SJTU SAIL(Software Architecture and Infrastructure Lab)

SHANGHAI JIAO TONG UNIVERSITY

Batch Processing

Xu Zhenglin

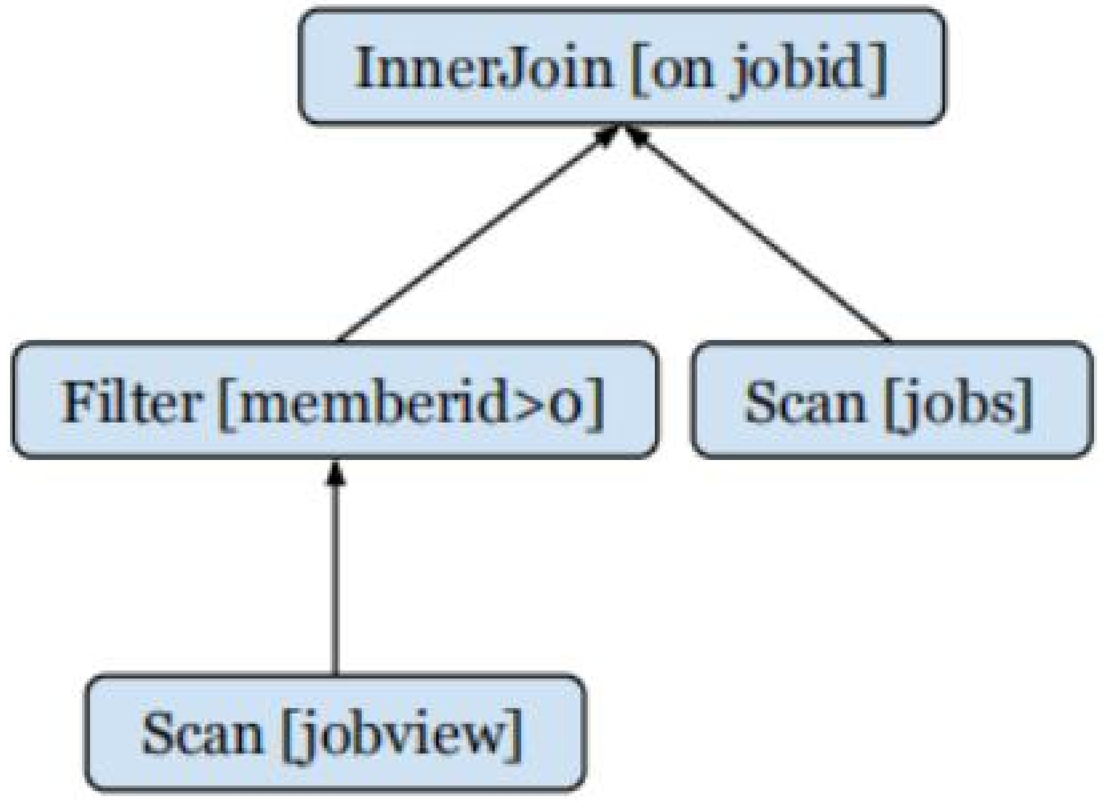
2020-10-07

Magnet

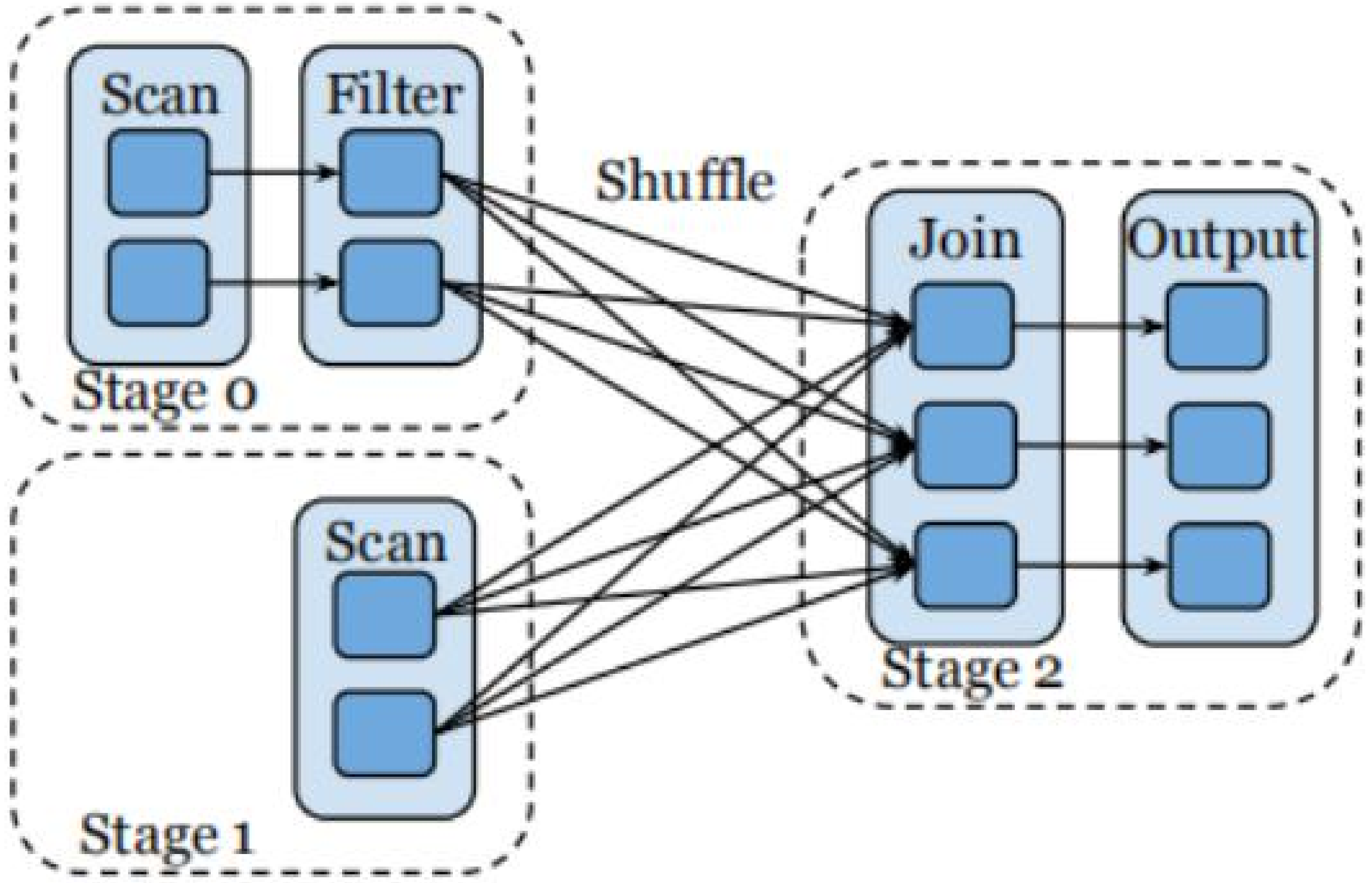
Push-based Shuffle Service for Large-scale Data Processing

Distributed data proessing frameworks

- Hadoop
- Spark
- Based on the MapReduce computing pradigm and leveraging a large suite of commodiy machines, these framworks have shown good characterstics of scalability and applicability.
- Spark will optimize a query by pushing down the filter condition before the join operation:
- then Spark will take this plan and converts it into jobs and each jobs consists of a DAG stages:



(a) Optimized compute plan



(b) Stage DAG

LinkedIn's Spark Shuffle Operation

- Spark on YARN and leverage the external shuffle service.
- Shuffle operation:
- 1、Register with ESS,which allows the ESS to
- know about the location of data.
- 2、Generate shuffle file.
- 3、Fetch shuffle blocks

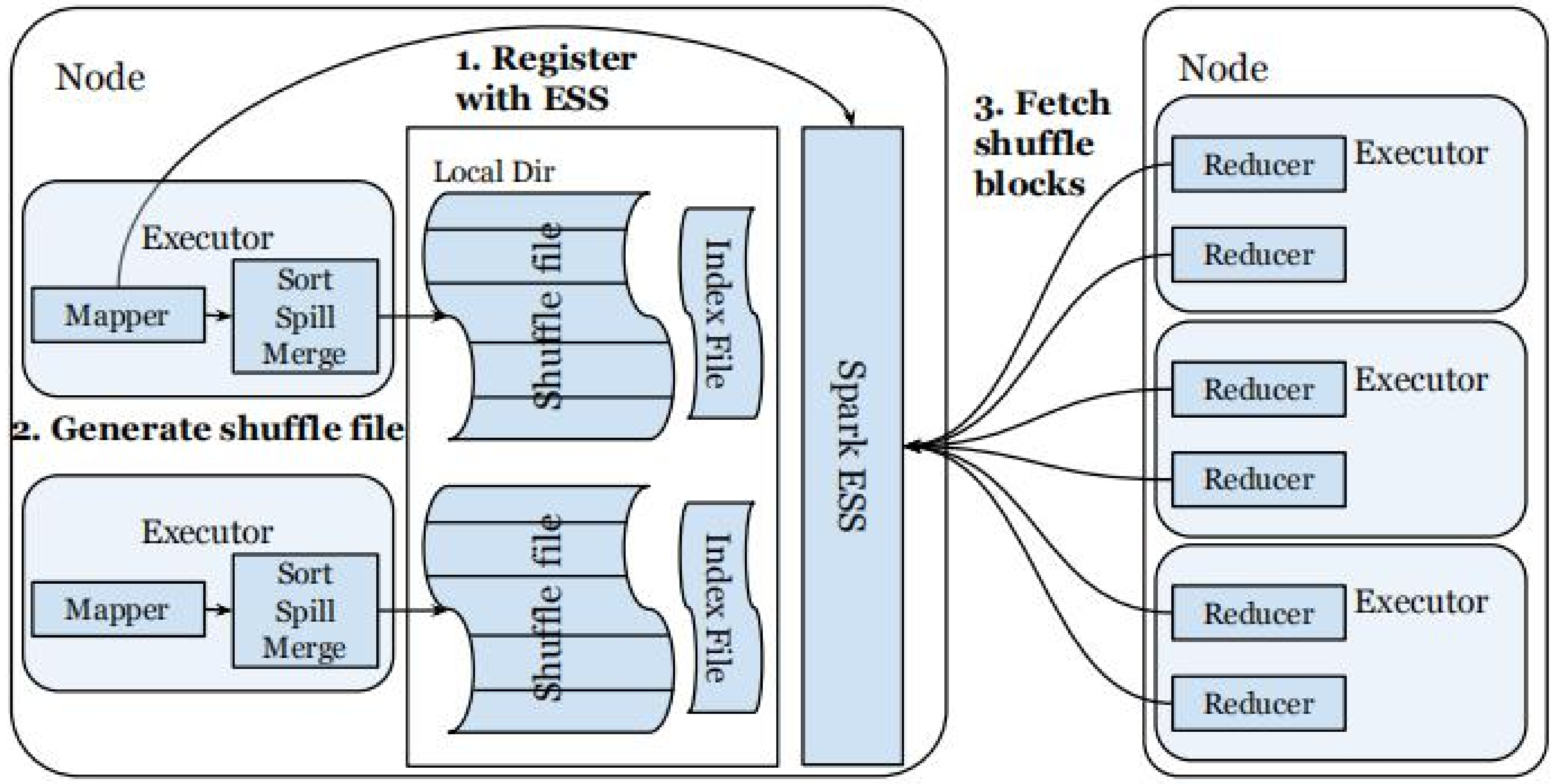
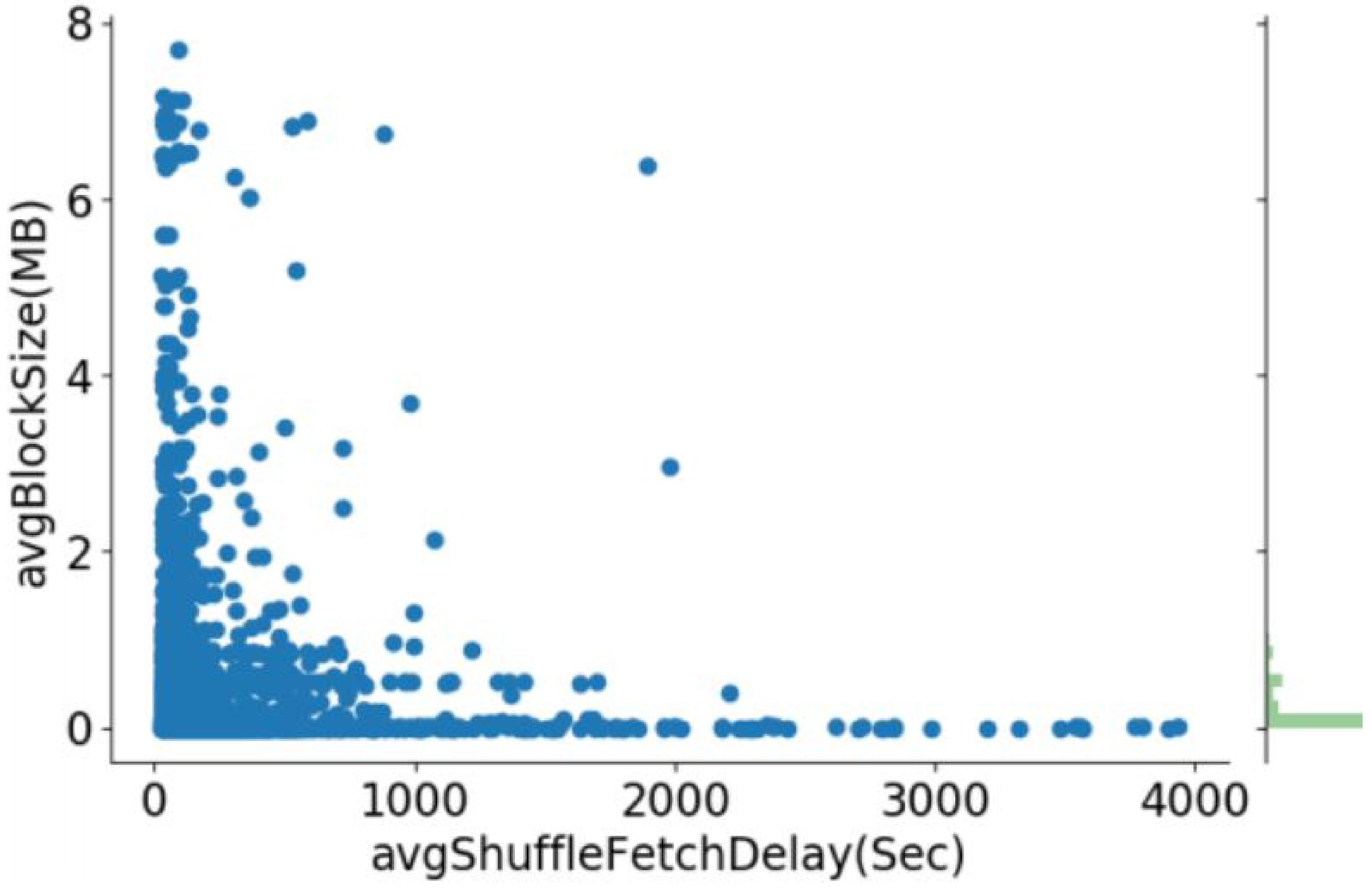


Figure 2: Illustration of the three main steps in Spark shuffle operation.

Common Issues

Inefficient Disk IO

- Inefficient disk I/O due to small shuffle blocks.
- HDDs: be limited by IOPS.
- Caching is not helpful!



Reliability

- S Spark shuffle services and E executors, up to S * E connecions would still be needed.
- both S and E can be up to 1000!
- Intermittent node availability issues can be happen.
- Spark ESS may get stressed during peak hours.
- Connection fails cause the data(cannot be fetched) to be regenerated.

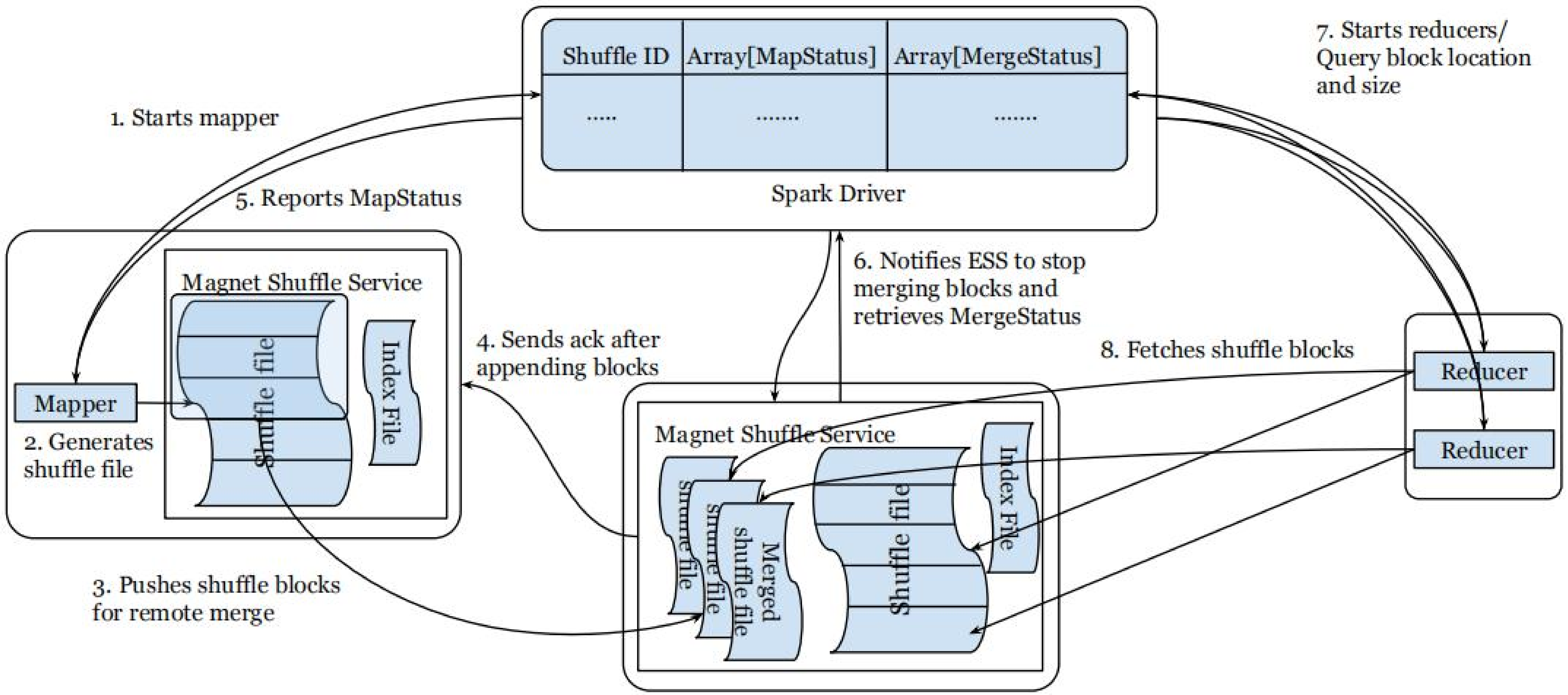
Where to place reduce tasks??

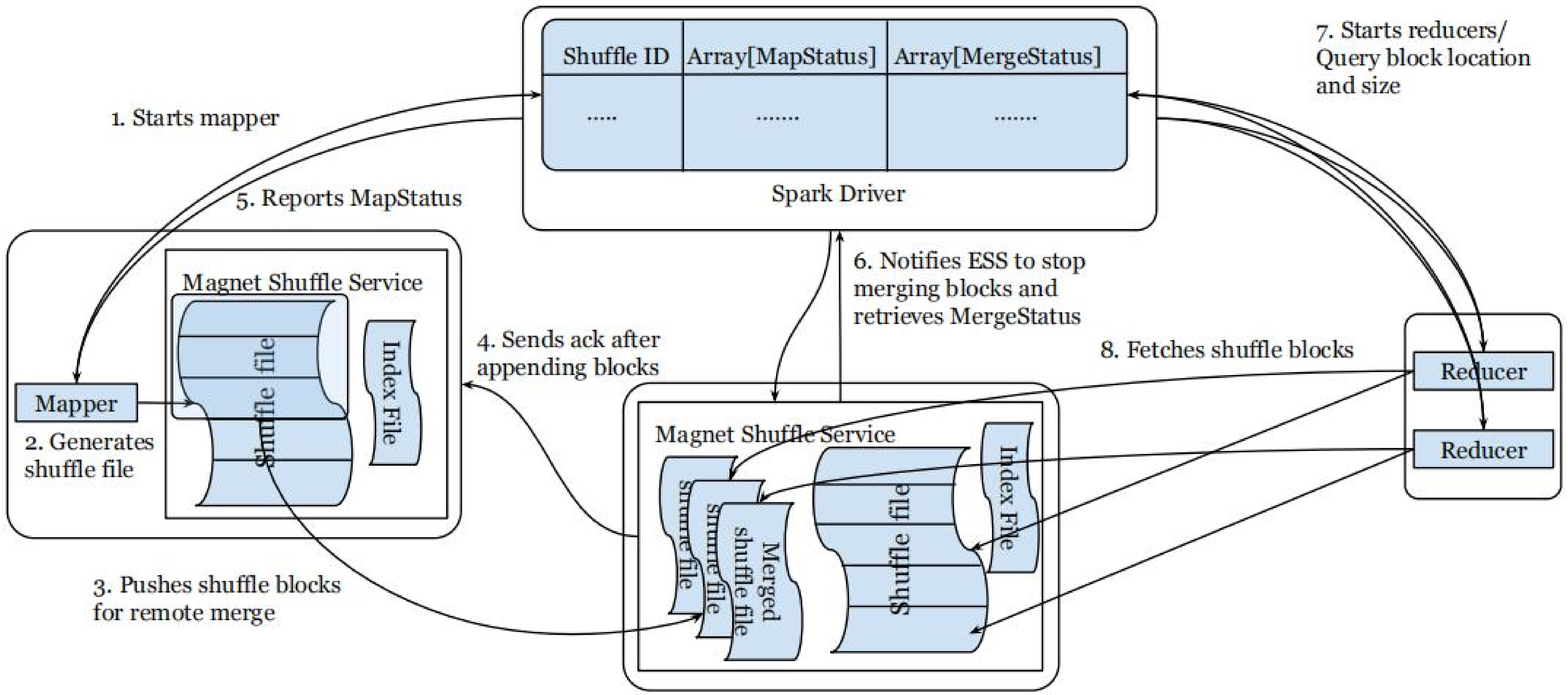
- We cannot saturate the network bandwidth due to IOPS.

- Reduce task input data is scatterd across all the map tasks.

Improvement

Magnet Architecture





- Mapper generated shuffle data
- shuffle data is pushed to Magnet shuffle services.
- Magnet merge shuffle data per shuffle partition.

Prepare Blocks for Remote Push

- Once the shuffle files are produced, they will be diveded by the map into MB-sized chunks

- each chunk only contains contiguous blocks inside the shuffle files up to a certain size

Algorithm 1: Dividing blocks into chunks

Constants available to mapper:

number of shuffle partitions: R
max chunk size: L
offset array for shuffle blocks inside shuffle file: $l_1, l_2, \ldots, l_R$
shuffle services chosen by driver: $M_1, M_2, \ldots, M_n$

Variables:

current chunk to add blocks to: $C \leftarrow \{\}$
current chunk size: $l_c \leftarrow 0$
current shuffle service index: $k \leftarrow 1$

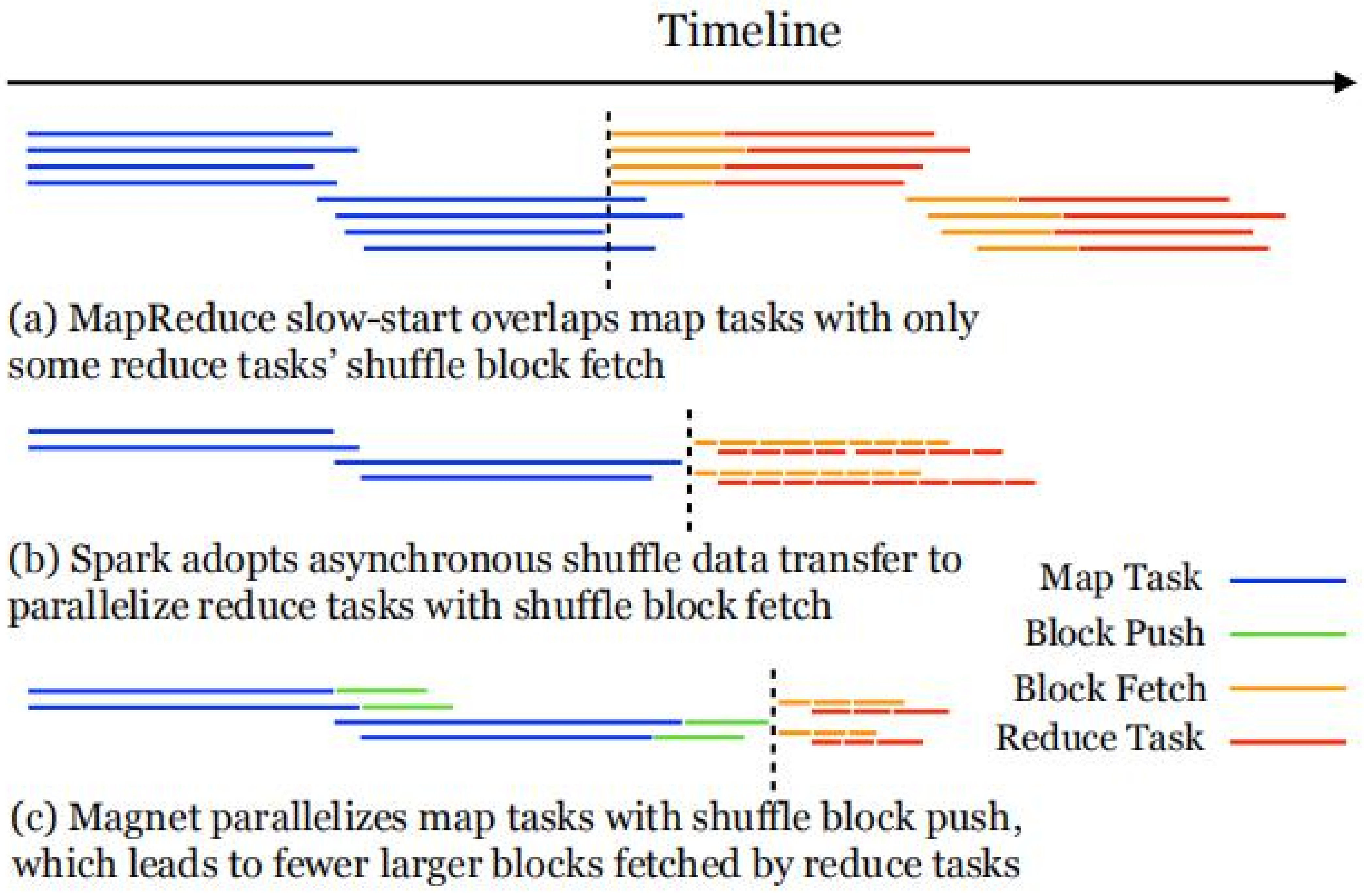
Output:

chunks and their associated Magnet shuffle services

for $i = 1 \ldots R$ **do**
  if $(i-1)/(R/n) + 1 > k$ *and* $k < n$ **then**
    output chunk and its shuffle service (C, M_k);
    $C \leftarrow \{block_i\}$;
    $l_c \leftarrow l_i$;
    $k++$;
  else if $l_c + l_i > L$ **then**
    output chunk and its shuffle service (C, M_k);
    $C \leftarrow \{block_i\}$;
    $l_c \leftarrow l_i$;
  else
    $C = C \cup \{block_i\}$;
    $l_c = l_c + l_i$;
output chunk and its shuffle service (C, M_k);

Merge Blocks on Magnet

- Magnet append the received corresponding blocks.
- Magent maintains some metadata for every merged shuffle partitions.



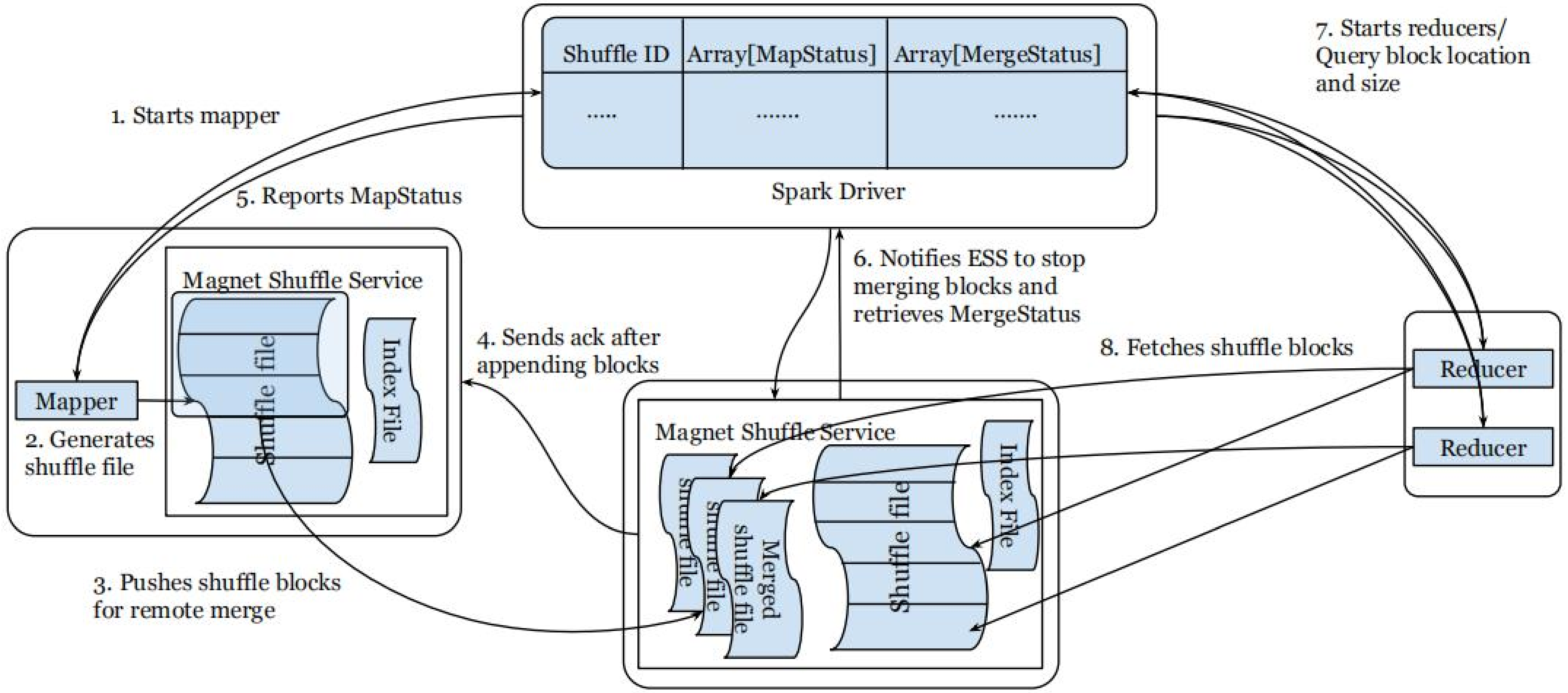
- Hadoop use a “slow start”.

- Spark achieves the parallelization with asynchronous RPC
(two threads act like a pair of producers and consumers).

- Magnet decouple push and map(thread pool), divdes each merged shuffle file into multiple slices
in order to achieve paralelization.

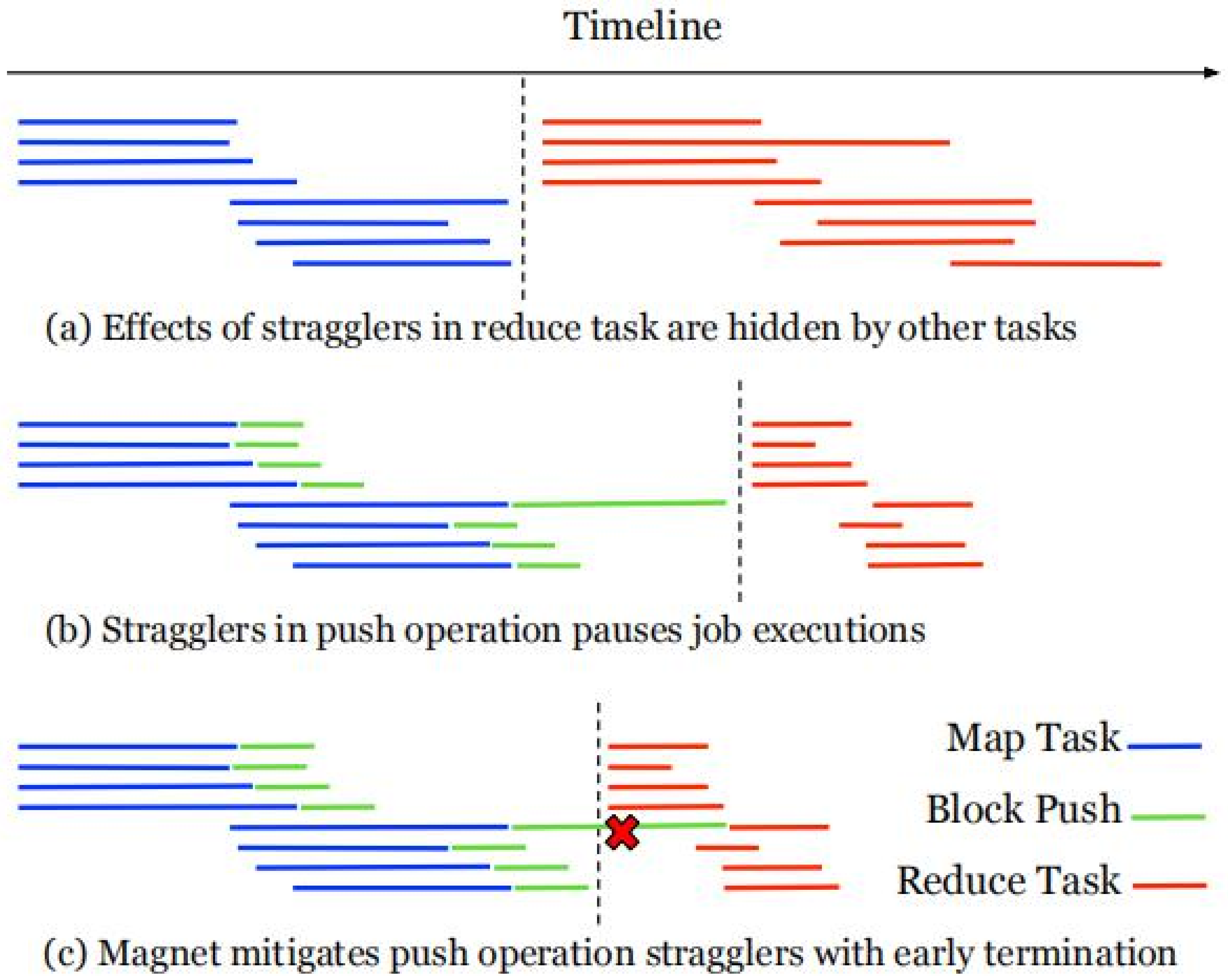
Improve Reliability

- Map tasks fail. Retry map tasks.
- Pushing shuffle block fails. The will be fetched in their original form.
- Merged-files have problem. The original unmerged blocks will be fetched instead.
- Reduce tasks fail to fetch merged-file. They can fall back to fetching the list of the unmerged shuffle blocks.



Handle Stragglers and Data Skews

- Task Stragglers

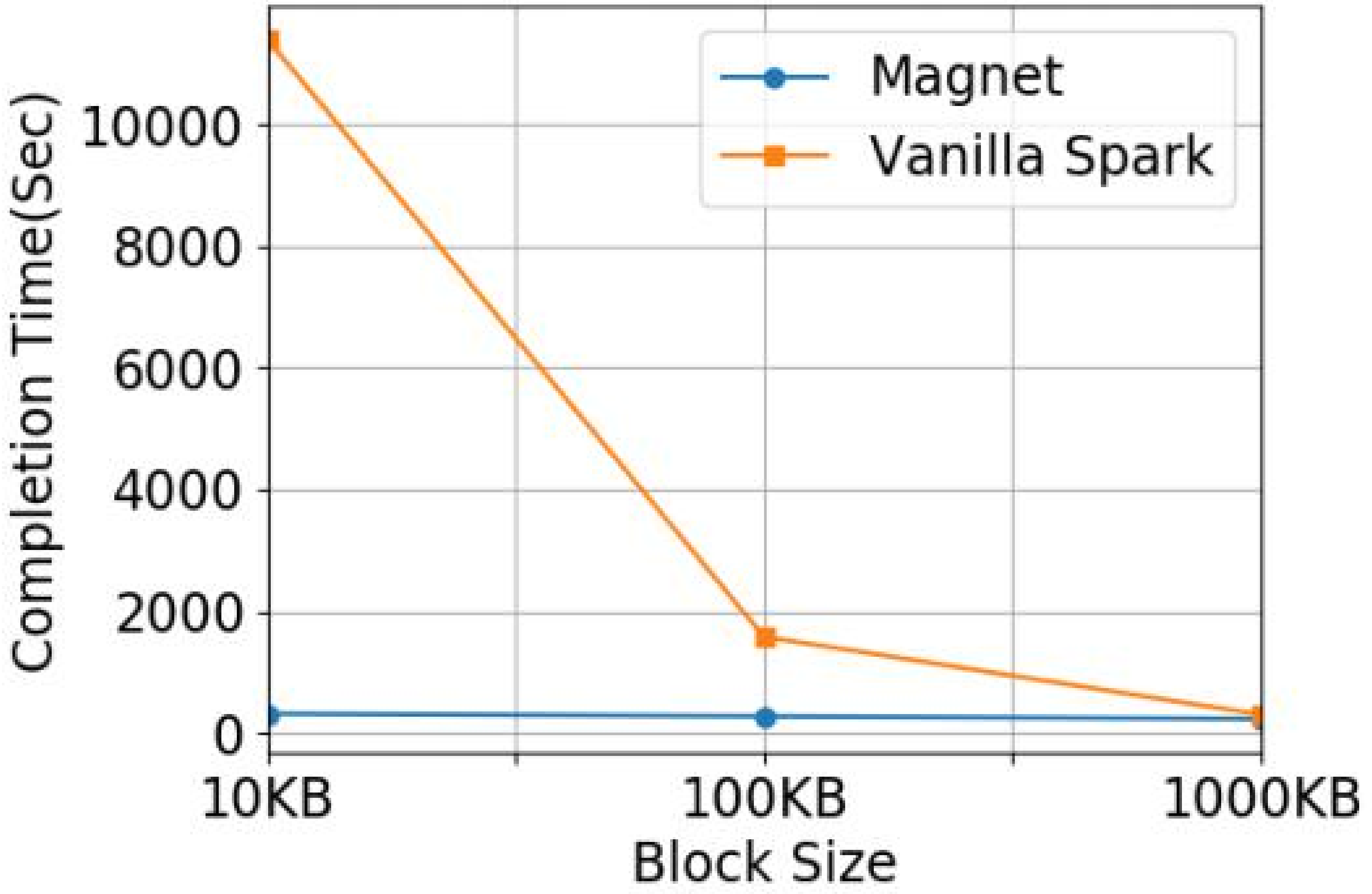


- Data Skews

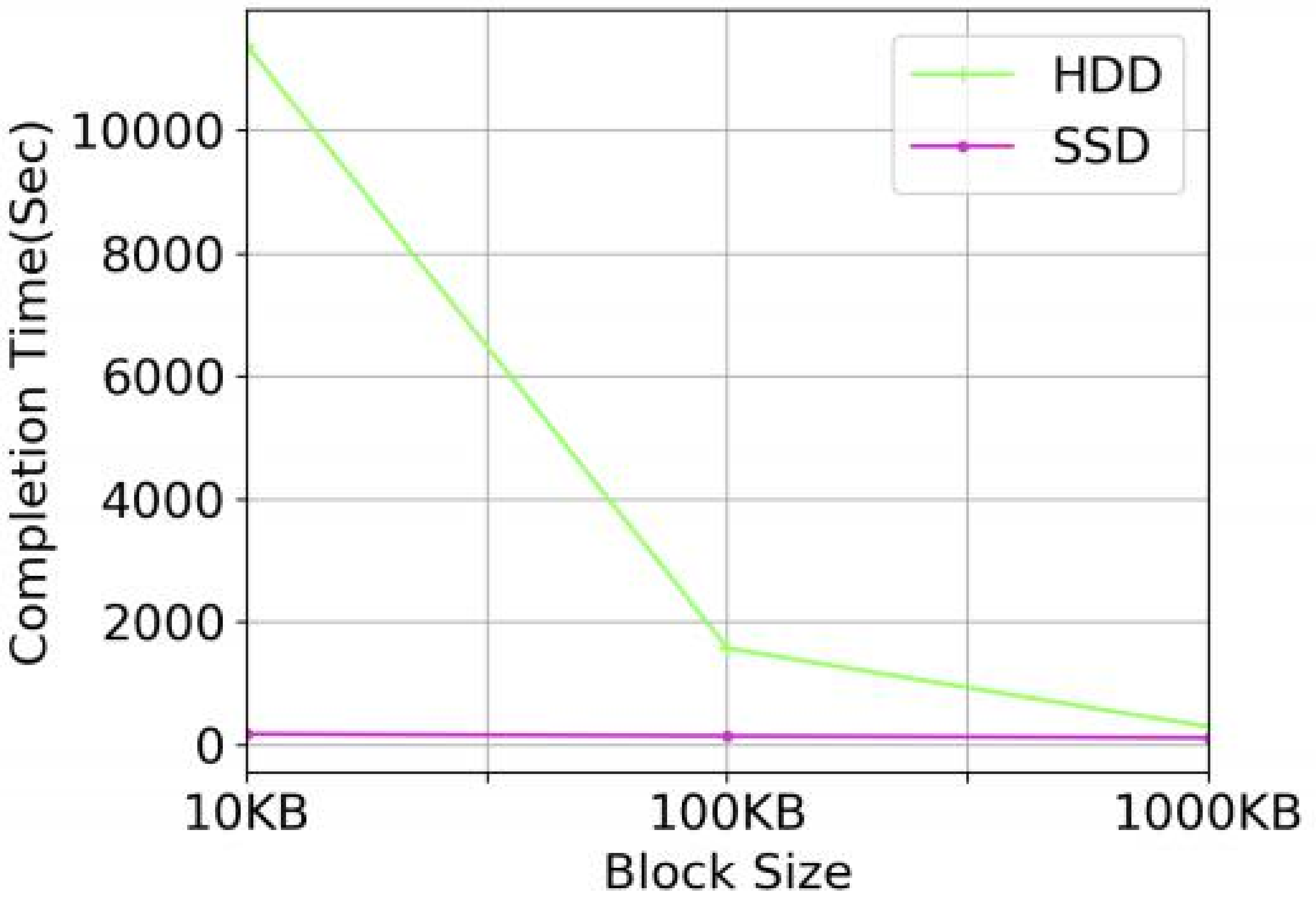
Magnet divide the skewed partitions into multiple buckets

Evaluation

Completion time

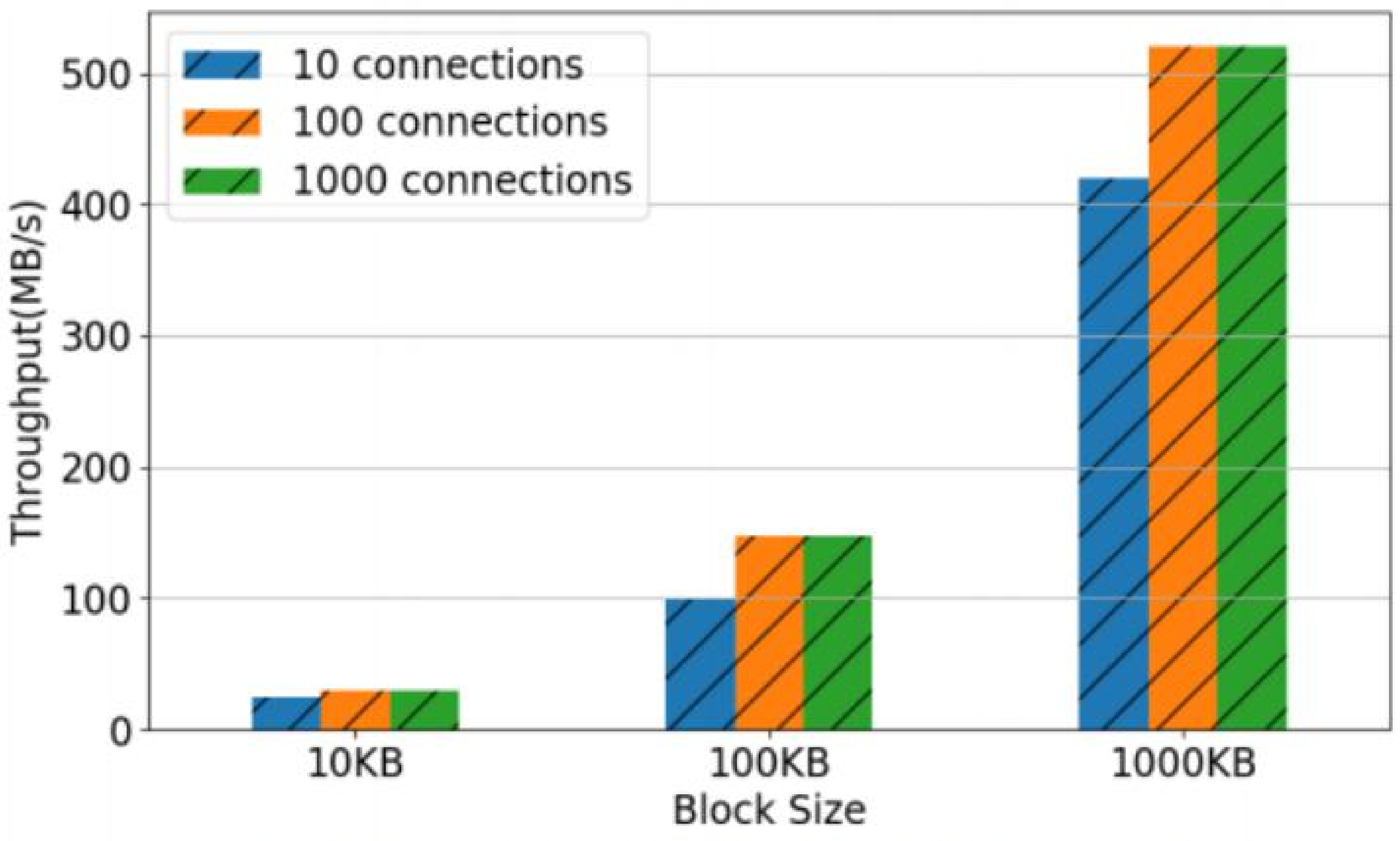


(a) Completion time for block push operation with Magnet and block fetch operation with vanilla Spark ESS

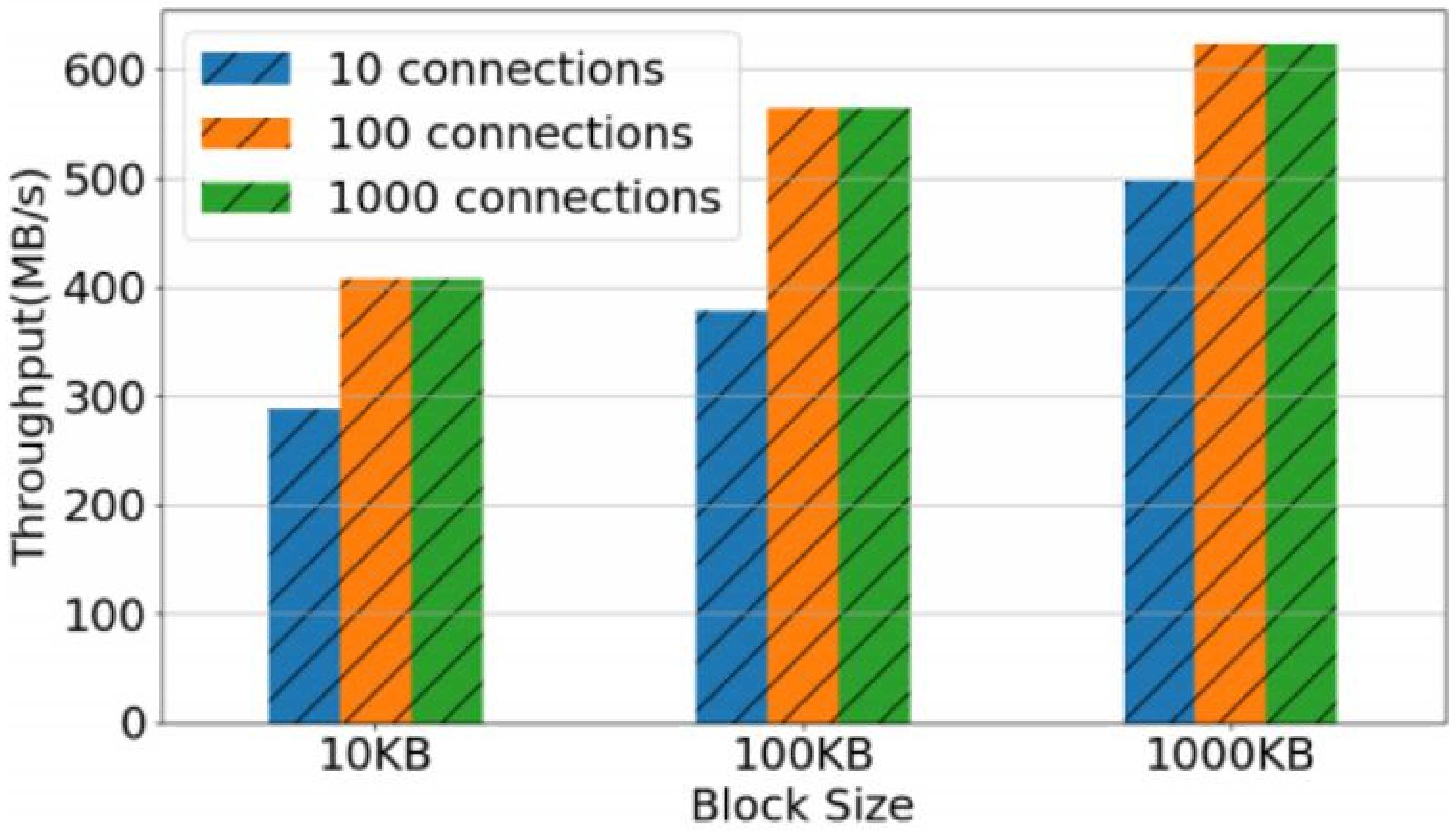


(b) Completion time for block fetch operation with vanilla Spark ESS using HDD vs. SSD

Disk IO



(a) Disk read throughput of block fetch operation



(b) Disk write throughput of block push operation

Workload Evaluation

	Map Stage Runtime	Reduce Stage Runtime	Total Task Runtime
Workload 1 w/o Magnet	2.7 min	37 s	38 min
Workload 1 w/ Magnet	2.8 min	33 s	37 min
Workload 2 w/o Magnet	2.4 min	6.2 min	48.8 hr
Workload 2 w/ Magnet	2.8 min	2 min (-68%)	15 hr (-69%)
Workload 3 w/o Magnet	2.9 min	11 min	89.6 hr
Workload 3 w/ Magnet	4.1 min	2.1 min (-81%)	31 hr (-66%)
Workload 2+3 w/o Magnet	7.1 min	38 min	144.7 hr
Workload 2+3 w/ Magnet	8.5 min	6.3 min (-83%)	43 hr (-70%)

Conclusion

- Use push-based operation to increase IO performance.
- Keep the original fetch operation for tolerance.
- Add new mechanism to handle Stragglers and skews.

Thanks