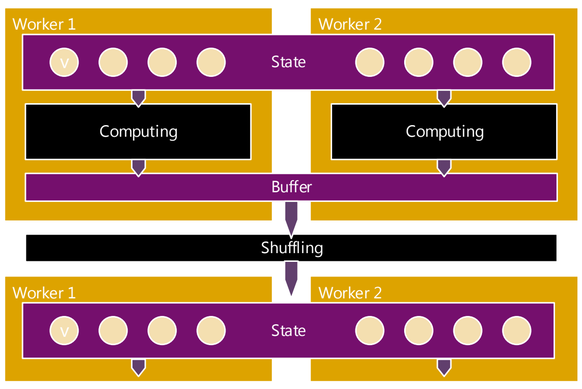
Graph Analytics Through Fine-Grained Parallelism

Zechao Shang[†], Feifei Li[‡], Jeffrey Xu Yu[†], Zhiwei Zhang[§], Hong Cheng[†]

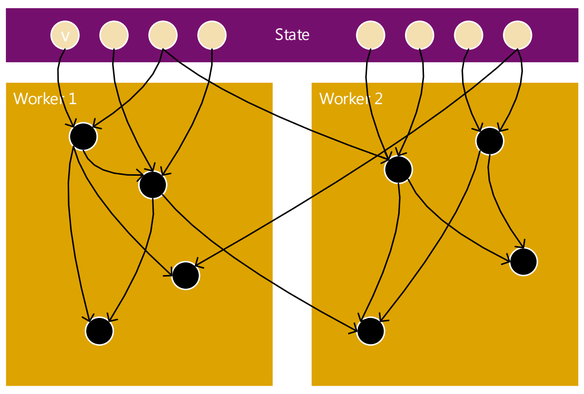
[†]The Chinese University of Hong Kong, [‡]University of Utah, [§]Hong Kong Baptist University

The Graph System War

Coarse-grained parallelism vs. fine-grained parallelism

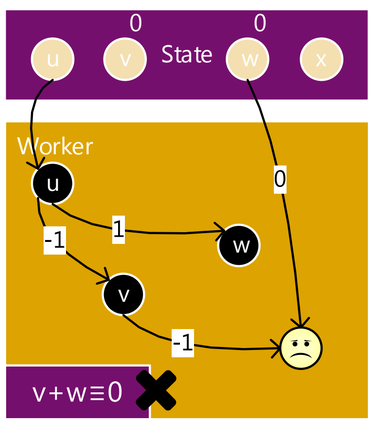


- Partitioned immutable state
- Embarrassingly parallelism
- Suffers from:
  - Straggler
  - Data shuffling/copy overheads
  - No intra-iteration optimizations

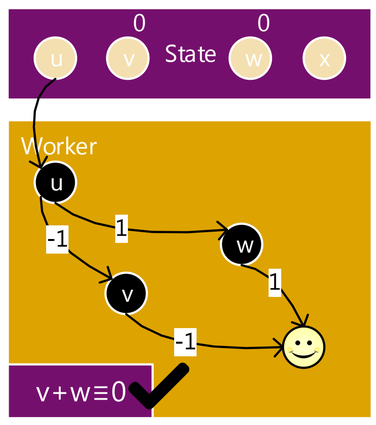


- Shared mutable state
- More flexible implementation
- Faster communication
  - Could be asymptotically better
- Modularized system design
- Potential problem:
  - Data consistency (see below)

Asynchronous vs. synchronous



- No consistency control
- Data integrity at risk
- May lead to:
  - Slower convergence
  - Incorrect results
  - Corrupted data



- Consistent data
  - Performance depends on consistency controller efficiency

Synchronous Parallel Processing for Fine-Grained Parallelism

System setting:

- *In-memory* graph analytics
- Static and exclusively owned data
- Target: as fast as possible

Our system components:

- Shared mutable state
- Controlled data consistency
- User-implemented UDF
- Customizable job scheduler

User API:

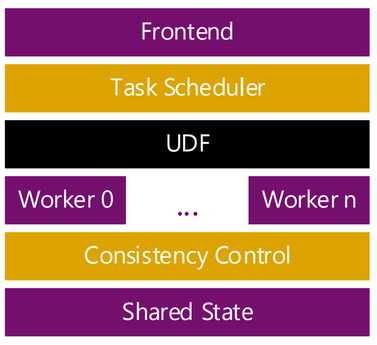
- A user-defined function
- Read and write shared state
- Add task to job queue
- Optional barrier

Execution Workflow:

- Get task from job queue
- Execute it with consistency guarantee (**serializable**)
- Synchronous at barrier if demanded
- Can simulate asynchronous, BSP and prioritized scheduling

System Features:

- Strong semantics
  - Equivalent to serial execution
  - No asynchronization, no nondeterministic
  - Always-correct results
- Flexible model
  - Can simulate asynchronous, BSP and prioritized scheduling by customizing schedulers and concurrency controller
- Fast information diffusion
  - Enable intra-iteration optimization
  - Support asymptotically better algorithm (for example, stochastic gradient descent vs. gradient descent)




```
// Vertex Class
class vertex {
  abstract void UDF(vertex v);
  value_t readVertex(vertex v);
  void writeVertex(vertex v, value_t val);
}

// Global Functions
addTask(vertex v, priority = 1.0);
barrier(function f);
```

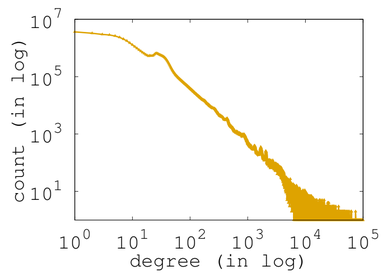
```
in parallel for each worker
  if at barrier
    wait until all workers reach barrier
    execute the function f
    continue
  if no more tasks in the scheduler queue
    exit
  retrieve (and remove) task vertex v
  begin transaction
    execute UDF(v) as a transaction t(v)
  end transaction
```

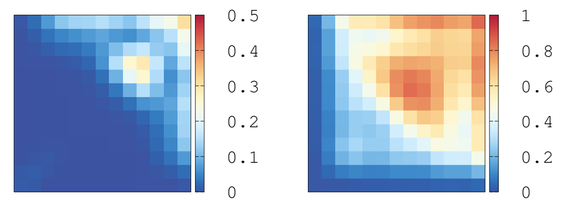
```
// A Page Rank example of usage
void UDF(vertex v) {
  // calculate the page rank value
  double sum = 0, previous = readVertex(v);
  for (all neighbors u)
    sum += readVertex(u);
  sum = sum * (1 - d) + d;
  writeVertex(v, sum);

  // schedule v with priority
  double gap = abs(sum - previous);
  if (gap > 1e-6)
    addTask(v, gap);

  // report vertices with top-k page rank
  // values periodically
  if (time elapsed 1 sec since last report)
    barrier({
  // find current top-k vertices and print
  });
}
```

HSync: A Hybrid Scheduler

Why existing ones are not working?

- Graph degree distribution is highly skewed (see right top)
- The transaction conflict rate is skewed (see right bottom)
- Traditional schedulers are optimized for particular homogeneous workload
  - Locking-based for high conflict rate
  - Optimistic for low conflict rate
  - Multi-version for read-only
- None of them works well for heterogeneous conflict rate workload



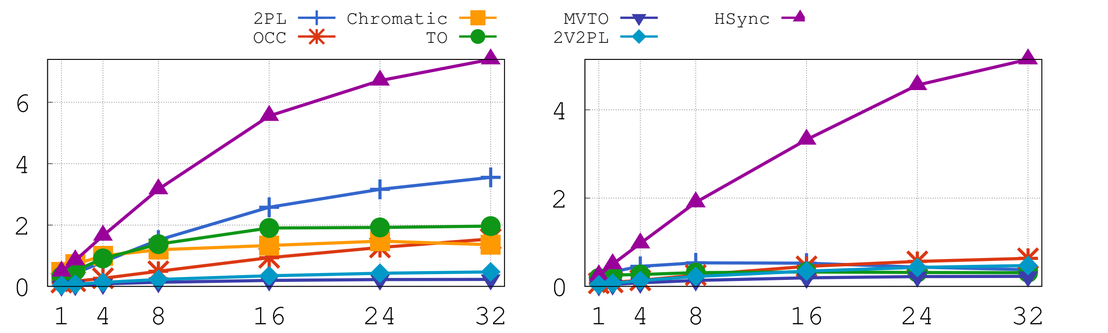


Our hybrid scheduler:

- Combines the advantages of two-phase lock (2PL) and optimistic scheduler (OCC)
- Routes transactions to an appropriate one based on the **degrees** of the vertices.
  - Large degree vertex ⇒ 2PL
  - Small degree vertex ⇒ OCC
- Ensures the serializability by:
  - OCC requests 2PL's locks at verification step
  - If fail, abort
- Check our paper for details and proofs

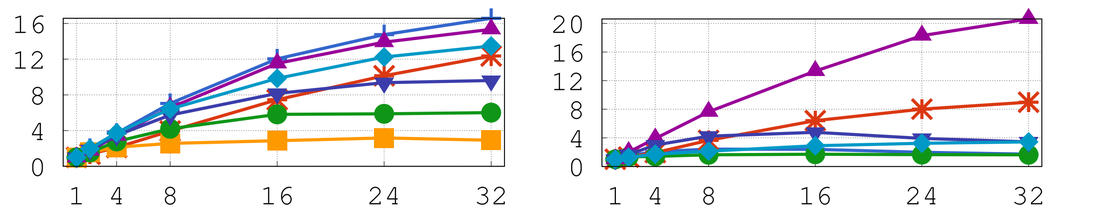
Experimental Studies

First, we measured the throughput of schedulers. Our HSync outperforms others significantly.



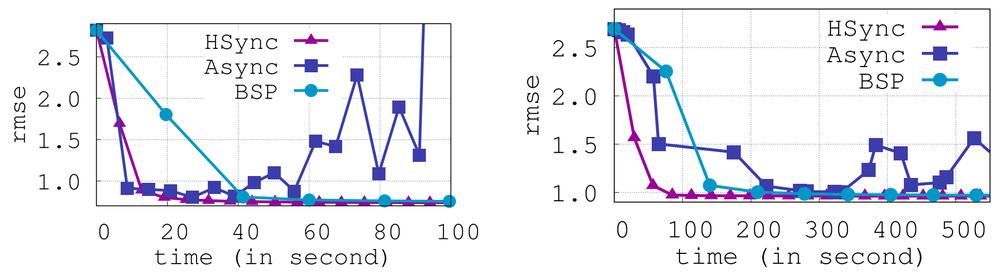
twitter-mpi dataset. x-axis for number of cores, y-axis for throughput ($\times 10^6$). Left for read-heavy and right for write-heavy workload.

We also computed the scalability of each scheduler.

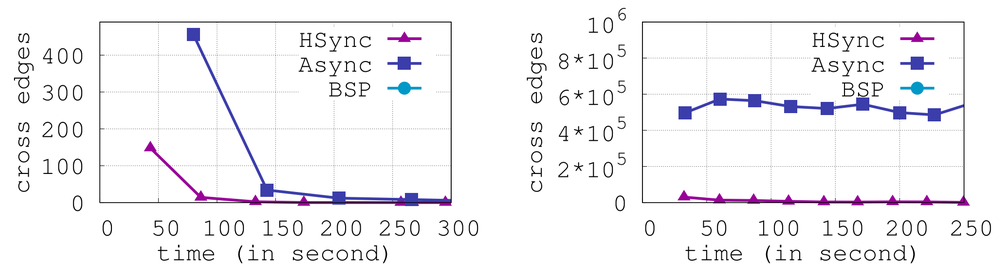


twitter-mpi dataset. x-axis for number of cores, y-axis for throughput normalized by throughput of 1-core. Left for read-heavy and right for write-heavy workload.

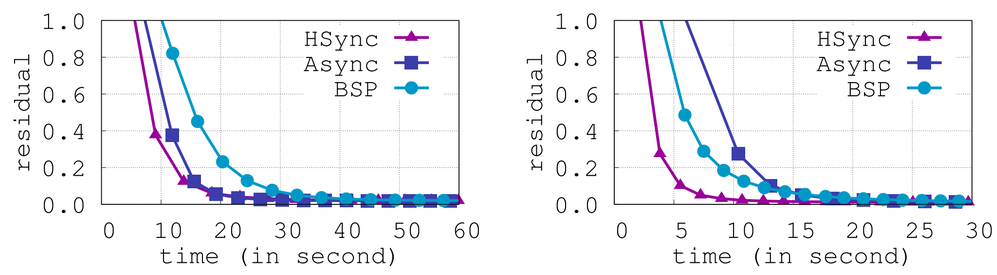
To verify the advantage of fine grained parallelism using synchronous parallel processing (with HSync) against asynchronous parallel and BSP processing for graph analytics, we tested PageRank, single source shortest path (SSSP), graph coloring, and alternative least squares (ALS) for matrix factorization.



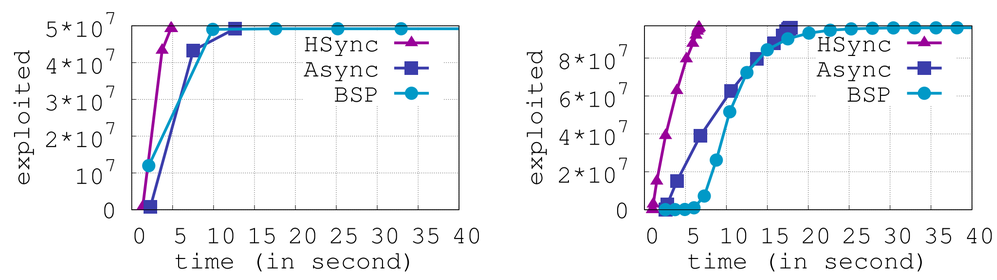
ALS. Left for netflix dataset and right for Yahoo! dataset. The smaller the better.



Graph coloring. Left for twitter-mpi dataset and right for uk-2007-05 dataset. The smaller the better. BSP results are beyond the figures' boundaries.



Page Rank. Left for twitter-mpi dataset and right for uk-2007-05 dataset. The smaller the better.



SSSP. Left for twitter-mpi dataset and right for uk-2007-05 dataset. The larger the better.

香港中文大學 The Chinese University of Hong Kong

THE UNIVERSITY OF UTAH

香港浸會大學 HONG KONG BAPTIST UNIVERSITY