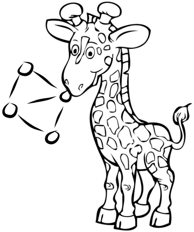
Optimizing Synchronous Online Computation of Large Graphs

Alexandre Fonseca

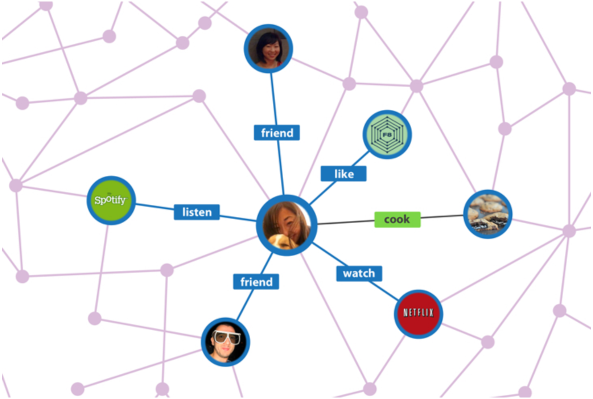


EMDC Thesis Defense — June 25, 2014

Table of Contents

Table of Contents

Graphs



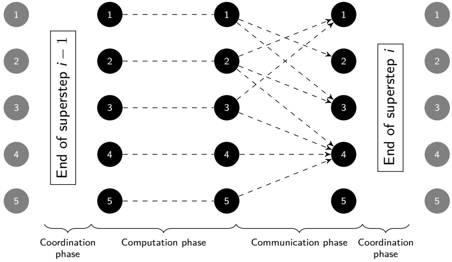
Business Insider - "So what the heck is the 'Social Graph' Facebook Keeps Talking About?"
http://www.businessinsider.com/explainer-what-exactly-is-the-social-graph-2012-3

Graph processing

- Graphs and graph analytics have unique characteristics:
    - Large size.
    - Traversal of relationship chains.
    - Highly dynamic.
    - Need to be kept up to date.
- Creation of graph-optimized distributed processing systems:
    - Examples: Pregel, Graphlab.

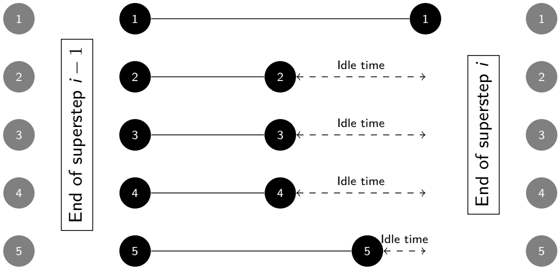
Synchronous model — Pregel

- Pregel uses a synchronous execution model (BSP):
  - Computation divided in a sequence of phases (supersteps).
  - Advantages:
    - Easy programming and debugging (deterministic).
    - Simple scalability.
  - Disadvantages:
    - Need for global coordination at the end of each superstep.



Synchronous model — Pregel

- Effect of computational skew on global coordination:



Asynchronous model — Graphlab

- Graphlab uses an asynchronous execution model:
  - Each processor executes at its own pace.
  - Advantages:
    - Relaxed coordination $\Rightarrow$ no idle time.
  - Disadvantages:
    - Programming and debugging is now harder.
    - Need complex agreement protocols.

Sync vs Async

- Both approaches have their issues.
- Would it be possible to reconcile:
  - Programming simplicity and scalability of synchronous.

    with
  - Reduced idle times of asynchronous.

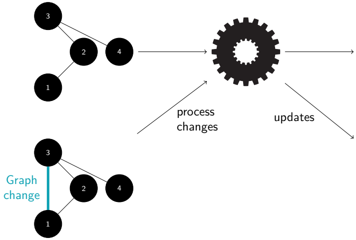
??

Contributions

- Improve latency and throughput of graph analytics over Pregel.
  - Take advantage of idle times caused by skew.
- 3 mechanisms:
  - Event pipelining.
  - Partition-level local synchronization.
  - Vertex-level local synchronization.

RTGiraph

- Online graph processing.
- Applies concepts of incremental computation to Pregel:
  - Speedup recomputations on mutations.
  - Reuse results of previous computations.



Vertex	Value
1	0.143
2	0.933
3	0.172
4	1.000

Vertex	Value
1	0.564
2	0.933
3	0.122
4	1.000

Table of Contents

Event pipelining — Introduction

- Event — Set of mutations to a base graph.
- RTGiraph processes events one at a time.
  - Guarantees no conflicting data operations.
  - Waste of computational resources.
    - Some nodes might not be involved.
    - Wasted idle times.
- Possibility to parallelize event execution.
  - Use idle times for useful computation of concurrent events.
  - Continue ensuring correctness.

Memoized state access pattern

	$S1$	$S2$	$S3$	$S4$	$S5$
E_{Y-1}					
E_Y					

- Execution of $E_Y S_X$ depends only on data from $E_{Y-1} S_X$.

Pipelined execution

$E_1 S_0$	$E_1 S_1$	$E_1 S_2$	$E_1 S_3$	$E_1 S_4$	$E_1 S_5$			
	$E_2 S_0$	$E_2 S_1$	$E_2 S_2$	$E_2 S_3$	$E_2 S_4$	$E_2 S_5$		
		$E_3 S_0$	$E_3 S_1$	$E_3 S_2$	$E_3 S_3$	$E_3 S_4$	$E_3 S_5$	
			$E_4 S_0$	$E_4 S_1$	$E_4 S_2$	$E_4 S_3$	$E_4 S_4$	$E_4 S_5$

- Assume X seconds necessary for processing a superstep.
  - Time required to process 4 events with 5 supersteps:
    - No pipeline $\Rightarrow 20X$ seconds.
    - Pipeline $\Rightarrow 9X$ seconds.

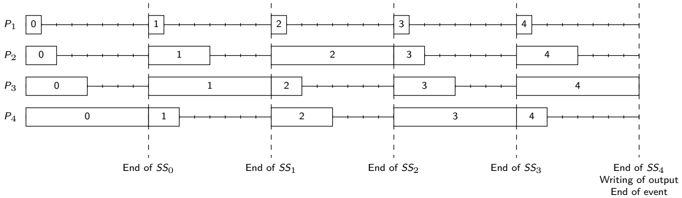
Challenges

- Coordinate event scheduling.
- Handle multiple simultaneous event executions.
- Memoization mechanism:
  - Guarantee correctness.
  - Optimize.

Solution

- New rules to allow execution of superstep S_X of event E_Y.
  1. E_{Y-1} has finished S_X.
  2. If $X = 0$, E_Y can only execute if there are less than *pipelineSize* events running.
- Multiple event executions per node.
  - Thread pool with 1 thread per execution.
  - Mapping and routing of messages to correct event execution.
- Rewrite memoization mechanism.
  - Keep information of different events in separate files.
  - Allow gaps in memoized data.

Partition localsync — Introduction

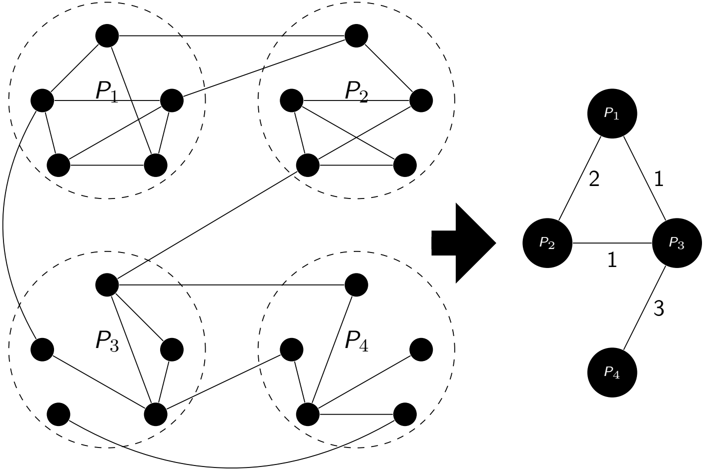


- Pregel splits graph computation over partitions of graph.
- Pregel uses global synchronization barriers at end of superstep.
  - Partitions only continue when all finished current SS.
  - Inefficient under skew.
- Replace global barriers with local ones:
  - Allow partition to continue when all data received.
  - Even if others still running current superstep.

Communication assumptions

- Pregel makes no assumptions regarding communication.
  - Every vertex can send a message to any other vertex.
  - Not compatible with local synchronization.
  - Not widely used.
- Stricter assumptions:
  - Vertex may only send messages through edges.
  - Edges between different partitions determine dependencies.

Partition meta-graph



Challenges

- Tracking dependencies.
- Original execution framework too rigid.
  - Partitions need to execute independently.
  - Pace set by dependencies.

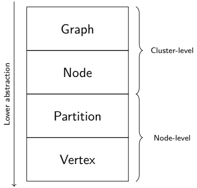
Solution

- Dependency tracking using partition meta-graph:
  - Always send messages to outgoing partitions at end of SS.
  - Receipt of message from P_Y tagged with SS S_X

    $\Rightarrow$ P_Y finished SS S_X.
- 2 execution layers:
  - Event layer
    - Bootstraps partition layer.
    - Global coordination with other nodes.
    - Determines end of event.
  - Partition layer
    - Performs actual computations.
    - Sends/Receives messages.
    - Progress determined by dependency tracking.

Vertex localsync — Introduction

- Partition-level localsync limited by connectivity.
- Another abstraction layer: vertices.
  - Local vertices.
  - Frontier vertices.
- Extend partition-level localsync.
  - Allow individual vertices to compute as their dependencies are met.
- Challenges:
  - Keeping overhead in check:
    - Complete vertex-level granularity too expensive.
    - Mix partition and vertex granularity.



Solution

- Up to 2 execution passes over a partition:
  1. Local + partial frontier.
  2. Remaining frontier.
- Which vertices execute in incomplete pass?
  - For each vertex scheduled to run:
    - Check source partition of incoming edges.
    - Check if all such partitions finished.

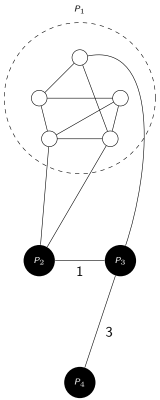
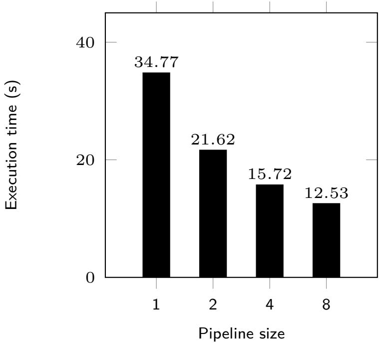
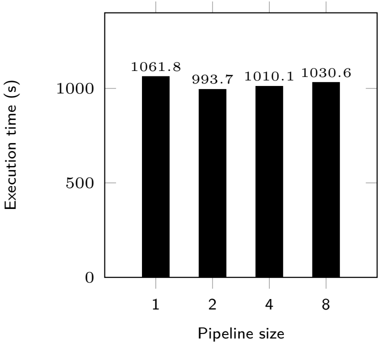


Table of Contents

Pipelining — Execution time vs pipeline size

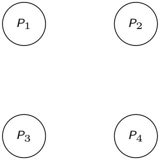


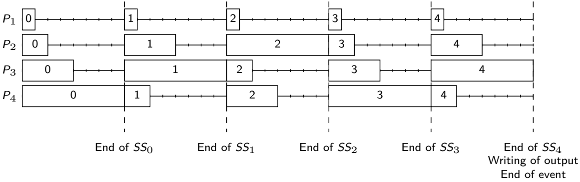
(a) PageRank



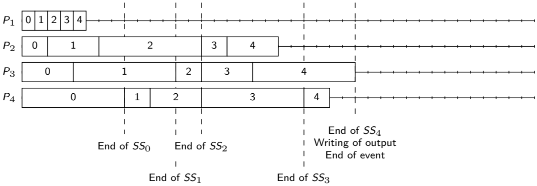
(b) Triangle Counting

Partition localsync — Disconnected components



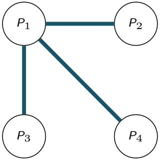


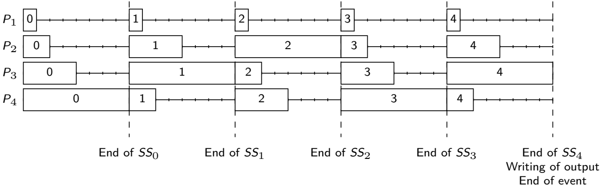
(a) Normal



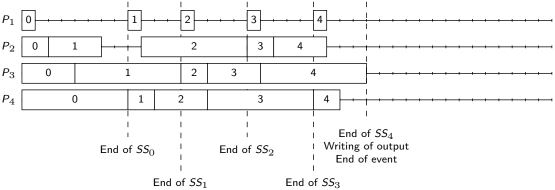
(b) Partition localsync (37.5% improvement + partial results)

Partition localsync — Sparse meta-graph



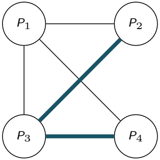


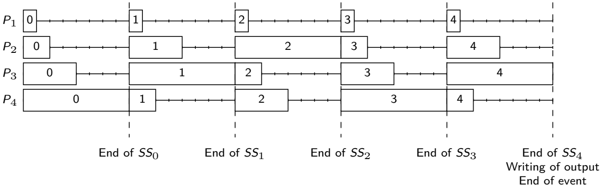
(a) Normal



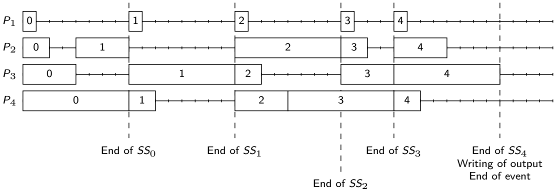
(b) Partition localsync (37.5% improvement)

Partition localsync — Dense meta-graph



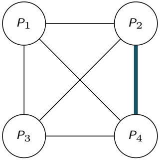


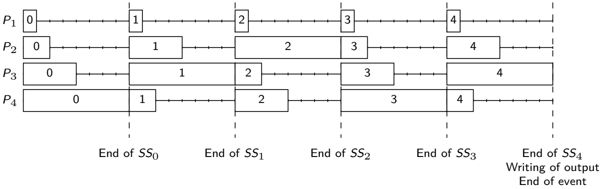
(a) Normal



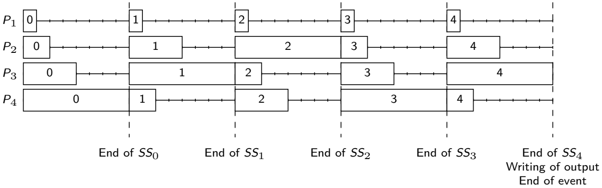
(b) Partition localsync (5.3% improvement)

Partition localsync — Completely connected meta-graph



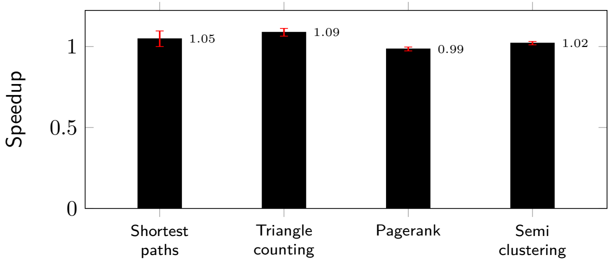


(a) Normal



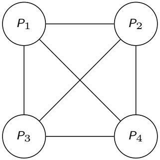
(b) Partition localsync (0% improvement)

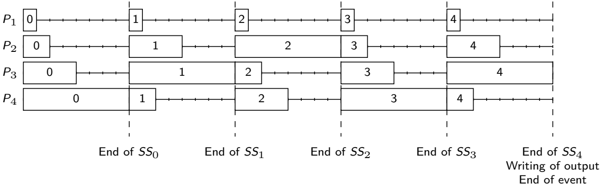
Partition localsync — Tuenti



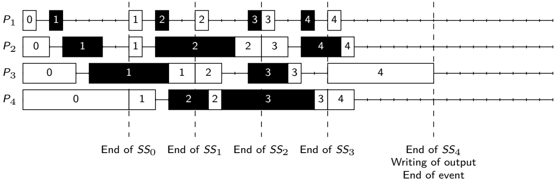
- Completely connected meta-graph.
- Prefetching effect.
  - Start reading & computation before complete coordination.
- No overhead.

Vertex localsync — Completely connected meta-graph





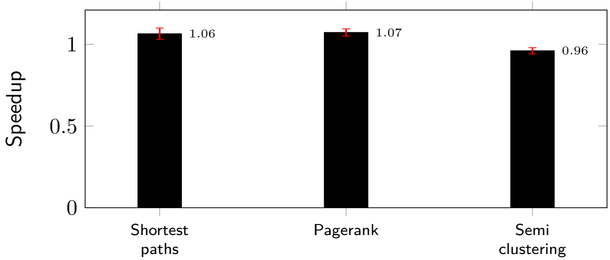
(a) Normal & Partition localsync



(b) Vertex localsync (16% improvement)

Vertex localsync — Tuenti with artificial skew

- Added extra variable skew to each vertex computation:
  - Similar to controlled skewed application.
  - Keep original computation & communication patterns.



- Overheads:
  - Memoized data merging.
  - Vertex dependency checking.

Table of Contents

Conclusion

- Optimize performance of Pregel by exploiting idle times.
- 3 mechanisms:
  - Event pipelining
    - Use idle times through parallelization.
    - Up to 137% improvement in latency.
  - Partition-level local synchronization
    - Reduce idle times.
    - Improvements with non-completely connected graphs.
    - Negligible overhead.
  - Vertex-level local synchronization
    - Further reduce idle times through partial passes.
    - Improvements even with completely connected graphs.
    - Further optimizations needed to reduce overhead.

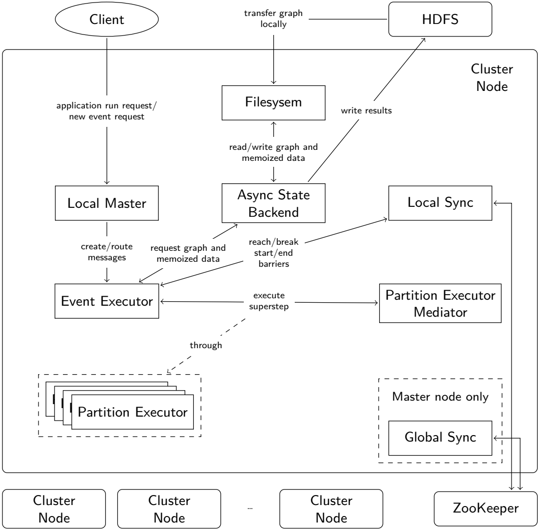
Conclusion

- Optimize performance of Pregel by exploiting idle times.
- 3 mechanisms:
  - Event pipelining
    - Use idle times through parallelization.
    - Up to 137% improvement in latency.
  - Partition-level local synchronization
    - Reduce idle times.
    - Improvements with non-completely connected graphs.
    - Negligible overhead.
  - Vertex-level local synchronization
    - Further reduce idle times through partial passes.
    - Improvements even with completely connected graphs.
    - Further optimizations needed to reduce overhead.

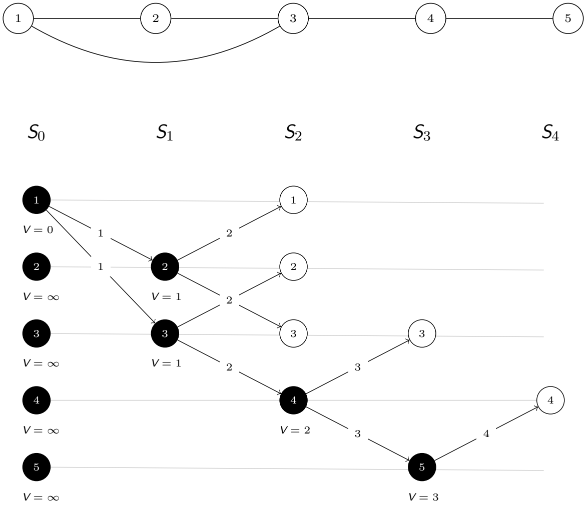
Questions??

Extra slides

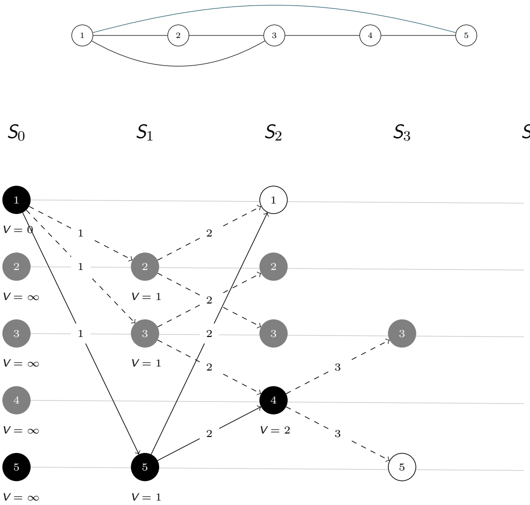
RTGiraph Architecture



Example — Event 0

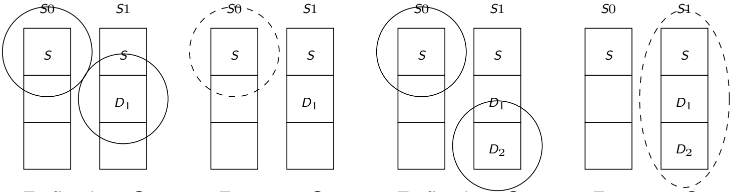


Example — Event 1



Memoization mechanism

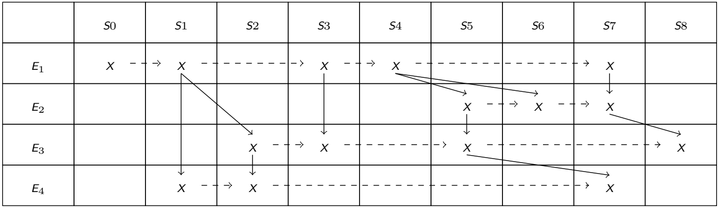
- Previous memoization mechanism (per partition):
  - One memoization file per superstep.
  - 1 read at beginning of superstep reads state and delta.
  - 2 writes at end of superstep:
    - State written to current superstep file.
    - Delta written to next superstep file.
- Problem — Correctness not ensured with parallel events:



E_1 finishes S_0 E_2 starts S_0 E_2 finishes S_0 E_1 starts S_1

Memoization mechanism

- New memoization mechanism (per partition):
  - One memoization file per superstep and per event.
  - 2 reads at beginning of superstep:
    - Read state from same or previous superstep of previous event.
    - Read delta from previous superstep of current event.
    - Smart index allows quick finding of right file.
  - 1 write at end of superstep writes state and delta to file of that event and superstep.

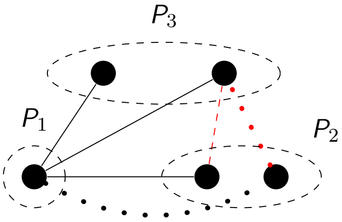


Memoization mechanism

- Advantages:
  - Focus on read IOs, usually faster than write IOs.
  - Allow gaps in memoized state and delta information.
    - If partition not involved in a superstep, no writing needed.
    - Previous mechanism required writing at each superstep.

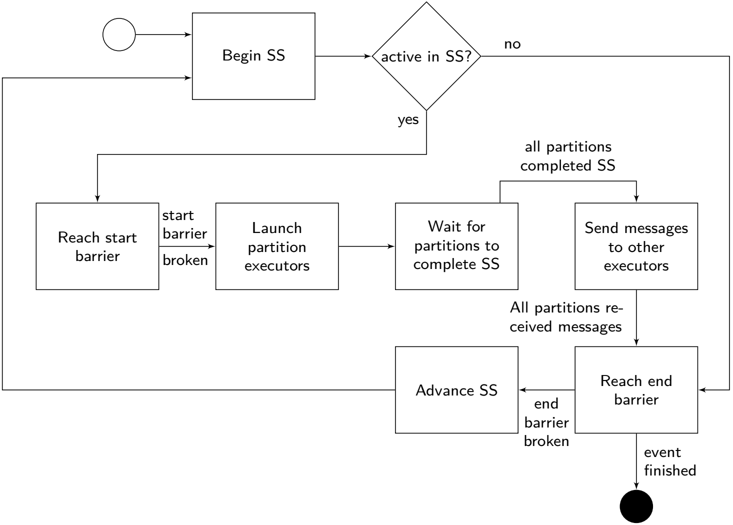
Partition meta-graph

- Meta-graph stored as a delta of connections.

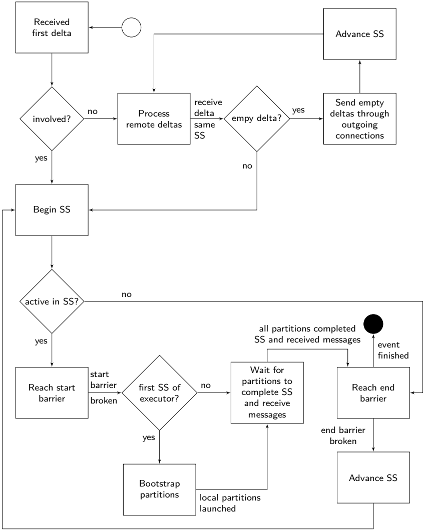


	P_1	P_3
E_0	1	2
ΔE_1		-1
ΔE_2	$+1$	-1
	$\downarrow$	
E_2	2	0
Conn. Set	$\{P_1\}$	

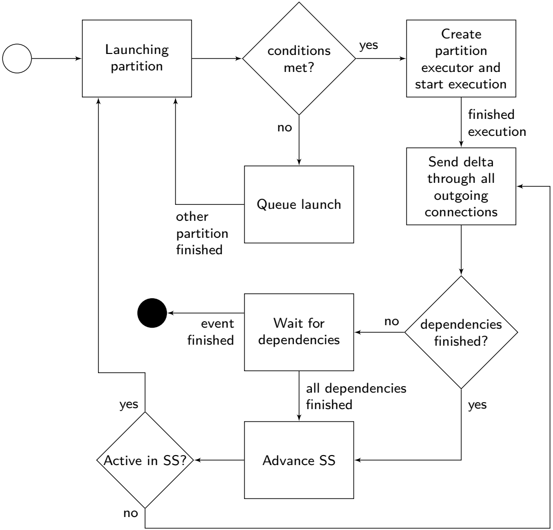
Non-localsync event executor



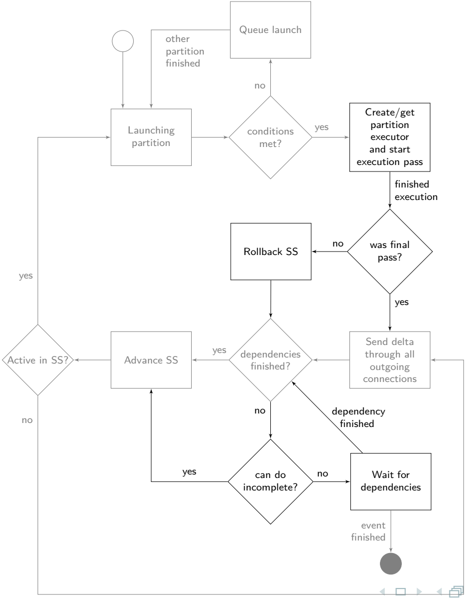
Localsync event executor — Event layer



Localsync event executor — Partition layer



Vertex local sync event executor — Partition layer



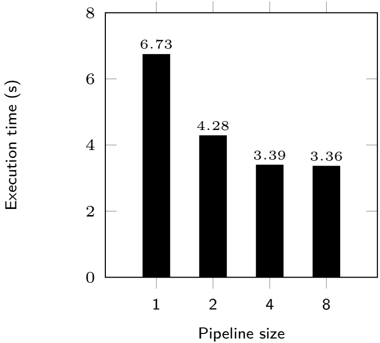
Environment

- Up to 21 Amazon EC2 r3.xlarge instances.
  - 4 vCPUs over Intel Xeon E5-2680 v2 CPUs.
  - 30.5 GB of RAM.
  - 80 GB of SSD.
- Datasets:
  - Artificial graphs.
  - Real-world — Snapshot of Tuenti social network.
    - 3.7M vertices.
    - 236M edges.
- Artificial and real-world applications.

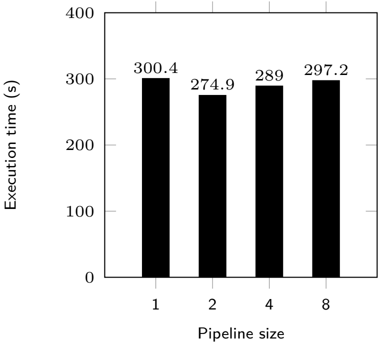
Tuenti statistics

	Min	Max	Avg	Total
Vertices	52,960	67,551	61,828	3,709,730
Local Vertices	410	1042	650	39,007
Edges	3,907,864	3,947,309	3,926,644	235,598,676

Pipelining — Execution time vs pipeline size

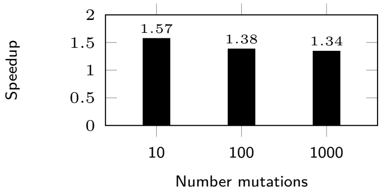


(a) SSSP

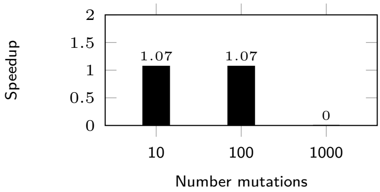


(b) Semi clustering

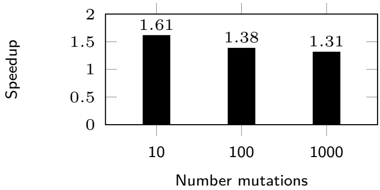
Pipelining — Speedup vs event size



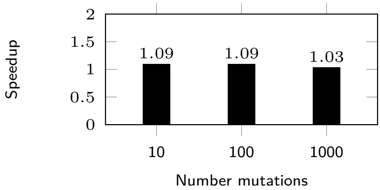
(a) SSSP



(b) Triangle Counting



(c) PageRank



(d) Semi clustering