INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

ICPP/2020/EDMONTON/CANADA

AUGUST 17-20, 2020

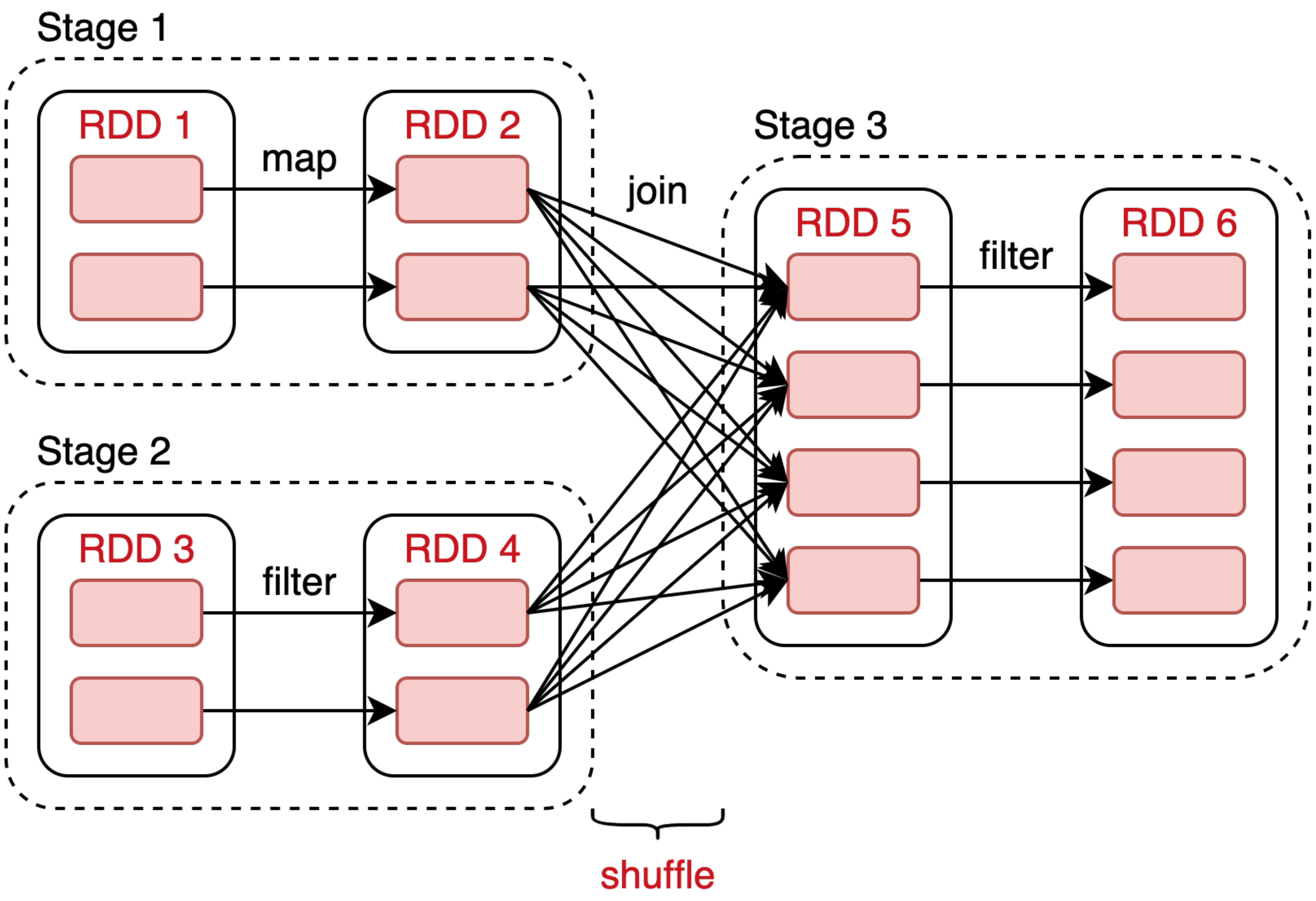
OPS: Optimized Shuffle Management System for Apache Spark

Yuchen Cheng*, Chunghsuan Wu*, Yanqiang Liu*, Rui Ren*, Hong Xu†, Bin Yang‡, Zhengwei Qi*

* Shanghai Jiao Tong University
† City University of Hong Kong
‡ Intel Corporation

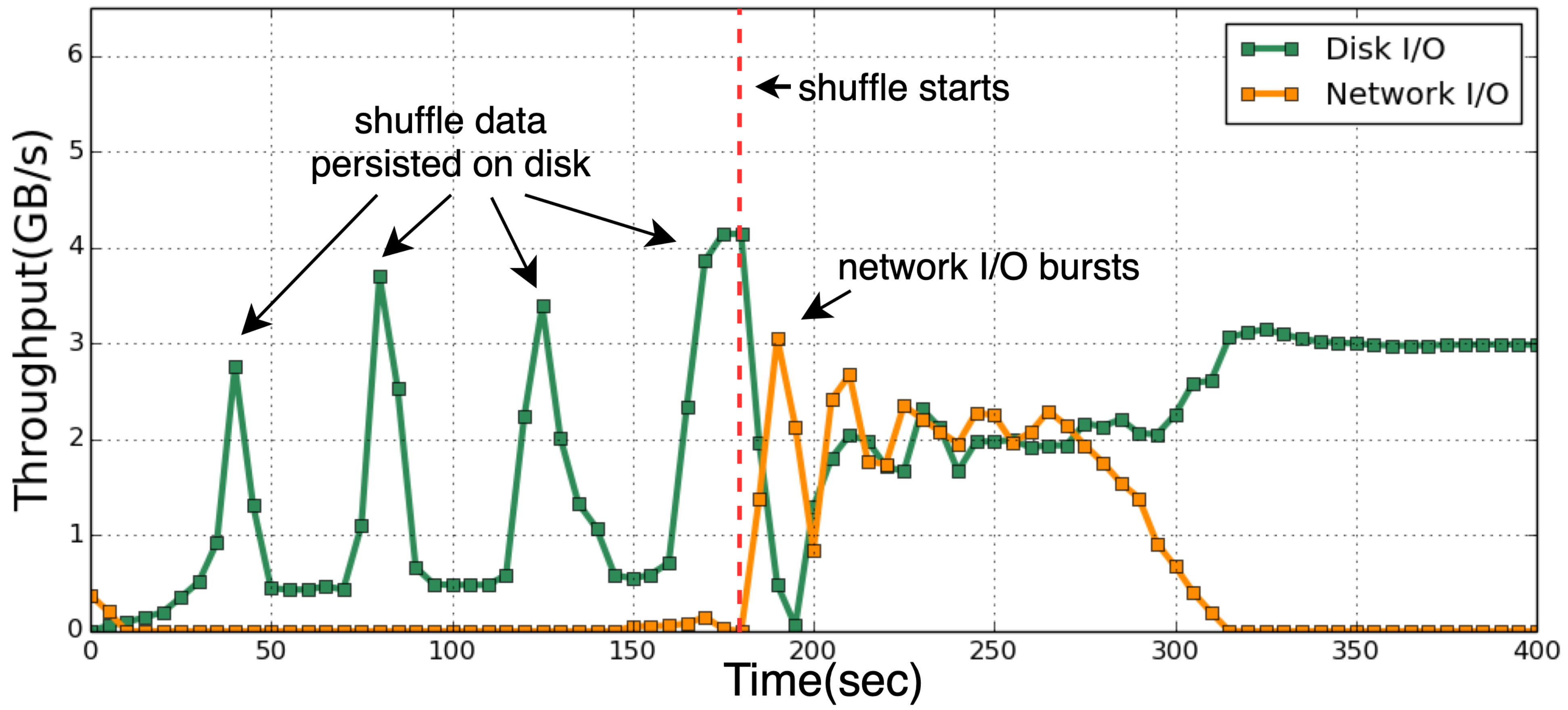
SHANGHAI JIAO TONG UNIVERSITY

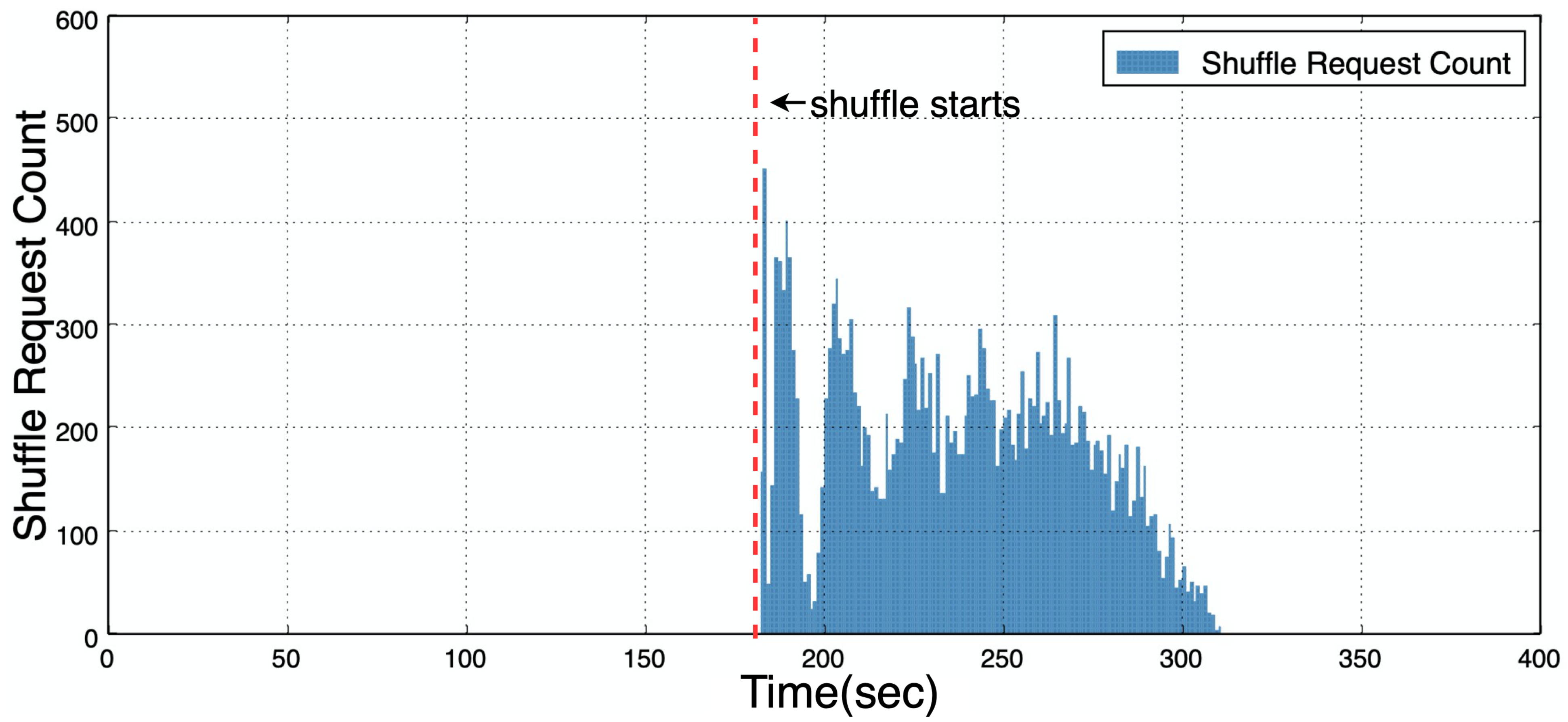
Data Processing in Spark



Dependent Shuffle Phase

- Map phase
  - intensive disk I/O for persisted shuffle data
  - idled network I/O resources
- Reduce phase
  - network I/O peaks
  - shuffle request peaks with a significant trough
- Observations
  - the resource slot-based scheduling method that does not consider I/O resources
  - the calculation logic that couples data transmission and calculation



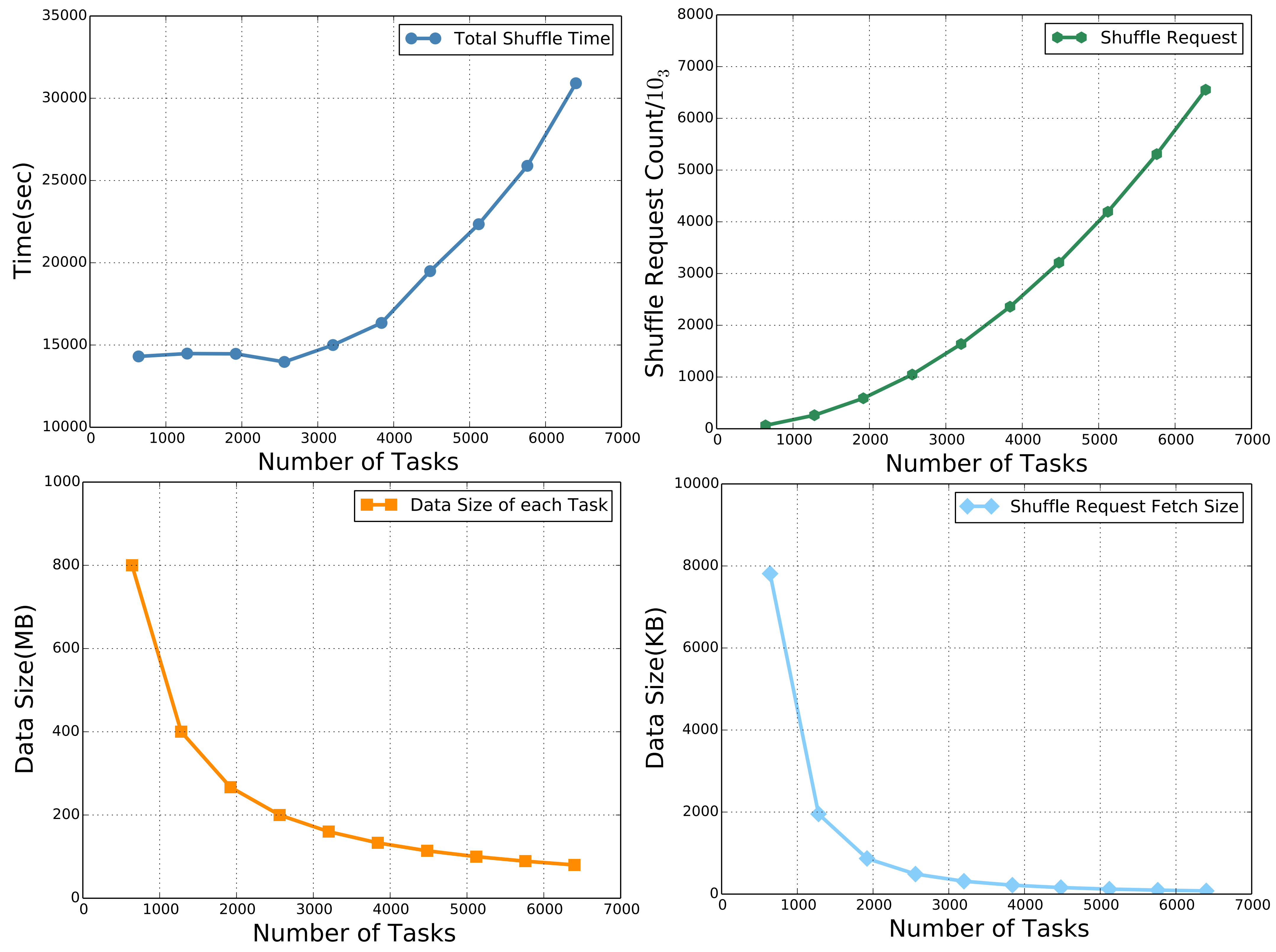


Multi-Round Sub-Tasks

- The number of sub-tasks is recommended to be at least twice the total number of CPUs in the cluster
- However, the intermediate data in this phase cannot be transmitted in time except the last round
  - Stragglers ☠️

Overhead of Shuffle Phase

- 512 GB two-stage sequencing application
- 640 to 6400 sub-tasks
- As the number of sub-tasks increases,
  - the total execution time of the shuffle phase increases sharply
  - the number of shuffle requests grows to the power of the original
  - the amount of transmission for each shuffle request also gradually decreases



Optimizations: I/O Requests

- Sailfish [SoCC ’12]
  - Aggregate intermediate data files and using batch processing
  - Modification of the file system is needed
- Riffle [EuroSys ’18]
  - Efficiently schedule merge operations
  - Convert small, random shuffle I/O requests into much fewer large, sequential I/O requests
  - Intensive network I/O

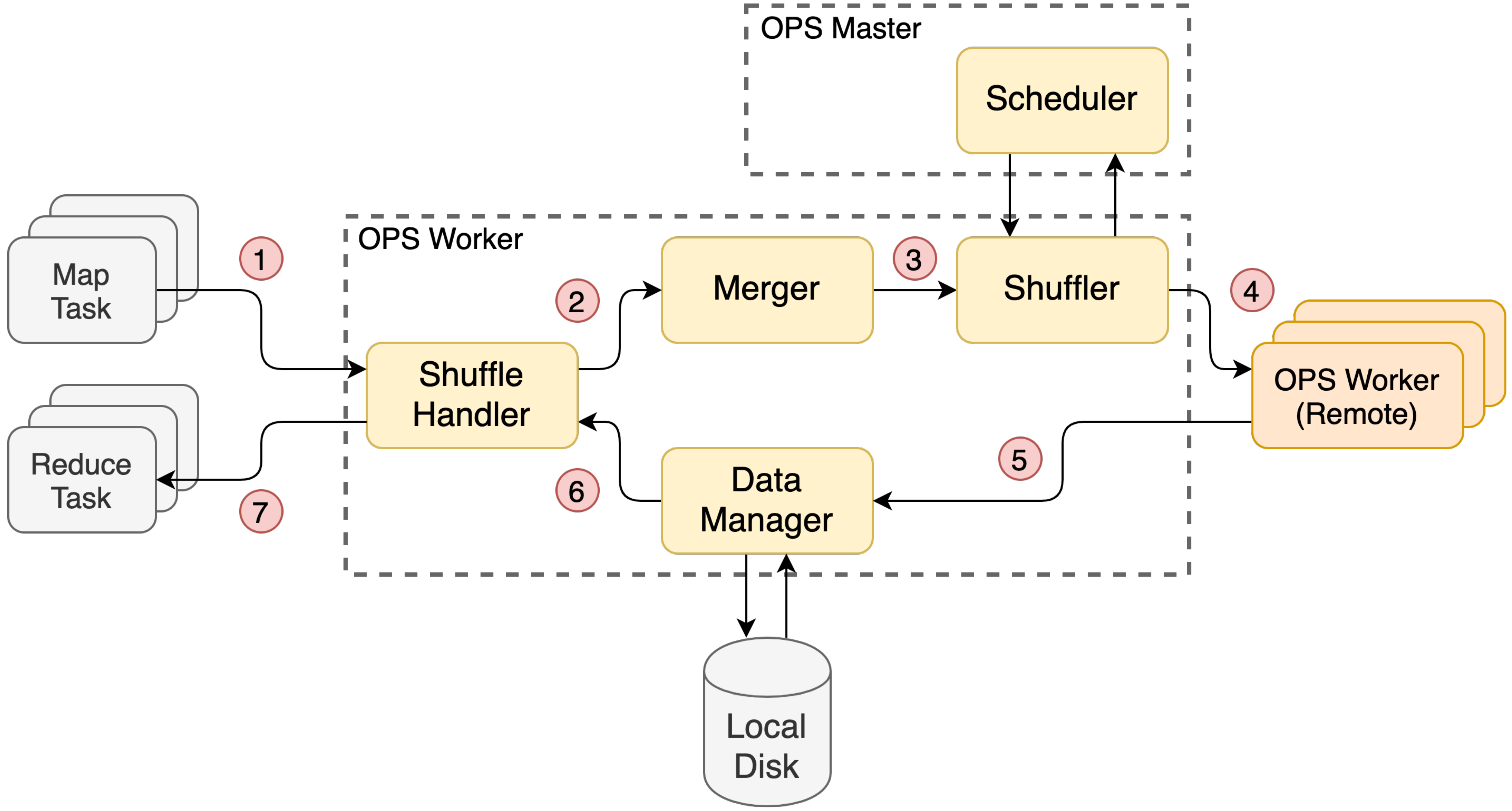
Optimizations: Shuffle Optimization

- iShuffle [TPDS, 2017]
  - Separate the shuffle phase from the reduce sub-tasks
  - Low utilization of I/O resources (e.g., disk and network)
- SCache [PPoPP ’18]
  - In-memory shuffle management with pre-scheduling
  - Lack the support of larger-than-memory datasets

Our Goal

- In-memory shuffle management with larger-than-memory datasets support
- Elimination of synchronization barrier
- Utilization of I/O resources
- Mitigation of the number of shuffle requests

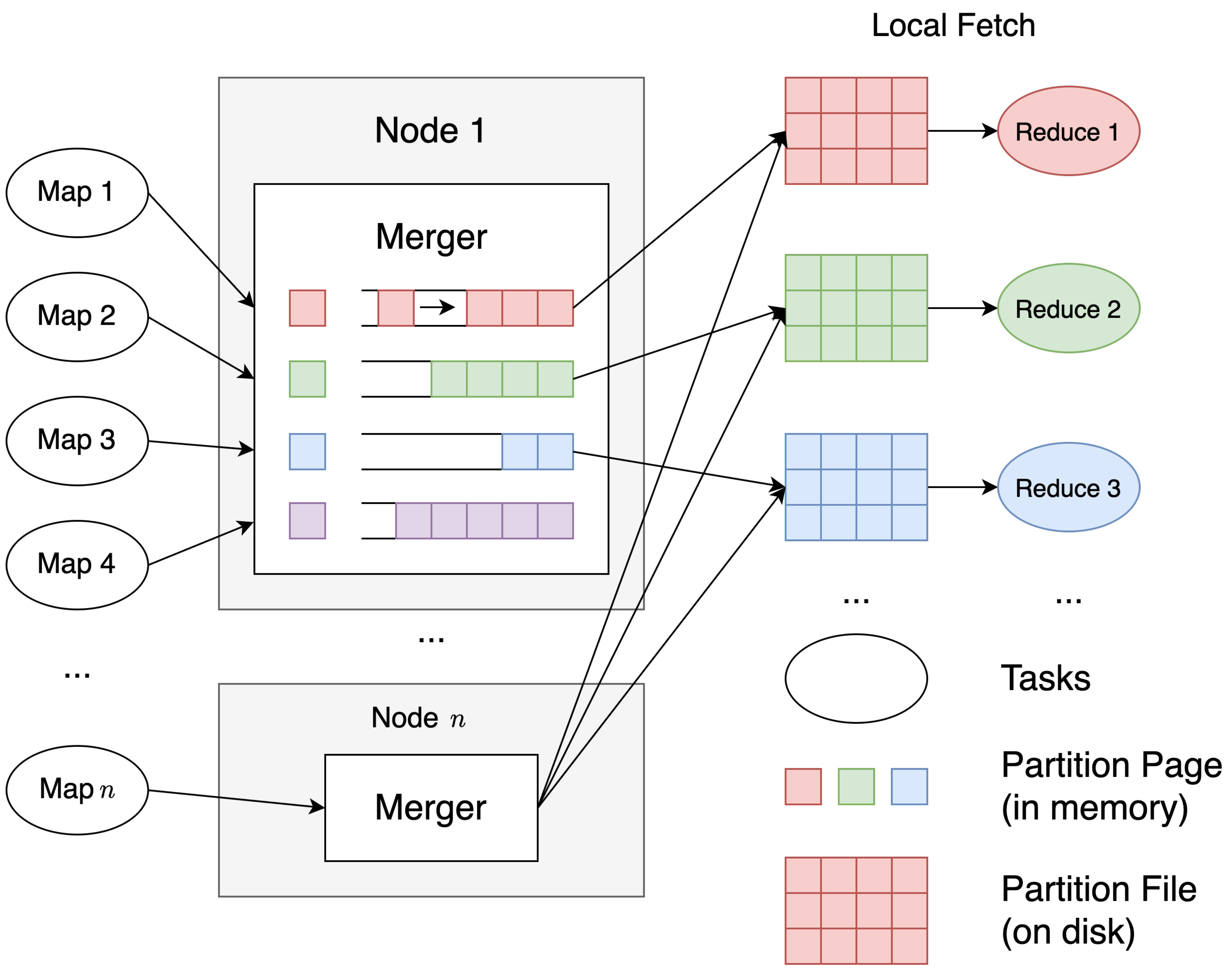
Proposed Design: OPS



- Early-merge phase: Step 1 and 2
- Early-shuffle phase: Step 3, 4 and 5
- Local-fetch phase: Step 6 and 7

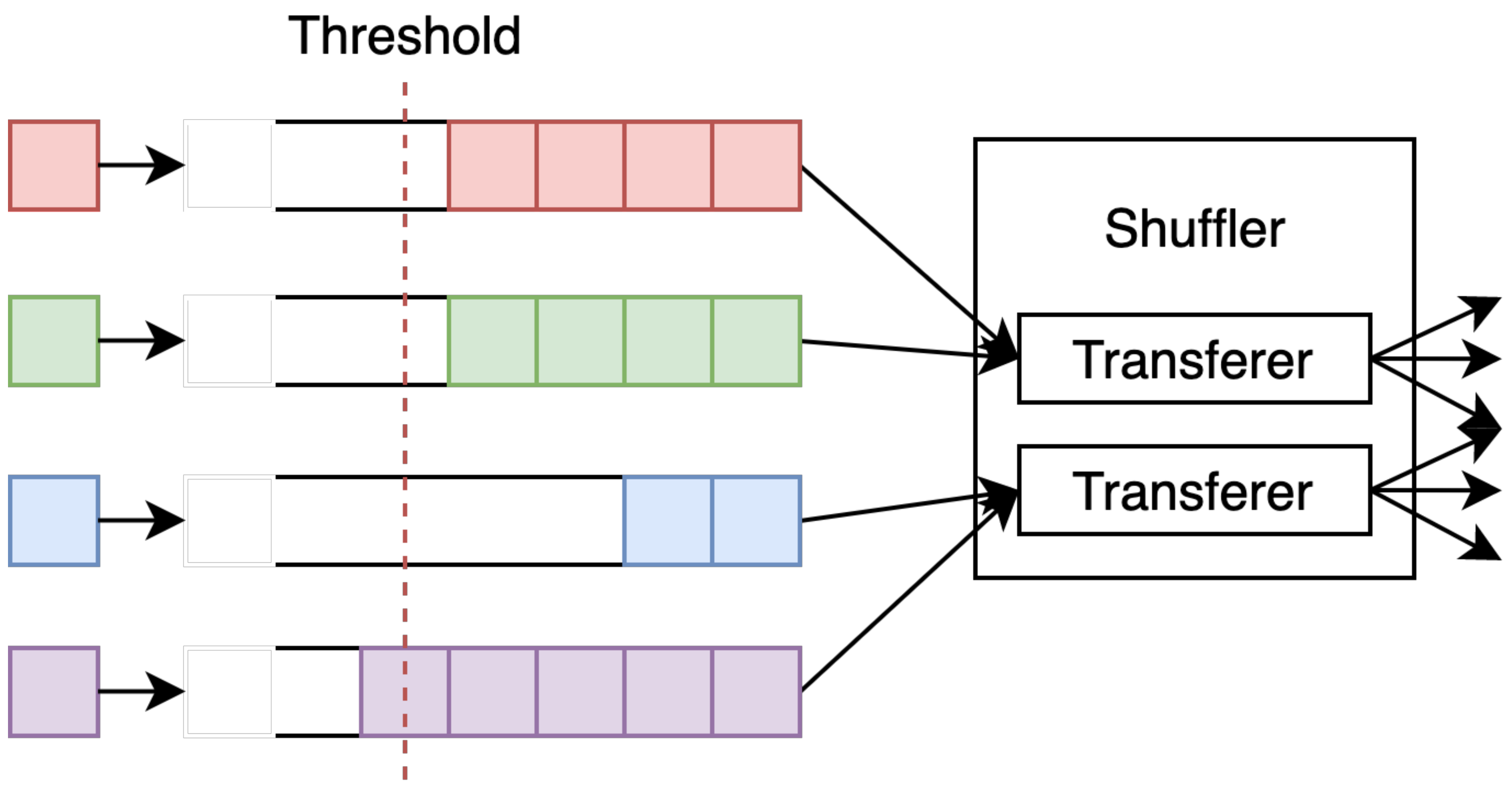
Early-Merge

1. The raw output data of the map sub-tasks is directly transferred to OPS
2. Intermediate data is temporarily stored in memory and transferred to the disk of the designated node
3. OPS releases memory resources after the early-shuffling of the partition page is completed



Early-Shuffle

- `Transferer` reads the partition pages in different partition queues in turn for transmission as a consumer
  - until all corresponding partition queues are empty
- Threshold can be set according to bandwidth and memory size of the cluster



Early-Schedule

- The execution of the early-shuffle strategy of OPS depends on the scheduling results of the reduce sub-tasks
- OPS is designed to trigger early-schedule in two cases:
  - when the first early-shuffle is triggered
  - when the number of completed map sub-tasks reaches the set threshold (5% as default)

Testbed

- 100 `t3.xlarge` EC2 nodes with a 4-core CPU and 16 GB of memory
- Hadoop YARN v2.8.5 and Spark v2.4.3
- 10 GB of memory is allocated for early-merging

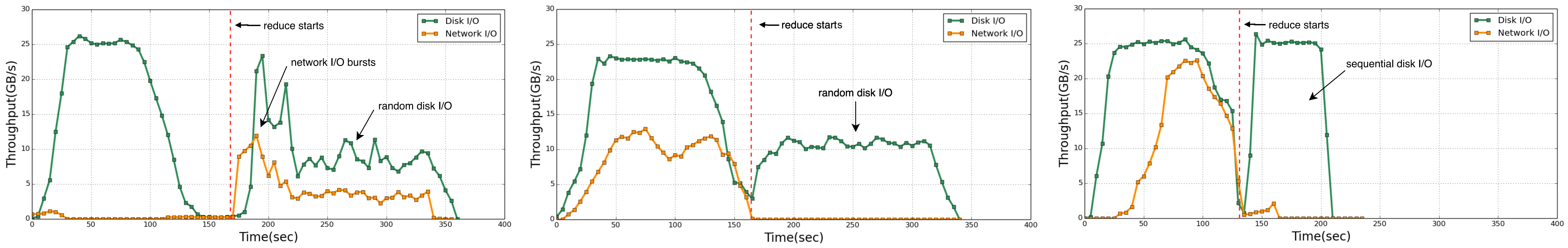
Metric	Value
CPU	3.1 GHz Intel Xeon Platinum 8000 series (Skylake-SP or Cascade Lake)
vCPU	4
Memory	16 GB
Storage	AWS EBS SSD (gp2) 256 GB
Storage IOPS	750
Storage Bandwidth	250 Mbps
Network Bandwidth	4.8 Gbps
OS	Amazon Linux 2

Workload

- `Sort` application with 1.6 TB of random text

	Input Splits	Partition	Rounds	Data / Task
1	1600	1600	4	1000 MB
2	2400	2400	6	670 MB
3	3200	3200	8	500 MB
4	4000	4000	10	400 MB
5	4800	4800	12	330 MB
6	5600	5600	14	290 MB
7	6400	6400	16	250 MB

I/O Throughput

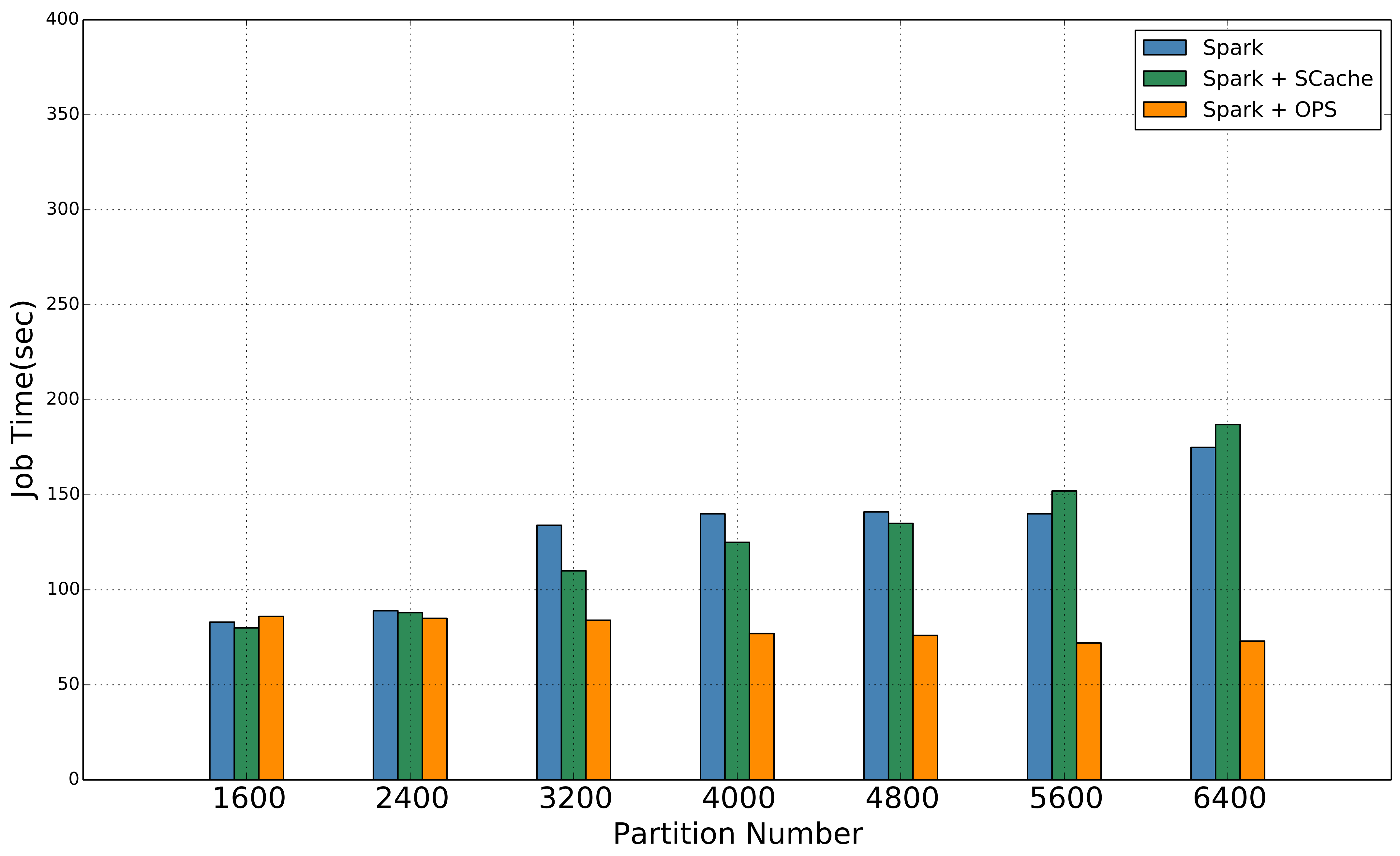


Spark | Spark + SCache | **Spark + OPS**

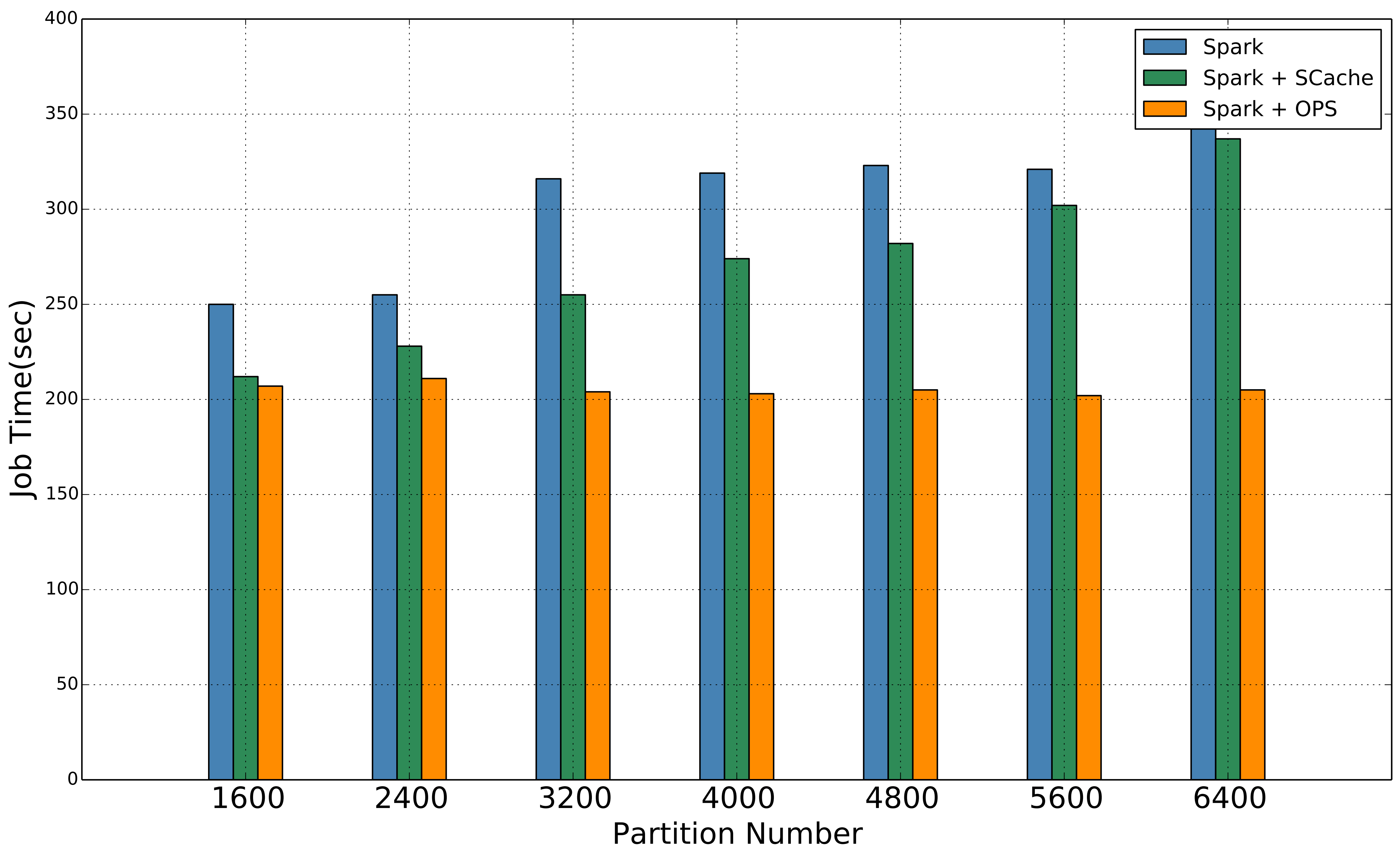
- OPS optimizes the total execution time by about 41%, and the execution time of reduce by about 50%
- Higher utilization of network I/O in the map phase
- Higher utilization of disk I/O in the reduce phase

Completion Time

Reduce

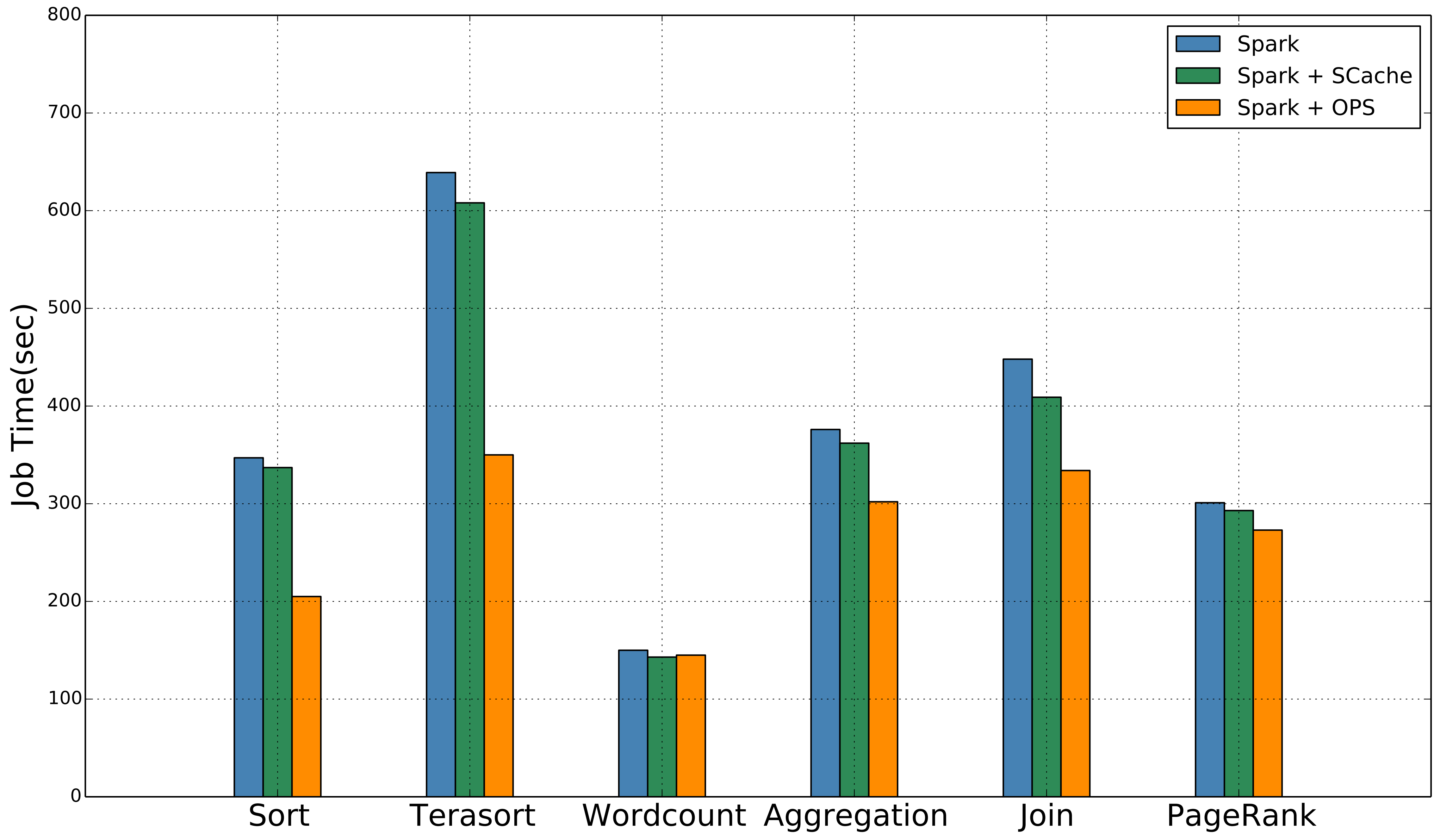


Total



- OPS reduces the total completion time by 17%-41%
- The completion time of the map phase is also steadily reduced

HiBench



- OPS outperforms in shuffle-intensive workload
  - e.g., `Sort` and `TeraSort`

Summary

- Early-merge intermediate data to mitigate intensive disk I/O
- Early-schedule based on partition pages
- Early-shuffle intermediate data stored in shared memory
- Optimize the overhead of shuffle by nearly 50%

Thanks for your attention.

Yuchen Cheng
Shanghai Jiao Tong University
rudeigerc@sjtu.edu.cn