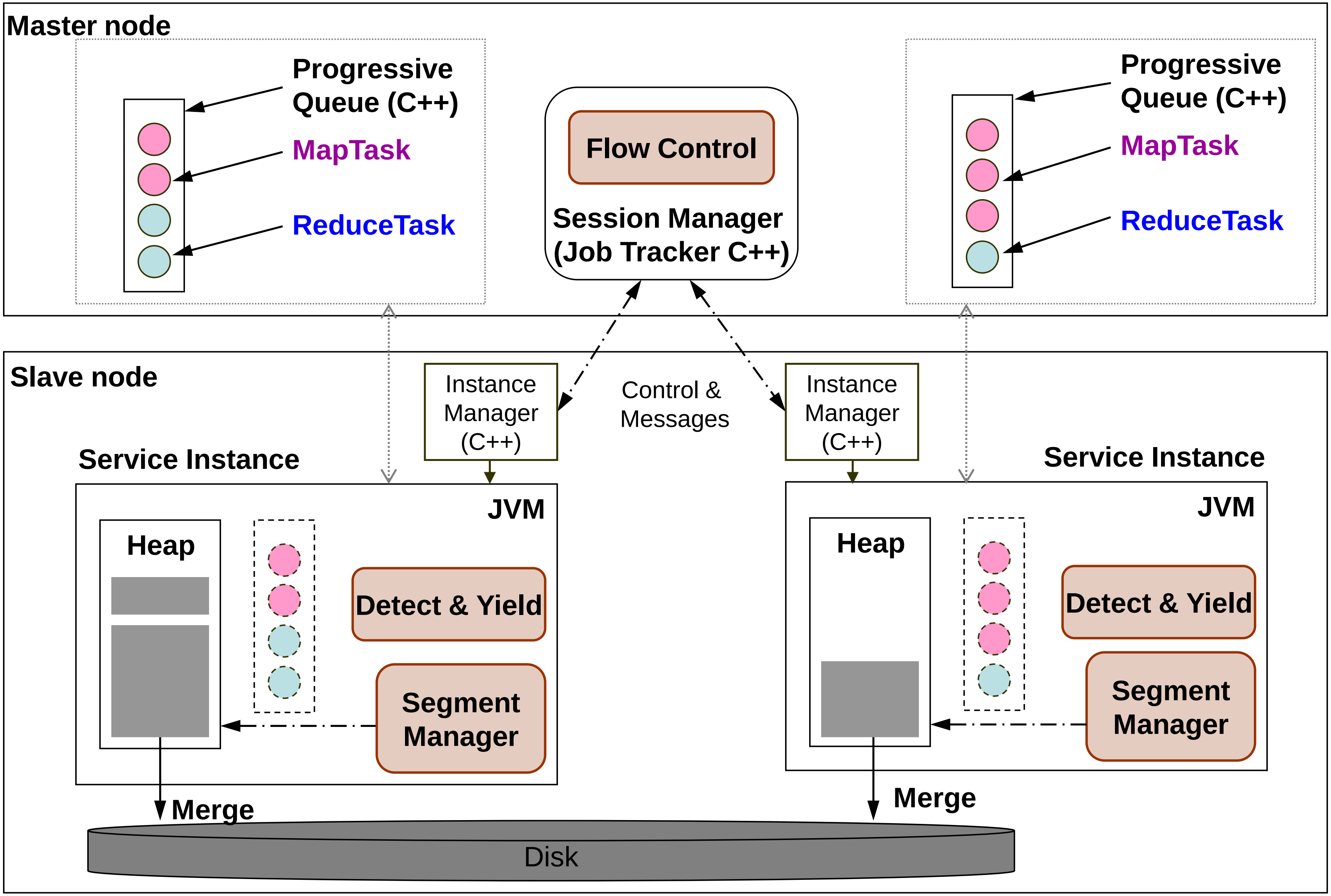
How many?

2. Flow control**: assemble multiple tasks into a progressive queue, to form a "bigger task"**

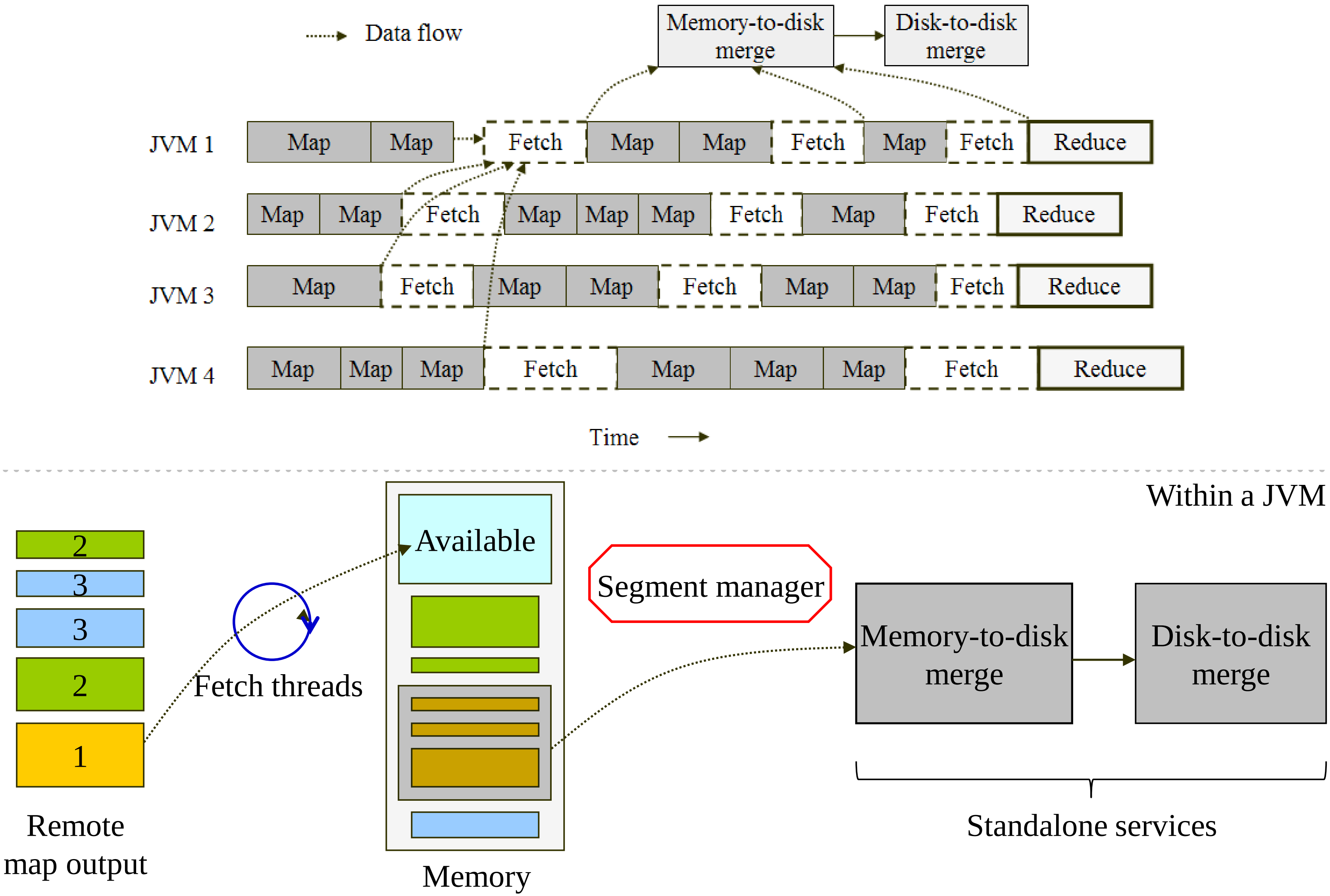
What about data?

3. Segment manager
   1. Manage data segments of multiple tasks
   2. Extract merge threads as standalone services

DynMR High Level Architecture



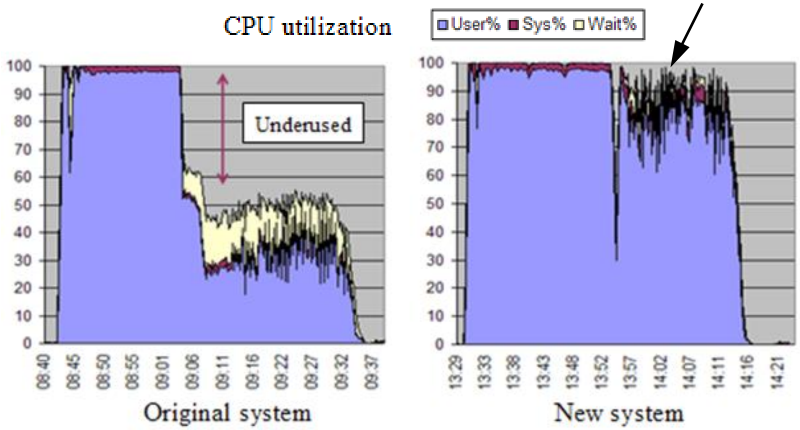
DynMR Execution Example



Experiments – Single Job

Terasort

Better utilization



Shorter execution time

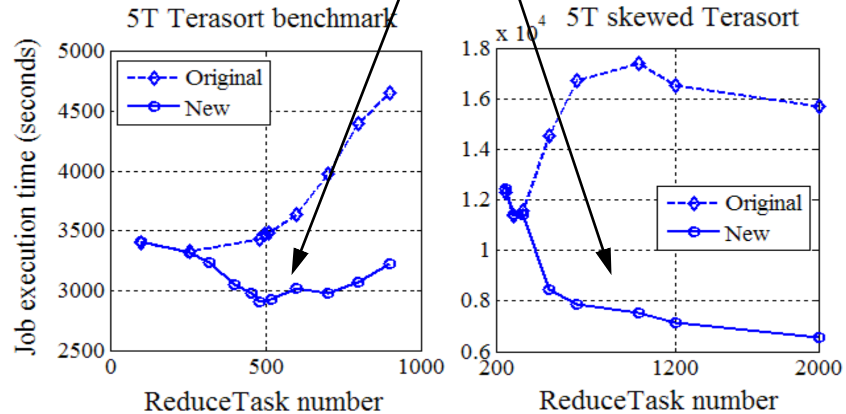


Table. 5T Terasort configurations

Parameter	Value
io.sort.mb	650
io.sort.factor	100
mapred.job.reduce.input.buffer.percent	0.96
mapred.job.shuffle.merge.percent	0.96
mapred.job.shuffle.input.buffer.percent	0.8
mapred.compress.map.output	true
mapred.map.output.compression.codec	Lz4Codec
mapred.output.compression.type	BLOCK
mapreduce.job.intermediatedata.checksum	false
JVM heap -Xmx -Xms	1000m

Table. Optimal configuration for the original system

Parameter	Value
map/reduce slot ratio	2:1
mapred.reduce.slowstart.completed.maps	0.25
mapred.reduce.tasks	256

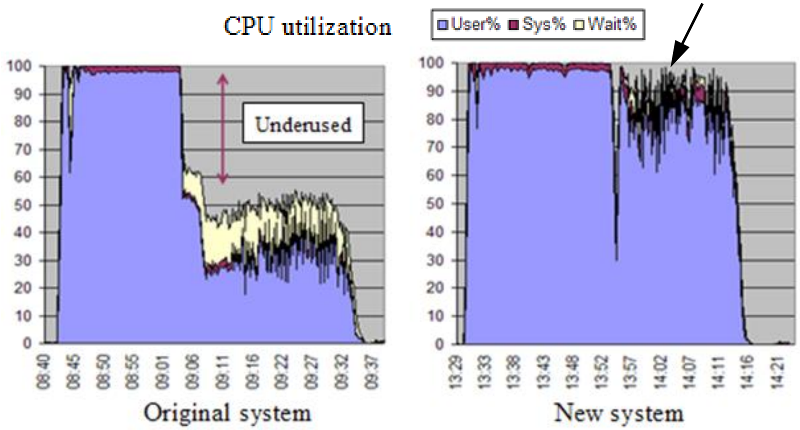
Default parameter values are: 1:1, 0.05, 384.

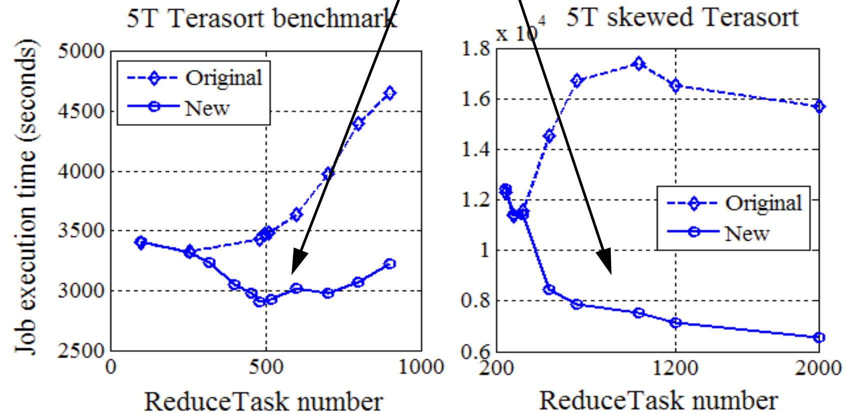
Experiments – Single Job

Terasort

Better utilization

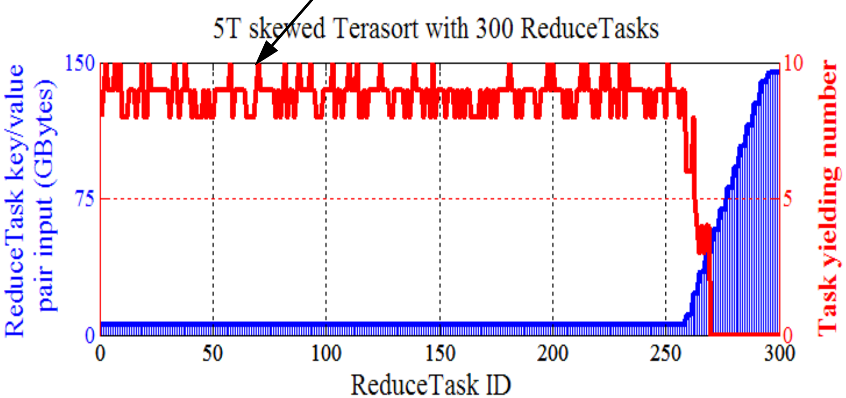
Shorter execution time



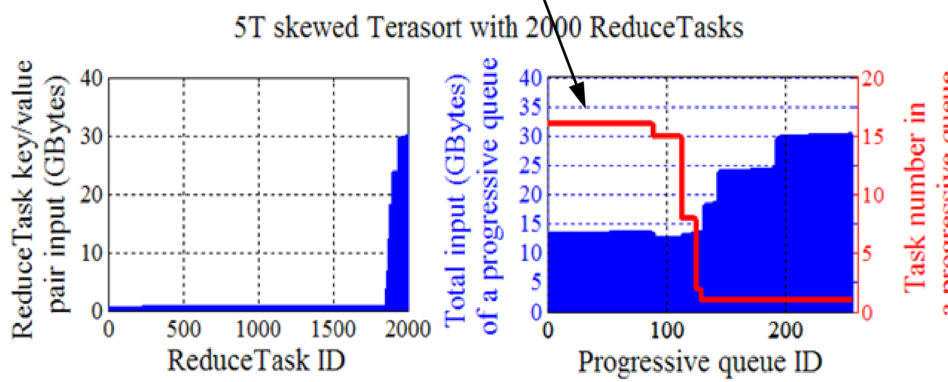


service rounds and # tasks in a progressive queue (skewed data)

Small ReduceTasks yield more often



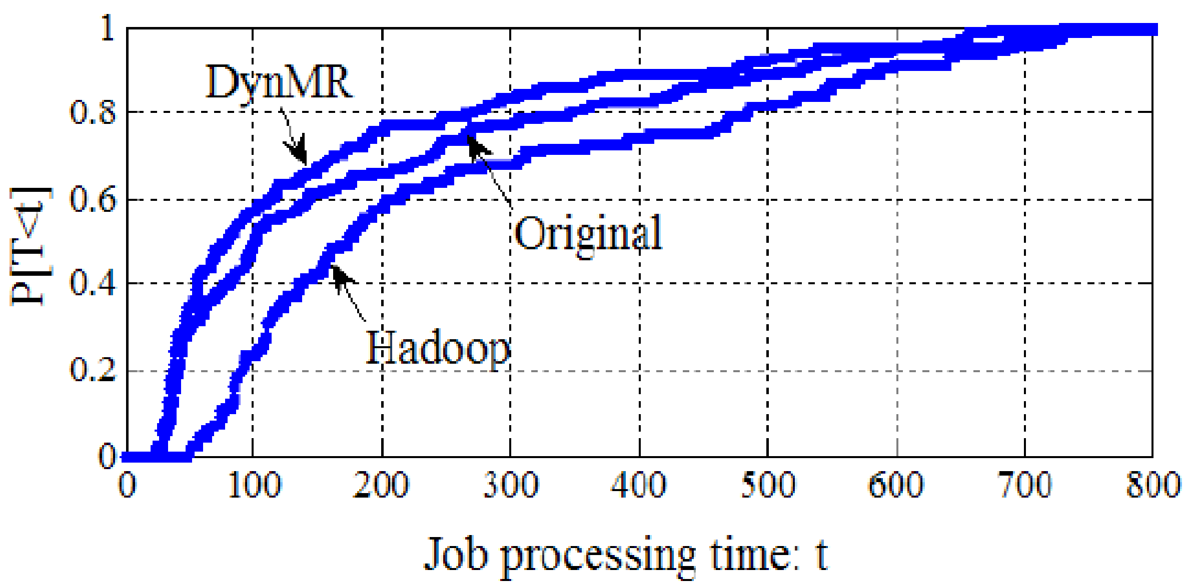
More Small ReduceTasks in 1 progressive queue



Experiment – Multiple Jobs

Table Job details

Group	Job name	Input data	Task # (m,r)	Job #
1	Histogram-movies	9.7GB	(75,10)	60
2	Grep '$\wedge a[a-z]*$'	31.1GB	(235,15)	40
3	Histogram-ratings	46.5GB	(359,20)	30
4	Inverted-index	52.3GB	(390,25)	25
5	Word-count	66.1GB	(498,30)	10
6	Sequence-count	99.3GB	(749,90)	3
7	Adjancy-list	29.1GB	(313,100)	2



Summary

- **MapReduce systems**
- **Starvation problem**
- **Solutions:**
  - **Coupling scheduler**
  - **Preemption mechanism**
  - **DynMR**
- **Performance Models**
  - **Overlapping Tandem Queue**
  - **Online scheduling**
  - **Heavy tail analysis**

Research Activities on MapReduce

Deeper Changes

- **Coupling scheduler for Hadoop**
  - Infocom 2012, Sigmetrics 2012, Infocom 2013, Cloud 2013
- **Pre-emptive scheduler for Hadoop**
  - Efficient pause/resume mechanism for Reduce tasks
  - Improves fairness & reduces execution time
  - ICAC 2013, Sigmetrics 2014
- **DynMR: dynamic task interleave for Symphony**
  - Interleave Map and Reduce tasks for Platform Symphony product
  - Improve Terasort speed to 1.7x
  - EuroSys 2014
- **Online parameter tuning**
  - Adjust parameter settings (cpu, memory) online for better performance
  - HPDC 2014
- **Performance modeling & capacity planning**
  - Overlapping tandem queue model & algorithm analysis
  - IFIP Performance 2013
  - Capacity planning: given a target execution time, determine number & type of VMs, number of reduce tasks per VM

Thank You!