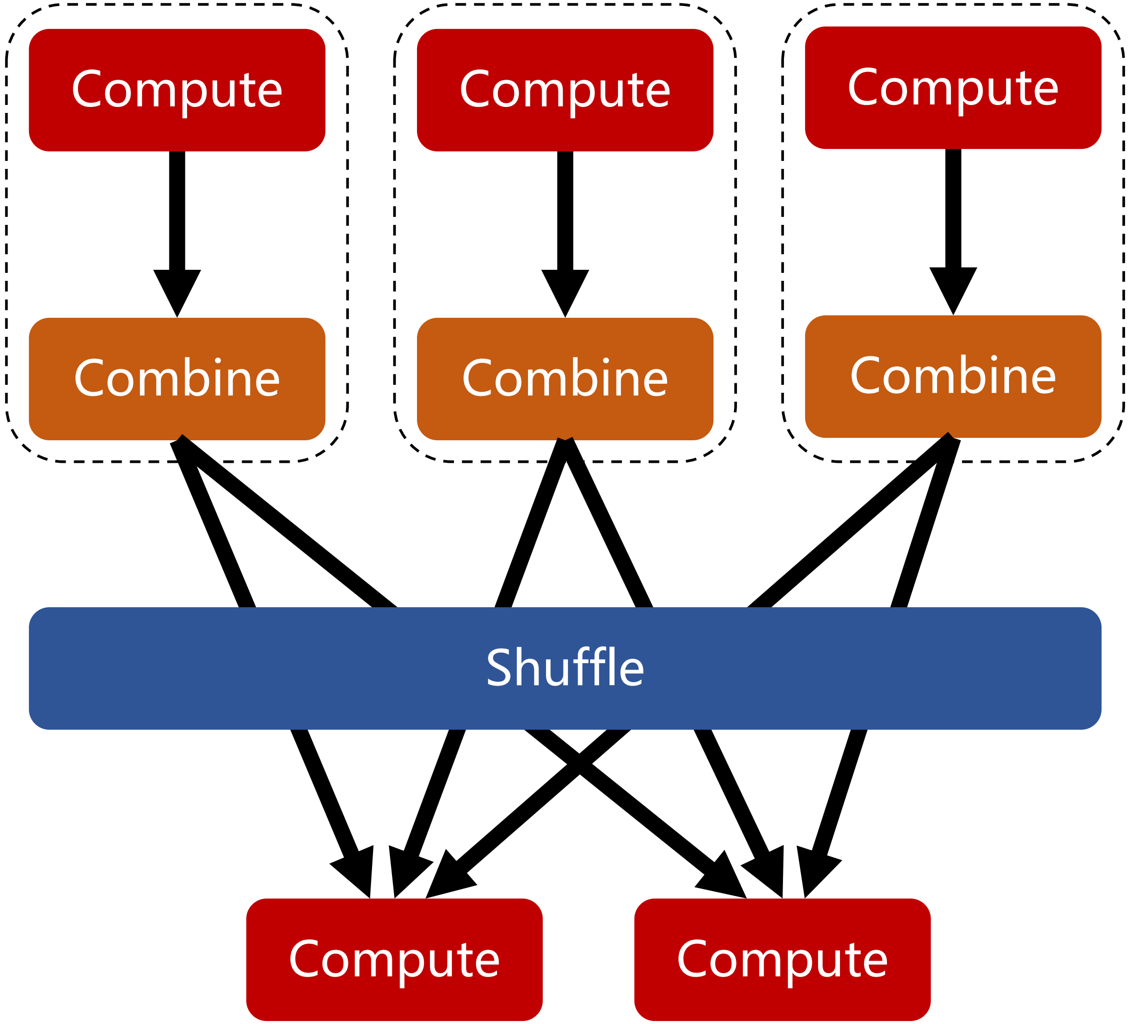
Templating Shuffles

Qizhen Zhang[1,2,3], Jiacheng Wu[4], Ang Chen[5], Vincent Liu[1], Boon Thau Loo[1]

[1]University of Pennsylvania, [2]Microsoft Research, [3]University of Toronto,
[4]University of Washington, [5]Rice University

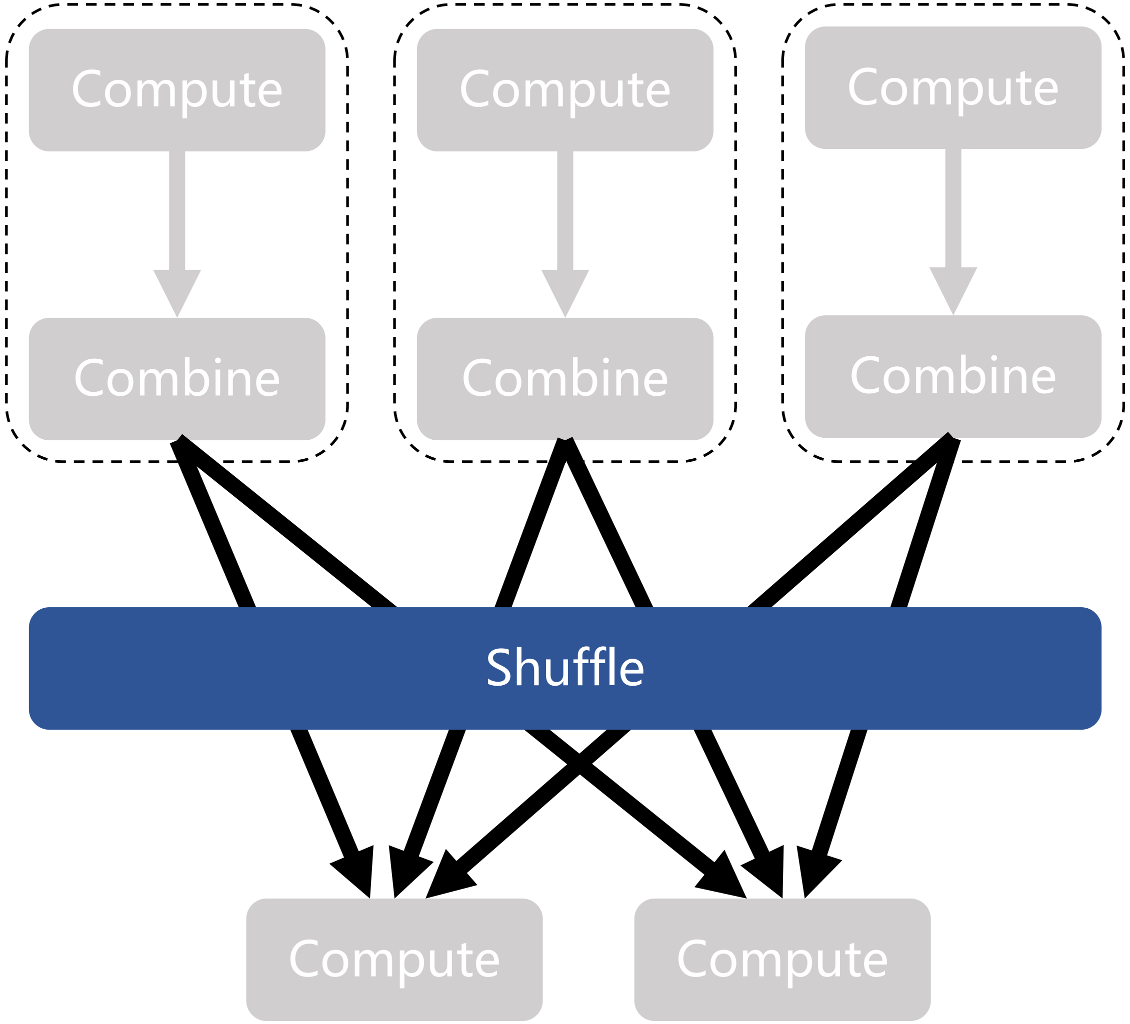
General Structure of Large-scale Analytics

- Compute
  - Distributed workers process local shards of data independently
- Combine (optional)
  - Preliminary results are locally processed before data exchange
- Shuffle
  - Resharding and transmitting data for the next phase of processing



Shuffle as Critical Component

- Encompasses CPU, bandwidth, and latency overhead
  - Compression, serialization, message processing, transmission, etc.
- A rich history of tuning shuffle for efficient data analytics
  - DBMSs, MapReduce, and graph processing
- A primary bottleneck in emerging cloud platforms
  - Serverless and disaggregated memory/storage

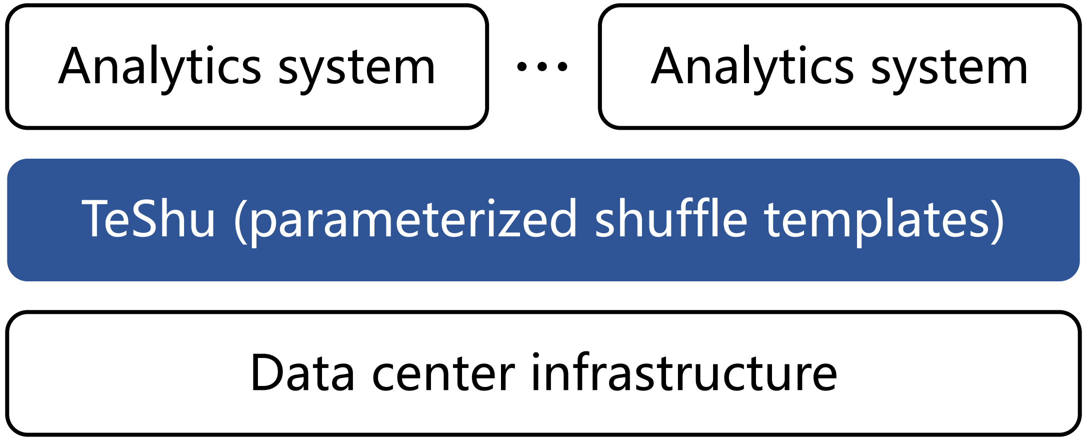


Challenges in Optimizing Shuffle

- Dependent on **workloads**
  - Disk activities, combinable, aggregable, data redundancy…
- Dependent on **data center architecture**
  - CPU performance, network bandwidth, communication locality across machines…
- Must adapt to **changes**
  - Data center topology updates caused by network failures
  - Disaggregated compute, memory, and storage
  - Next-generation data center network designs are increasingly complicated

TeShu: A Templated Shuffle Layer

- **Adapts** to workload and data center infrastructure **changes**
- **Easily supports** existing data analytics systems and enables future ones
- Users **implement** shuffle primitives as **templates** with unknown characteristics of workloads and infrastructure as parameters
- Data analytics systems **instantiate templates** by populating parameters



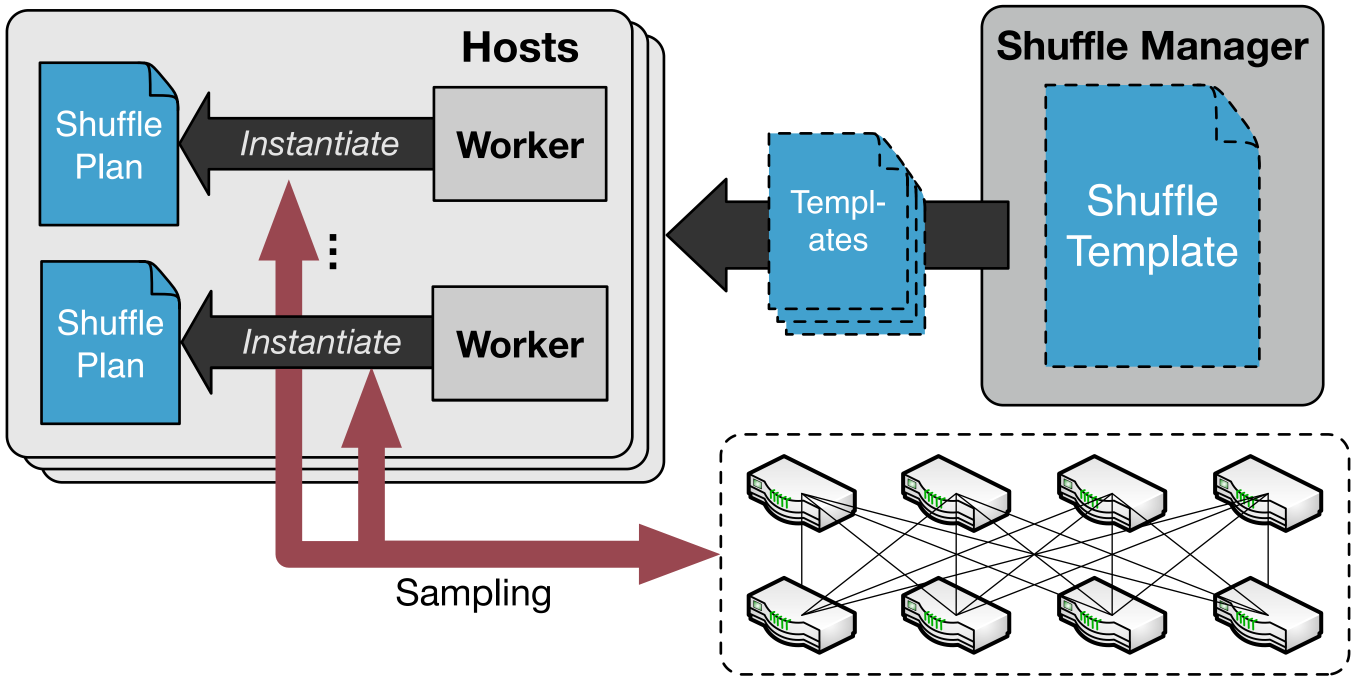
Outline

- Motivation & TeShu Vision
- TeShu Design
- Expressiveness
- Evaluation
- Future Directions

Architecture

Applications invoke shuffle API
and instantiate templates to plans

Shuffle Manager stores
and serves templates



A plan can sample messages to measure the efficiency
of a particular shuffle strategy for adaptiveness

Shuffle Templates

- Python-like programs with parameters below

Template parameter	Description
`SEND(dst, msg)`	Send `msg` to `dst`
`RECV(src)`	Return data received from `src`
`FETCH(src)`	Return data fetched from `src`
`PART(msgs, srcs, partFunc)`	Partition `msgs` into `dsts` according to `partFunc`
`COMB(msgs, combFunc)`	Combine `msgs` according to `combFunc`
`SAMP(msgs, rate, partFunc)`	Sample `msgs` based on `rate` and `partFunc`

Basic communication (supporting both pull and push)
Populated by framework-native communication libraries

Partitioning, combing, and sampling
Populated by shuffle arguments and our sampling approach

- Example: vanilla shuffling (pull mode)

Sender template:

```
PART(bufs, dsts, partFunc)
```

Receiver template:

```
for s in srcs:
  bufs[n] = FETCH(n)
```

Shuffle API

- Shuffles are instances of concurrent communication between a fixed set of sources and destinations

IDs of worker, template, and shuffle call   Sources, destinations, and data buffers

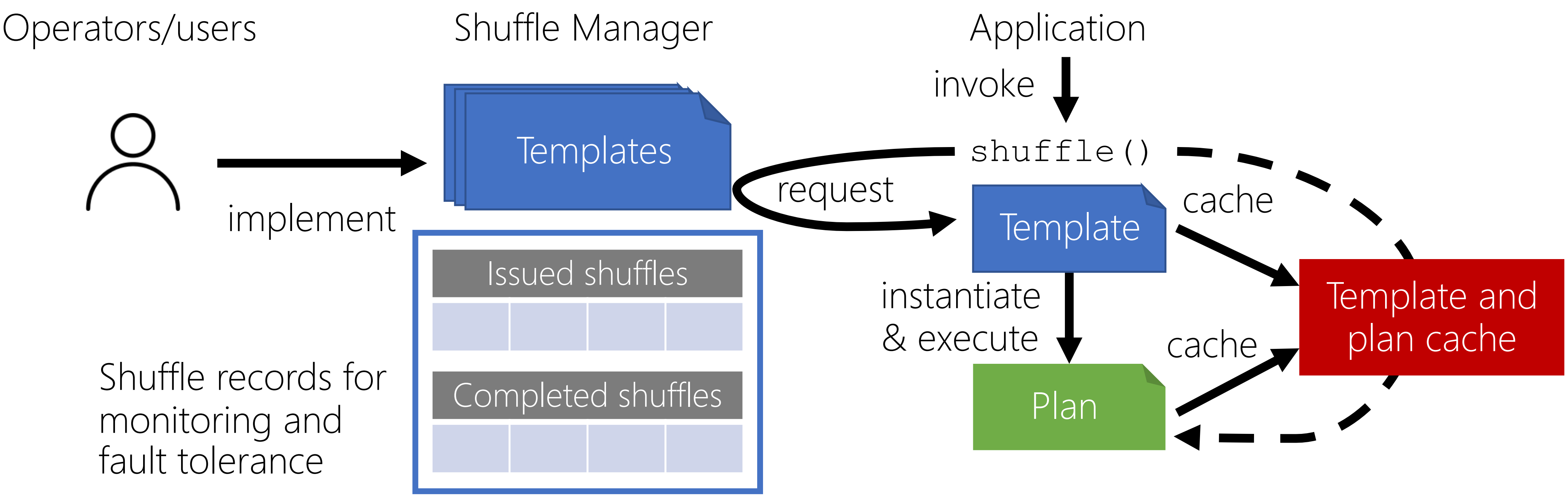
- `shuffle(wId, templateId, shuffleId, srcs, dsts, bufs, partFunc, combFunc)`

Functions for data partitioning and combining (optional)

- Partition function maps a piece of data to a destination worker
- Combine function merges two pieces of data into one
- These arguments are used to populate template parameters

Shuffle Management

- Involves Shuffle Manager and the application
  - System operators implement shuffle templates that are stored in Shuffle Manager
  - Application invokes shuffle API, instantiates templates into plans, and caches them



Outline

- Motivation & TeShu Vision
- TeShu Design
- Expressiveness
- Evaluation
- Future Directions

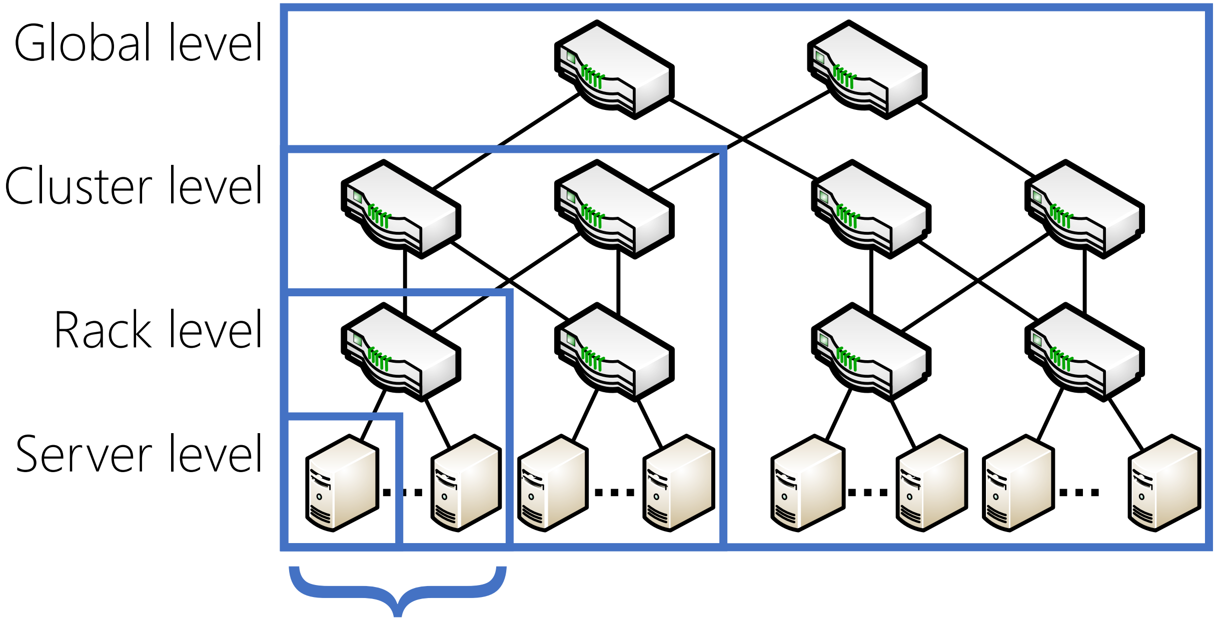
Implementing Existing Shuffle Algorithms

- TeShu can express vanilla shuffling and existing shuffle optimizations in a few lines of code

Shuffle algorithm	Pattern	LoC
Vanilla shuffling	Push/pull	5
Coordinated shuffling [CIDR ’13]	Pull	9
Bruck shuffling [IJHPCA ‘05]	Push	11
Two-level exchange [SIGMOD ‘20]	Push	18

Adaptive Shuffling

- Sampling-enabled data center network-aware shuffle optimization




```
COMB(bufs, combFunc)
sNbrs = $FIND_NBRS_PER_SERVER(wId, srcs)
sSampMsgs = SAMP(bufs, $RATE, partFunc)
(S_EFF, S_COST) = $COMPUTE_EFF_COST(sSampMsgs)
if S_EFF > S_COST:
    sPartMsgs = PART(bufs, sNbrs, partFunc)
    for n in sNbrs:
        SEND(n, sPartMsgs[n])
        sPartMsgs[n] = RECV(n)
    bufs = COMB(sPartMsgs, combFunc)
rNbrs = $FIND_NBRS_PER_RACK(wId, srcs)
rSampMsgs = SAMP(bufs, $RATE, partFunc)
(R_EFF, R_COST) = $COMPUTE_EFF_COST(rSampMsgs)
if R_EFF > R_COST:
    rPartMsgs = PART(bufs, rNbrs, partFunc)
    for n in rNbrs:
        SEND(n, rPartMsgs[n])
        rPartMsgs[n] = RECV(n)
    bufs = COMB(rPartMsgs, combFunc)
partMsgs = PART(bufs, dsts, partFunc)
for d in dsts:
    SEND(d, partMsgs[d])
```

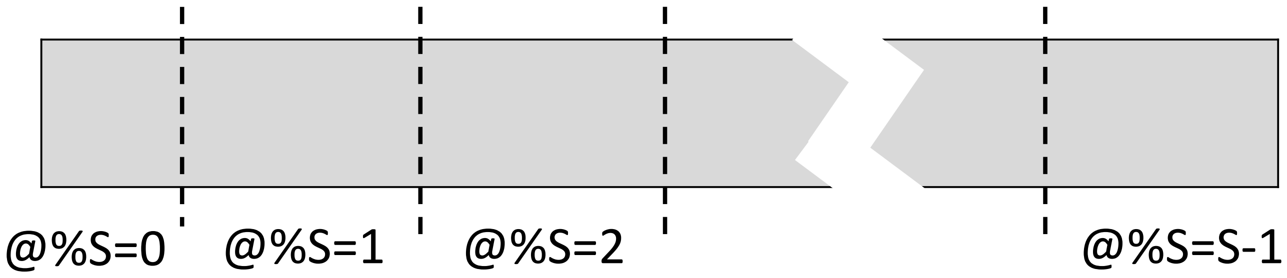
A local shuffle at each level

- Pros: reduced traffic at next level by combining (benefit oversubscribed networks)
- Cons: additional shuffling and combining overhead
- Need to compare the benefit of traffic reduction and the overhead

Measure through data sampling

Data Sampling

- Selects a subset of shuffle data from each worker based on sampling rate
- **Baseline: random sampling**, which is inaccurate with real data when sampling rate is kept low
- **Partition-aware sampling: samples data according to the destinations** to evaluate reduction rate when combiner is applied
  - Divides destination space into S buckets (S is calculated by sampling rate)
  - Allocates each piece of data into one of the buckets based on its destination
  - Selects Bucket j for sampling (j is randomly selected and consistent across workers)



Outline

- Motivation & TeShu Vision
- TeShu Design
- Expressiveness
- Evaluation
- Future Directions

Evaluation Setup

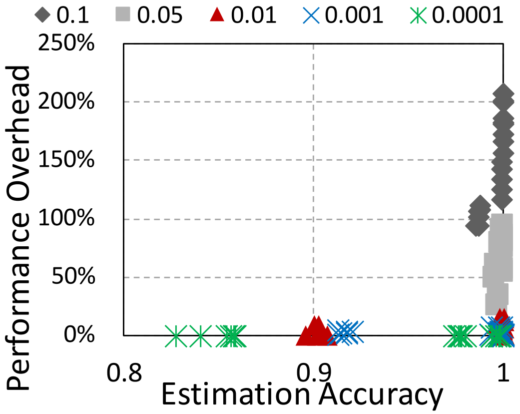
- Cluster: 2 racks of 10 servers, each with 16 cores @2.6GHz, 128 GB memory, and 10 Gbps NIC
- Network: oversubscription of inter-rack network varies (10:1, 4:1, 1:1)
- Software: Pregel+ for graph analytics
- Queries: PageRank and single source shortest path
- Datasets: UK-Web (3.7 billion edges) and Friendster (3.6 billion edges)

Sampling Performance

- Duplication estimation on shuffled data with typical workloads

Sampling rate	Ground truth	Part.-aware sampling	Random sampling
0.9	0.1833	0.1833	0.1986
0.1	0.1833	0.1833	0.7241
0.01	0.1833	0.1832	0.9622
0.001	0.1833	0.1829	0.9965
0.0001	0.1833	0.1838	0.9997

- Sampling overhead (execution time)



Adaptive Shuffling Performance

- Compared to vanilla shuffling, adaptive shuffling speeds up queries from 3.9× to 14.7× by eliminating most of the communication cost
- Across network oversubscription scenarios, adaptive shuffling always identifies the optimal shuffling strategy

	PR-UK	**PR-FR**	**SSSP-UK**	**SSSP-FR**
	Oversubscription Ratio = 10:1			
Execution Speedup	14.7×	9.4×	6.1×	7.1×
Communication Saving	87.7%	89.8%	84.6%	84.3%
Shuffle Decision	*S,R,G*	*S,R,G*	*S,R,G*	*S,R,G*
	Oversubscription Ratio = 4:1			
Execution Speedup	9.4×	5.6×	5.2×	6.2×
Communication Saving	85.5%	85.9%	81.6%	79.3%
Shuffle Decision	*S,R,G*	*S,R,G*	*S,R,G*	*S,R,G*
	Oversubscription Ratio = 1:1			
Execution Speedup	7.7×	3.9×	4.8×	4.8×
Communication Saving	80.7%	76.4%	75%	66.8%
Shuffle Decision	*S,G*	*S,G*	*S,G*	*S,G*

Outline

- Motivation & TeShu Vision
- TeShu Design
- Expressiveness
- Evaluation
- Future Directions

Future Directions

- **Co-scheduling shuffles** to achieve shorter flow completion times and thus improve application performance
- **Handling failures and stragglers** with shuffle records
- **Integrating with in-network techniques** to apply combining and sampling in the network to further improve performance
- **Templating shuffles for future data centers**, e.g., data movement between disaggregated resource components

Summary

- Tuning shuffle for large-scale data analytics is necessary but challenging and requires adaptiveness and portability
- TeShu provides a simple and expressive shuffle abstraction and offers it as a general layer to benefit various data analytics systems
- Shuffle templates and efficient sampling in TeShu enable portable and adaptive shuffle optimizations
- More aspects to be investigated and opportunities to be explored

Backup Slides

Related Work

- Riffle [EuroSys '18]
  - External shuffle service in Spark developed by Facebook
  - Merges small files to reduce random disk I/O
- Magnet [VLDB '20]
  - Optimized shuffle service in Spark developed by LinkedIn
  - Pushes shuffle data from mappers to reducers to pre-merge intermediate results before the reduce stage
- Remote Shuffle Service (RSS)
  - Spark shuffle service with dedicated remote shuffle servers developed by Uber
  - Separates shuffle data from mappers to improve reliability and scalability
- Exoshuffle
  - Shuffle layer for MapReduce in Ray that easily supports state-of-the-art shuffle optimizations and enables pipelined shuffles

More Related Work

- Optimizing shuffle for graph processing
- Optimizing shuffle for machine learning
- Optimizing the exchange operator in DBMSs