Improving Resource Utilization by Timely Fine-Grained Scheduling

Tatiana Jin, Zhenkun Cai, Boyang Li,
Chenguang Zheng, Guanxian Jiang, James Cheng

Department of Computer Science and Engineering
The Chinese University of Hong Kong

Core Problem

Central Idea

System: Ursa

Experimental Evaluation

Core Problem

Cluster Resource Utilization

- Scheduling Efficiency
- Utilization Efficiency

Cluster Resource Utilization

hadoop YARN

kubernetes

Sparrow

Apollo

MESOS

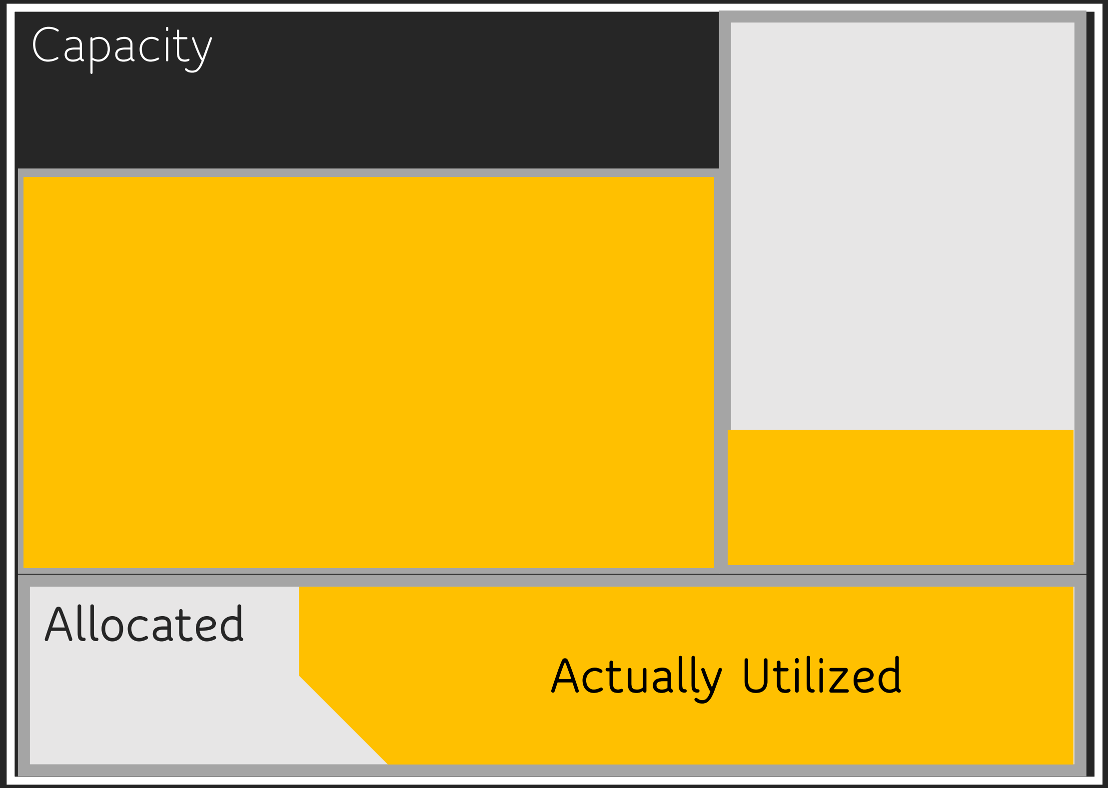
Borg

Mercury

Scheduling Efficiency and Utilization Efficiency

Scheduling Efficiency (SE)

Utilization Efficiency (UE)

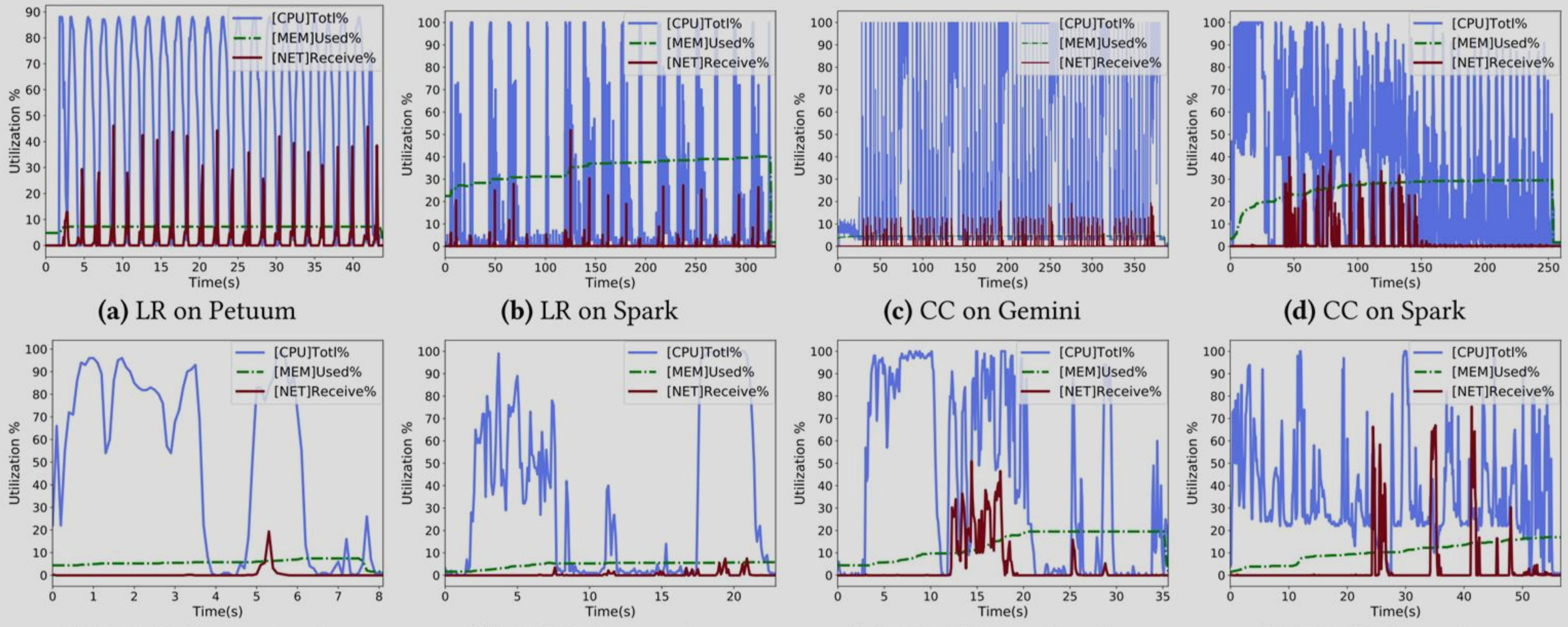


Application Scenario

Project → Quota Group → Virtual Cluster

- Workload: 70% OLAP, 20% machine learning and 10% graph analytics
- Performance Objective
  1. Maximize job throughput (minimize makespan)
  2. Minimize average job completion time (JCT) (time from submission to completion)

Dynamic Resource Utilization Pattern



(a) LR on Petuum **(b)** LR on Spark **(c)** CC on Gemini **(d)** CC on Spark

(e) TPC-H Q14 on Spark **(f)** TPC-H Q14 on Tez **(g)** TPC-H Q8 on Spark **(h)** TPC-H Q8 on Tez

Figure 1. Resource utilization of different workloads (best viewed in color)

Central Idea

Ursa: achieving high SE and UE by fine-grained, dynamic, load-balanced resource negotiation

Design Objectives

SE

Obj-3. Load-balanced task assignment

Obj-4. Low-latency resource scheduling

UE

Obj-1. Accurate resource request

Obj-2. Timely provision and release of resource

Using Monotask to Handle Dynamic Patterns

- Monotask* is a unit of work that uses only a single type of resource (e.g. CPU, network bandwidth, disk I/O) apart from memory
- Introduced for job performance reasoning
- A unit of execution with **steady and predictable** resource utilization

Container		Dataflow Tasks
Resource-oriented, execution-agnostic	Monotask	Execution-oriented, resource-agnostic
Scheduling	Ursa	Execution

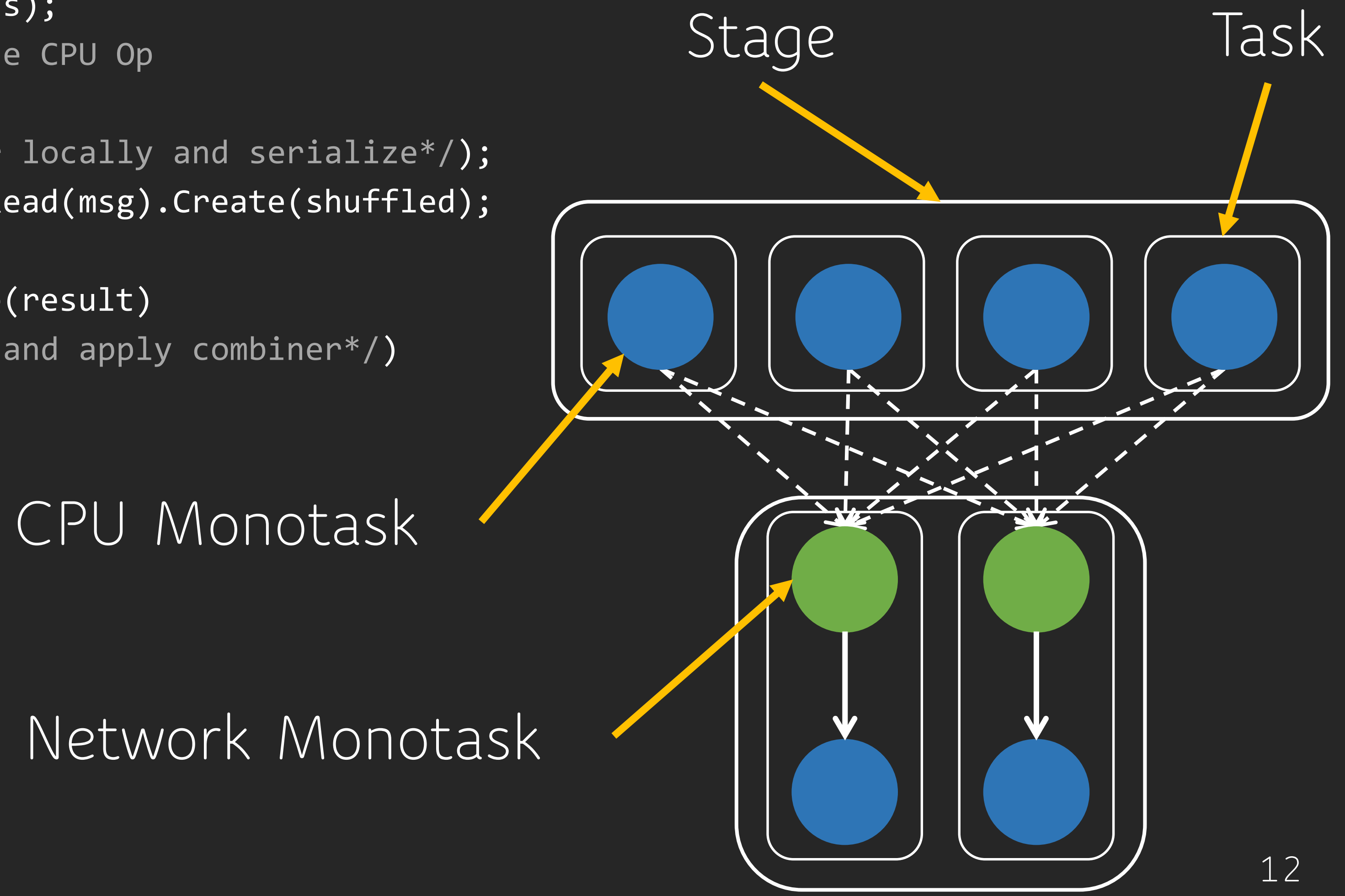
* Kay Ousterhout, Christopher Canel, Sylvia Ratnasamy, and Scott Shenker. 2017. Monotasks: Architecting for performance clarity in data analytics frameworks. In Proceedings ofthe 26th ACMSymposium on Operating Systems Principles (SOSP 17). ACM, 184–200.

System: Ursa

A scheduling and execution framework

API and Monotask Generation

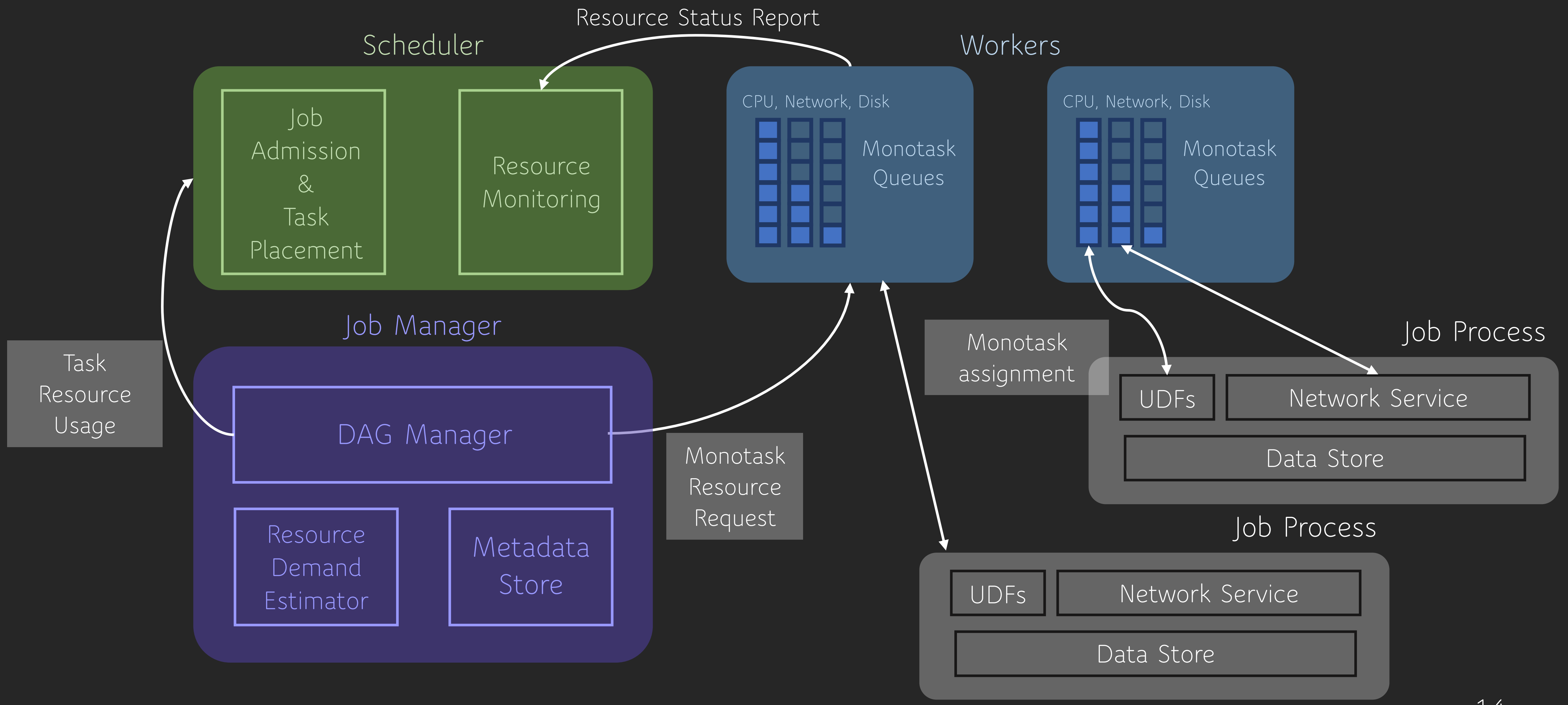
```cpp
template <typename ValueType>
class Dataset { // ...
 auto ReduceByKey(Combiner combiner, int partitions) {
   auto msg = dag.CreateData(this->partitions);
   auto shuffled = dag.CreateData(partitions);
   auto result = dag.CreateData(partitions);
   auto ser = dag.CreateOp(CPU)  // create CPU Op
                 .Read(this).Create(msg)
                 .SetUDF(/*apply combiner locally and serialize*/);
   auto shuffle = dag.CreateOp(Network).Read(msg).Create(shuffled);
   auto deser = dag.CreateOp(CPU)
                  .Read(shuffled).Create(result)
                  .SetUDF(/*deserialize and apply combiner*/)
   this->creator.To(ser, ASYNC);
   ser.To(shuffle, SYNC);
   shuffle.To(deser, ASYNC);
   return result;
 }
 // ...
 OpGraph dag;
 Op creator;
 int partitions;
};
```



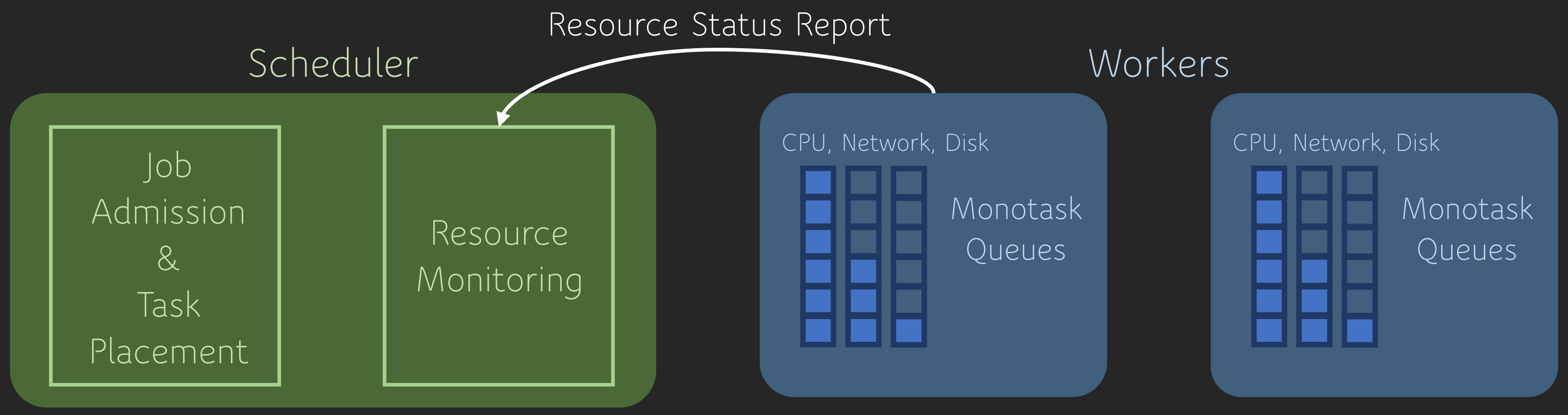
High-Level APIs

- SQL (connected to Hive)
- Spark-like dataset transformations
- Pregel-like vertex-centric interface

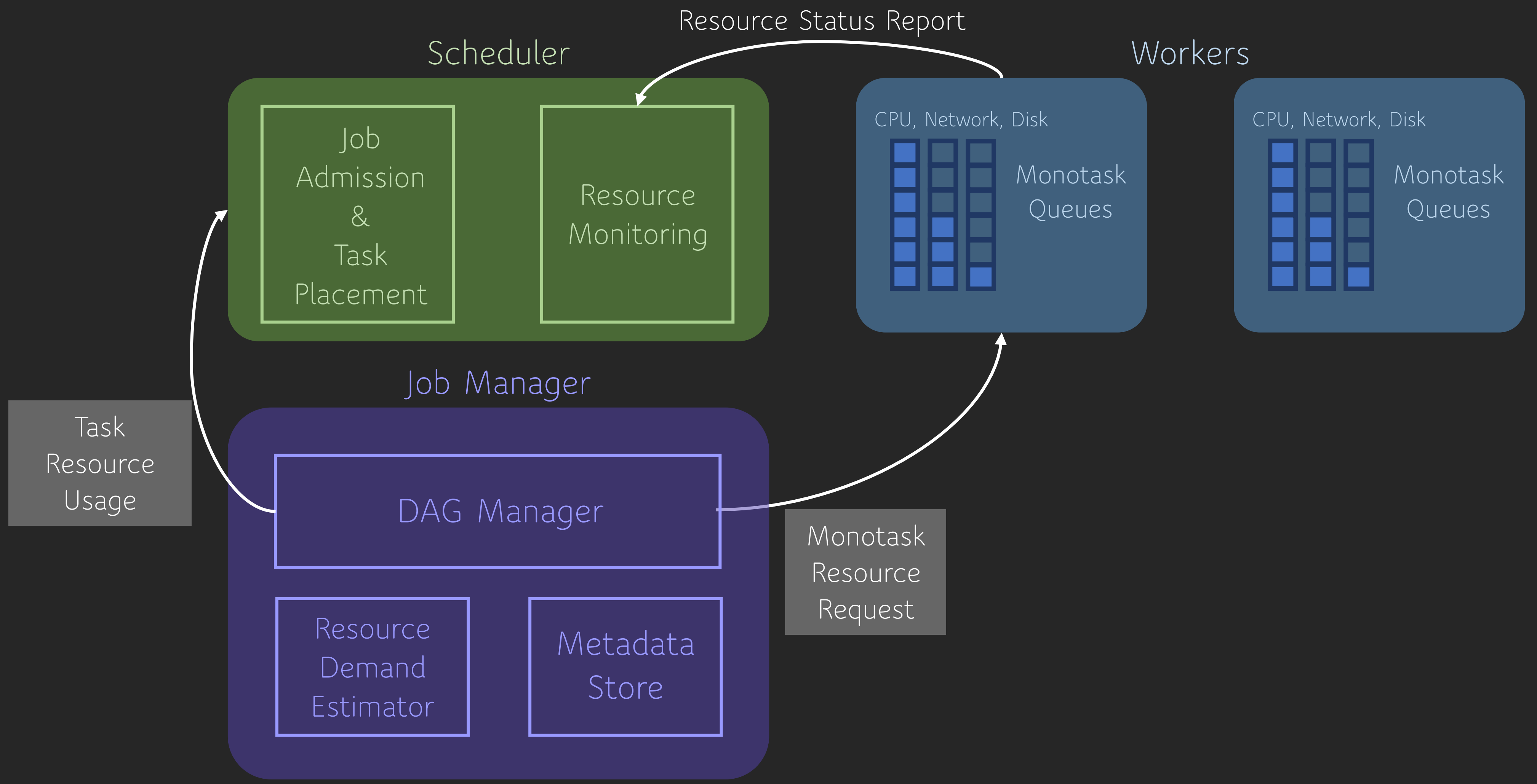
System Overview



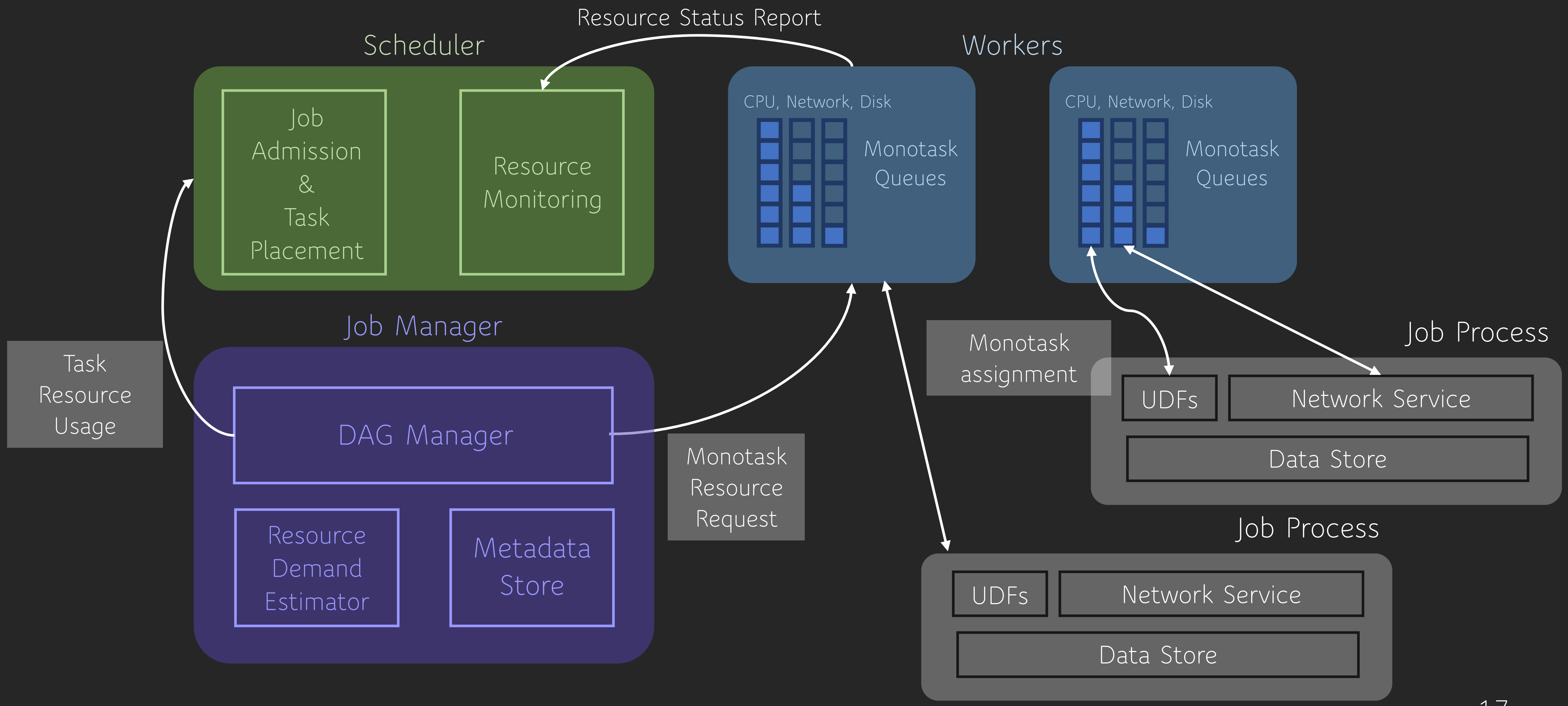
System Overview



System Overview



System Overview



Task placement

- Resource usage estimation
  - The CPU, network and disk I/O usage is estimated on a monotask basis
    - The execution layer is designed to guarantee **stable** resource utilization by each type of monotasks during their execution
  - The memory usage is estimated on a task basis
    - The memory usage during the execution of a task is relatively **stable**

In contrast to simply using coarse-grained (historical) peak resource demands, monotask-based resource estimation allows

per-resource needs to be captured dynamically at runtime

Task placement

- Stage-aware load-balanced task placement
  - A unified measure for multi-dimensional resource consumption
  - Total resource consumption in contrast to the peak demands of tasks
  - Stage-aware task placement to avoid stragglers due to scheduling delay

Task placement

- Stage-aware load-balanced task placement
  - Approximate Processing Time (APT_r)

    =(Total input data size of assigned type−r monotasks) / (Processing rate)
    - APT_r tells when resource-r on a worker will become idle
    - Per-resource processing rates on each worker are periodically updated to the scheduler
  - Expected Processing Time (EPT)
    - EPT is an indicator of whether a worker is over-loaded or under-loaded
    - Set to slightly larger than the scheduling interval

Task placement

From APT and EPT, we can compute

- Difference between EPT and APT for resource r at worker w as

$$D_r(w) = \max(0, \frac{EPT - APT_r(w)}{EPT})$$

- The increase in the load of worker w in using resource r if task t is placed in w as $Inc_r(t, w)$
- Task placement score as a dot product

$$F(t, w) = \sum_{r \in \{CPU, network, disk, mem\}} D_r(w) \times Inc_r(t, w)$$

Pick more lightly-loaded workers

Pick tasks with heavier load (harder to place)

Task Placement

- Stage-awareness
  - Each schedule decision is a plan with tasks in the same stage instead of with a single task
  - Ranking plans by stage-average scores
  - A large bonus is given to a plan if the plan assigns all tasks in stage S, so that such plans are always considered before other plans

Other Scheduling Details

- Supporting scheduling policies
  - Earliest Job First (EJF) and Smallest Remaining Job First (SRJF)
  - Job ordering at the scheduler and monotask ordering at distributed queues

- Concurrency control
  - Avoid resource contention among running monotasks
  - Maintain high utilization of resource

Experimental Evaluation

Settings

- Workloads
  - OLAP: TPC-H and TPC-DS
  - Mixed: 70% OLAP, 20% machine learning and 10% graph analytics (ratio by total CPU usage)

- A cluster of 20 machines connected by 10 Gbps Ethernet
  - Resembles a small cluster requested by a quota group

Limitations of using coarse-grained containers

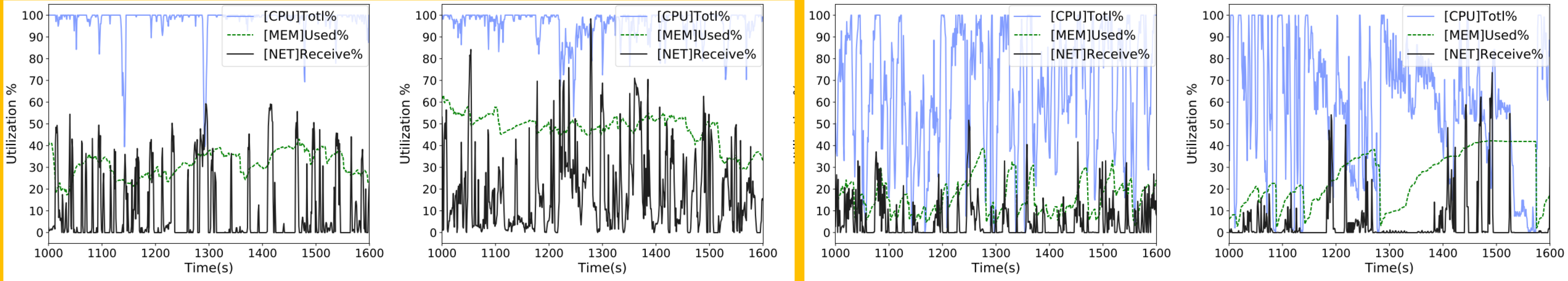
Performance on TPC-H

	makespan	avgJCT	UE_{cpu}	SE_{cpu}	UE_{mem}	SE_{mem}
EJF	2803	600.00	99.64	92.47	78.83	39.80
SRJF	2859	489.96	99.65	89.73	78.02	48.85
YARN+Spark	3849	1407.40	69.35	93.32	34.69	44.13
YARN+Tez	9228	4287.00	58.97	98.19	28.81	70.71

Performance on TPC-DS

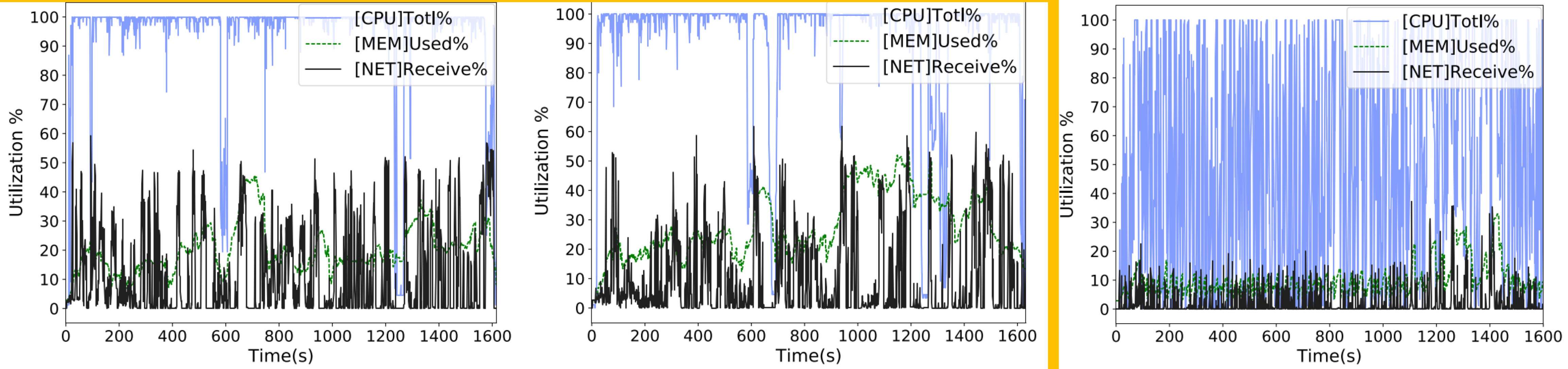
	makespan	avgJCT	UE_{cpu}	SE_{cpu}	UE_{mem}	SE_{mem}
EJF	1613	453.20	99.57	88.31	81.64	25.01
SRJF	1630	242.27	99.75	86.99	85.83	32.93
YARN+Spark	2927	894.36	48.56	90.48	19.39	37.65

TPC-H



(a) Ursa-EJF **(b)** Ursa-SRJF **(c)** Y+S **(d)** Y+T

Limitations of using coarse-grained containers



(a) Ursa-EJF **(b)** Ursa-SRJF **(c)** Y+S

TPC-DS

Compare with Alternative Approaches

Performance on Mixed

	makespan	avgJCT	UE_{cpu}	SE_{cpu}
Ursa-EJF	464.00	208.21	99.57	86.60
Ursa-SRJF	473.50	170.64	98.89	86.08
YARN+Ursa	842.92	443.80	44.15	89.97
YARN+Spark	1072.66	435.00	67.92	83.84
Capacity	511.00	226.16	99.77	78.66
Tetris	562.33	254.52	98.62	70.02
Tetris2	506.00	240.83	99.71	79.75

Subscription ratio	makespan (YARN+Ursa)	avgJCT (YARN+Ursa)	makespan (YARN+Spark)	avgJCT (YARN+Spark)
1	842.92	443.80	1072.66	435.00
2	637.96	345.99	872.67	341.77
4	596.66	325.32	892.83	365.30

Using monotasks alone

Using other scheduling algorithms

Over-subscription of CPU

Conclusions

Ursa:

- A framework for both resource scheduling and job execution
- Handles jobs with frequent fluctuations in resource usage
- Captures dynamic resource needs at runtime and enables fine-grained, timely scheduling
- Achieves high resource utilization, which is translated into significantly improved makespan and average JCT

Thank You

Contact: Tatiana Jin (tjin@cse.cuhk.edu.hk)