TEZ means fast

Wojciech Indyk

@woj_i

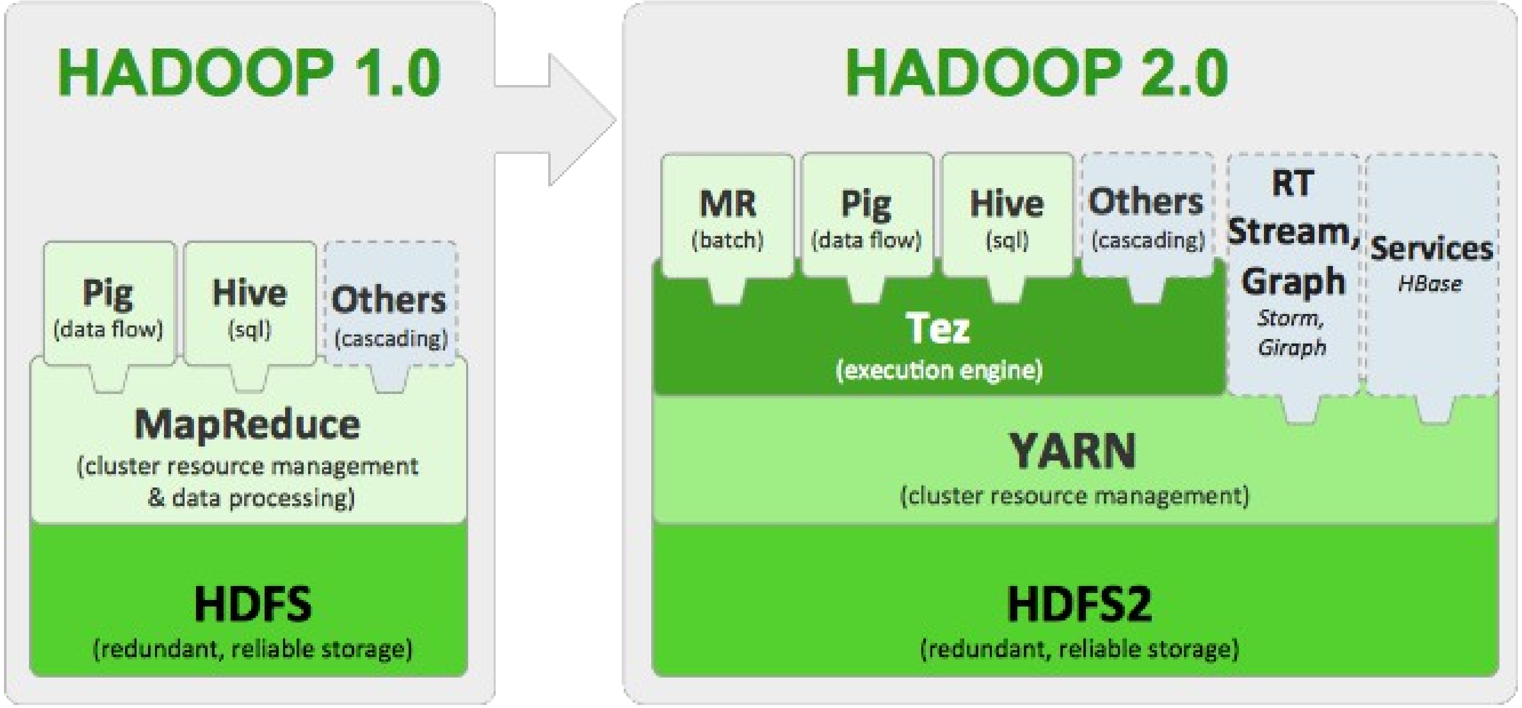
https://www.linkedin.com/in/wojciechindyk

Agenda

1. Introduction – evolution of Hadoop stack
2. Processing model- Directed Acyclic Graph
3. Architecture of TEZ
4. Example- MR implementation
5. Performance gain over MR- theory
6. TEZ Sessions
7. Benchmarks
8. Current status, next steps with TEZ

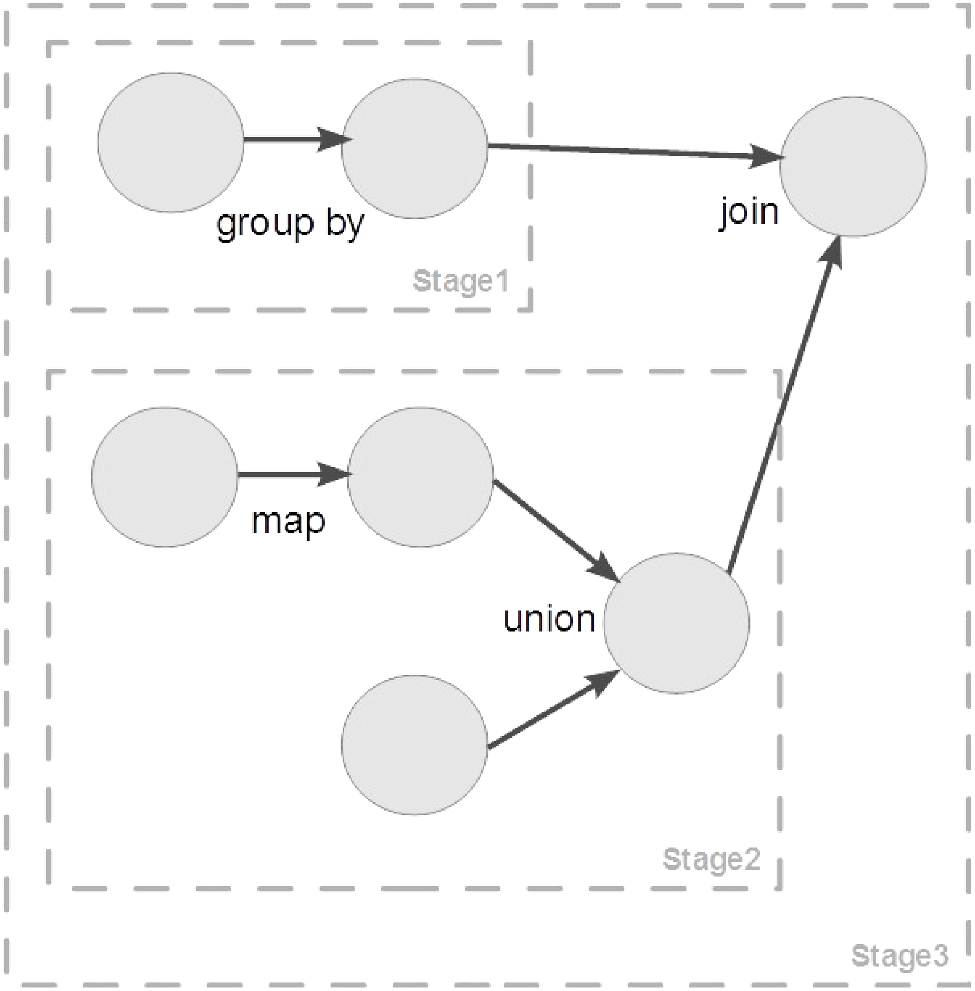
Introduction- evolution of Hadoop stack

- Decomposition of resource manager and computation engine (flexible processing approach)
- Why to interest in TEZ? It is the next generation of MapReduce execution engine



Directed Acyclic Graph

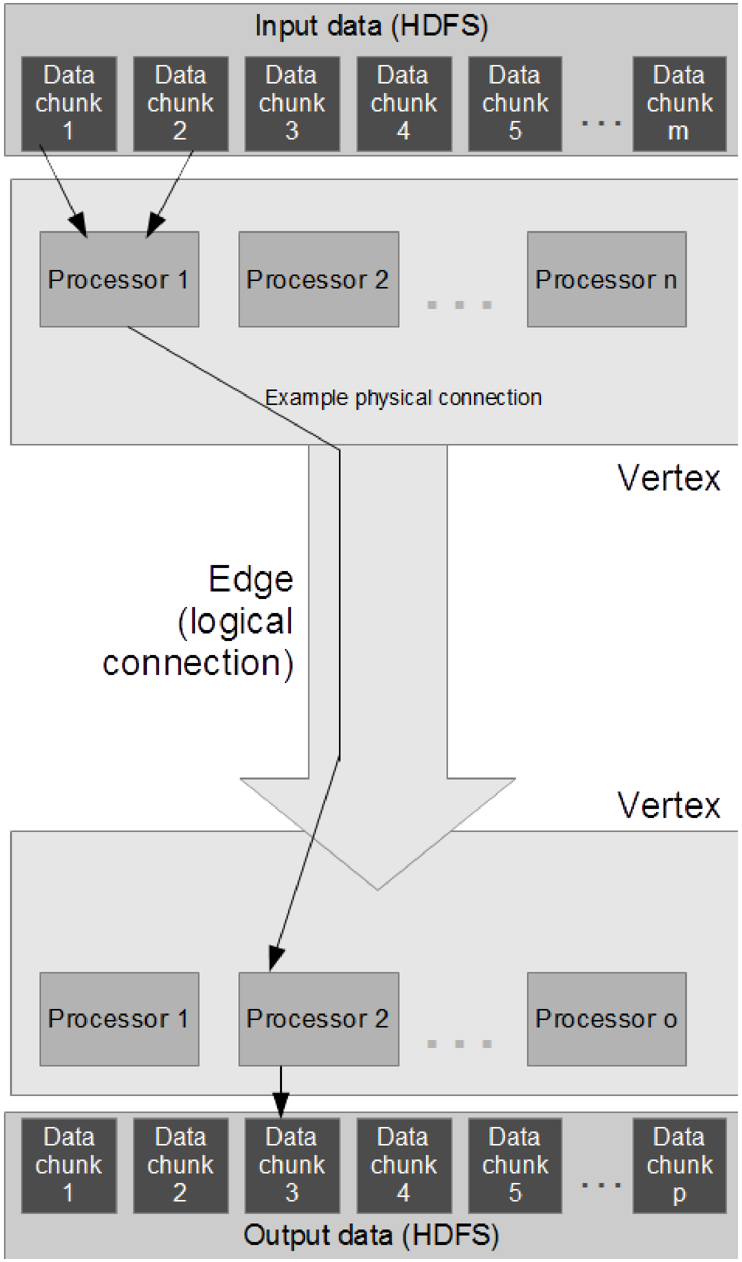
- Vertices and edges
- More elastic than MR
- Dedicated for complex data flow processing
- Intermediate phases can be stored in memory
- Optimizations of processing flow



TEZ Architecture

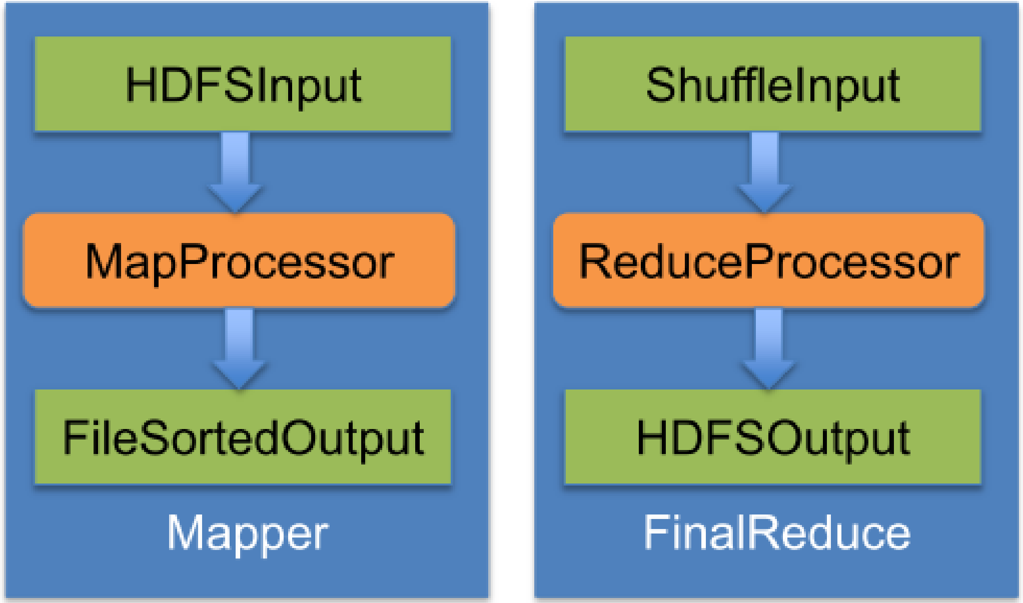
- Vertex as a data processor
- Edges connect vertices
  - Routing (1-1, broadcast, scatter-gather)
  - Scheduling (sequential, concurrent)
  - Data source (persisted, ephemeral)

  - e.g. MapReduce: scatter-gather, sequential, persisted



Example- MR implementation

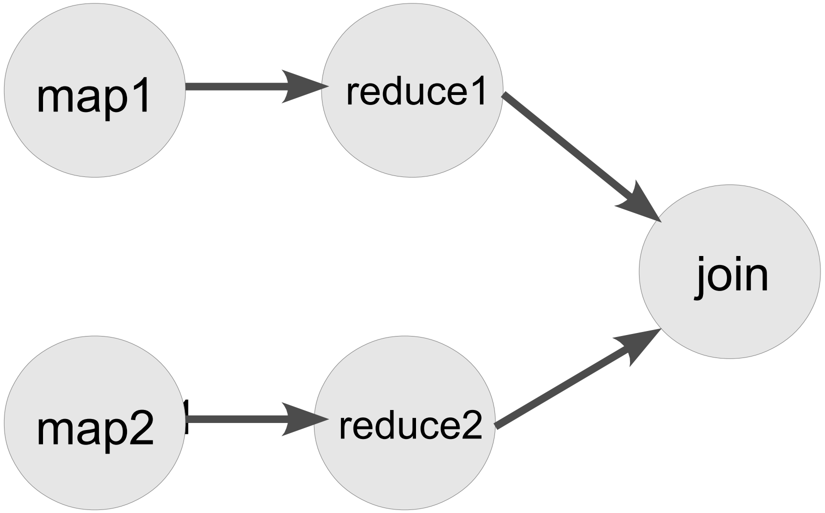
- Flexible Input-Processor-Output runtime model
- Data type agnostic
- API changes frequently... (TEZ 0.2-0.5) But the idea is still the same
- Let's see the code



ReduceProcessor code

```
public class ReduceProcessor extends MRTask implements LogicalIOProcessor {

  private static final Log LOG = LogFactory.getLog(ReduceProcessor.class);

  private Counter reduceInputKeyCounter;
  private Counter reduceInputValueCounter;

  public ReduceProcessor() {
    super(false);
  }

  @Override
  public void initialize(TezProcessorContext processorContext)
      throws IOException {
    ...
  }



  @Override
  public void handleEvents(List<Event> processorEvents) {
  }

  public void close() throws IOException {
  }

  @Override
  public void run(Map<String, LogicalInput> inputs,
      Map<String, LogicalOutput> outputs) throws Exception {
      THE CONTEXT OF PROCESSOR HERE
      …
      LogicalInput in = inputs.values().iterator().next();
      …
      ShuffledMergedInputLegacy shuffleInput = (ShuffledMergedInputLegacy)in; //data-type agnostic
  }
…
  @Override
      public void close(TaskAttemptContext context) throws IOException,
      InterruptedException {
      }
```

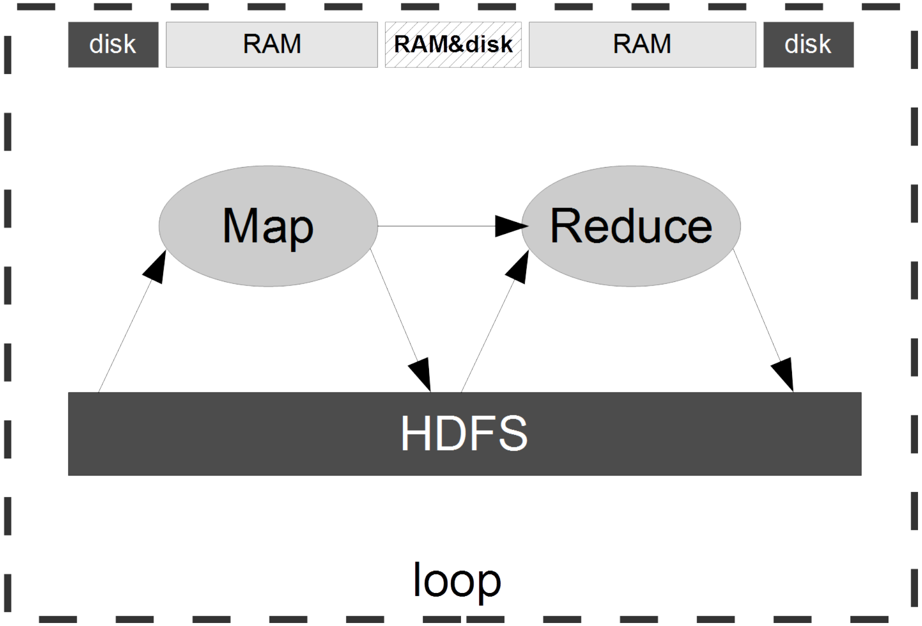
Create DAG



```
DAG dag = new DAG();

    Vertex map1 = new Vertex(MapProcessor.class);
    Vertex map2 = new Vertex(MapProcessor.class);
    Vertex reduce1 = new Vertex(ReduceProcessor.class);
    Vertex reduce2 = new Vertex(ReduceProcessor.class);
    Vertex join1 = new Vertex(JoinProcessor.class);

...

     Edge edge1 = Edge(map1, reduce1, SCATTER_GATHER,
PERSISTED, SEQUENTIAL, MOutput.class, RInput.class);
    Edge edge2 = Edge(map2, reduce2, SCATTER_GATHER, PERSISTED, SEQUENTIAL,
MOutput.class, RInput.class);
    Edge edge3 = Edge(reduce1, join1, SCATTER_GATHER, PERSISTED, SEQUENTIAL,
MOutput.class, RInput.class);
    Edge edge4 = Edge(reduce2, join1, SCATTER_GATHER, PERSISTED, SEQUENTIAL,
MOutput.class, RInput.class);

...

     dag.addVertex(map1).addVertex(map2)
  .addVertex(reduce1).addVertex(reduce2)
  .addVertex(join1)
  .addEdge(edge1).addEdge(edge2)
  .addEdge(edge3).addEdge(edge4);
```

Performance gain over MR

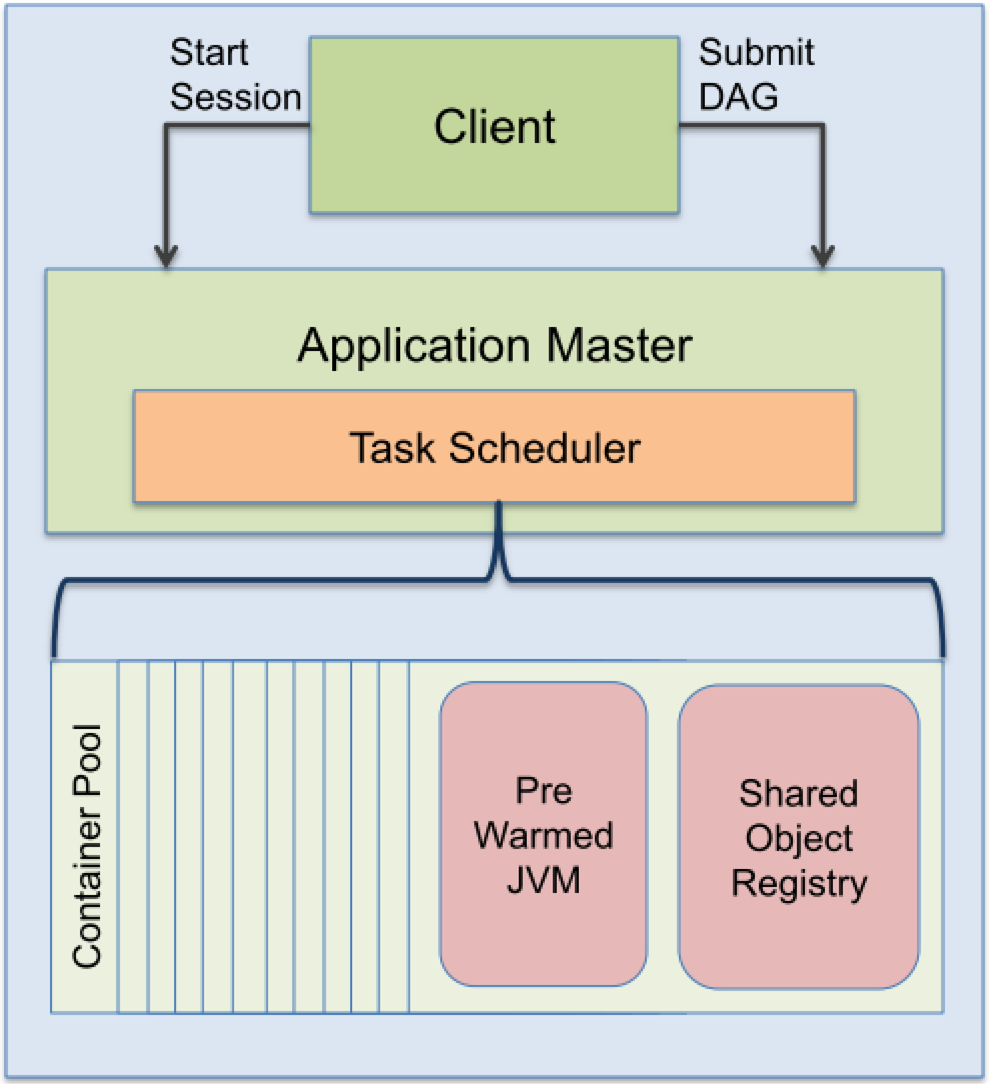
- Eliminate replicated write barrier between successive computations.
- Eliminate job launch overhead of workflow jobs.
- Eliminate extra stage of map reads in every workflow job.



TEZ Sessions

- One Application Master
- Re-use containers
- caching

```
tezSession = new
TezSession("PageRankSession",
           appId, sessionConfig);
tezSession.start();
…
tezSession.submitDAG(dag);
…
tezSession.stop();
```



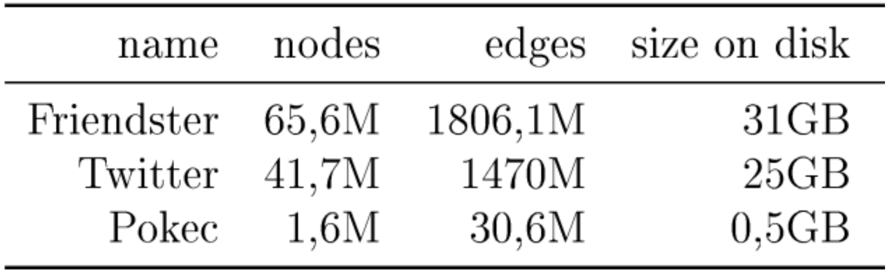
When to use sessions

- Analytic tasks on the same data (e.g. drill down)
- Data flows consisted with many sequential tasks
- But also by iterative jobs
  - Optimization of JVM for the same task
  - Near the same resource utilization over iterations

- The resources (memory, CPU) of the AM are fixed so please keep this in mind when configuring the AM for use in a session. For example, memory requirements may be higher for a very large DAG.

Experiment

- Algorithms:
  - Single Source Shortest Paths
  - PageRank
- Datasets:

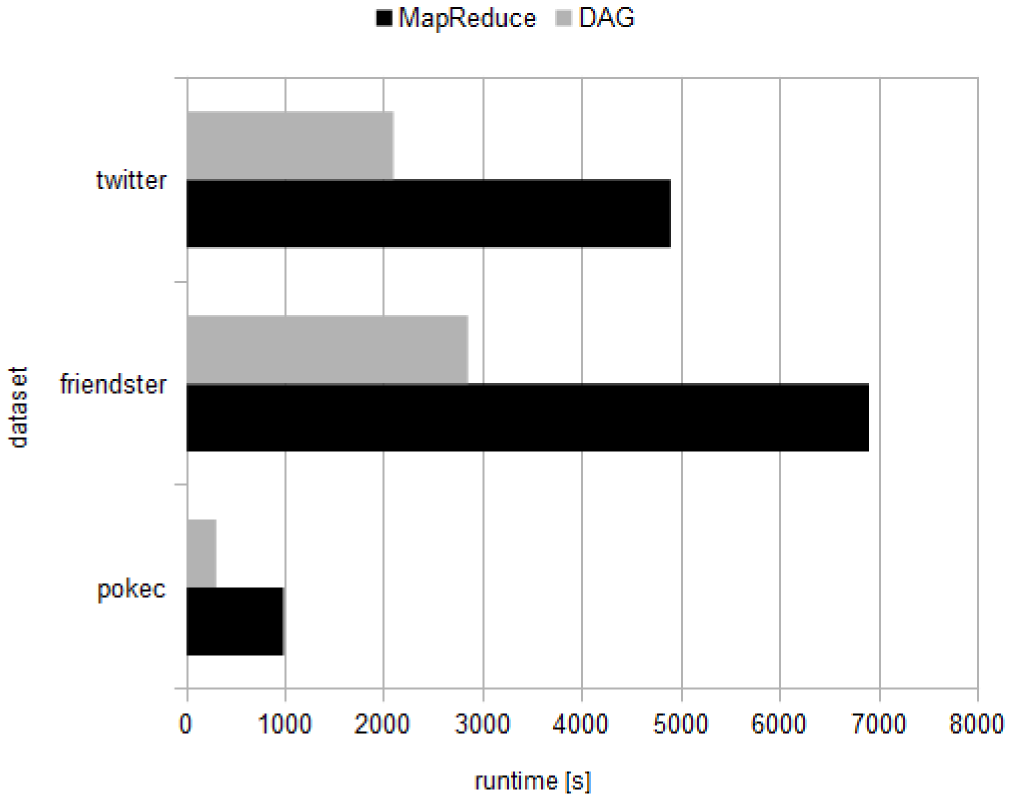


name	nodes	edges	size on disk
Friendster	65,6M	1806,1M	31GB
Twitter	41,7M	1470M	25GB
Pokec	1,6M	30,6M	0,5GB

Environment:

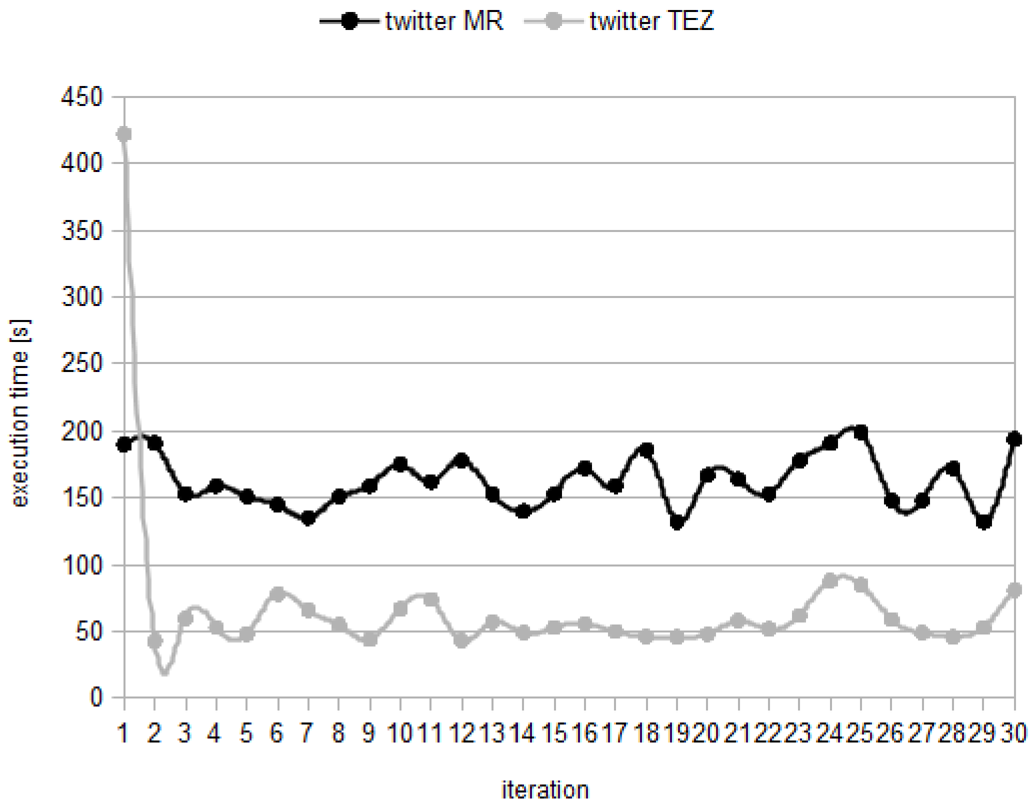
- 14 servers, Hadoop Hortonworks 2.1

Results- PageRank overview

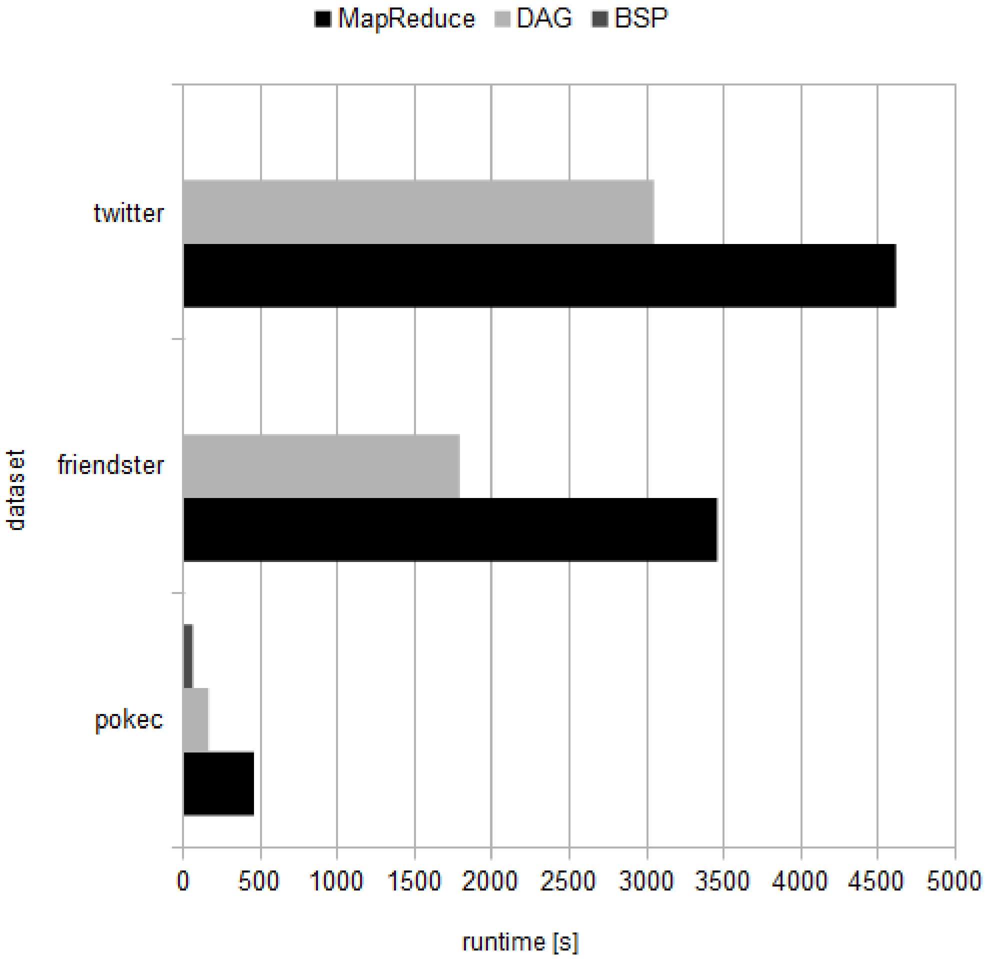


Results- PageRank for Twitter dataset

- Stable iteration results, using session (1-2 iteration) for TEZ is visible

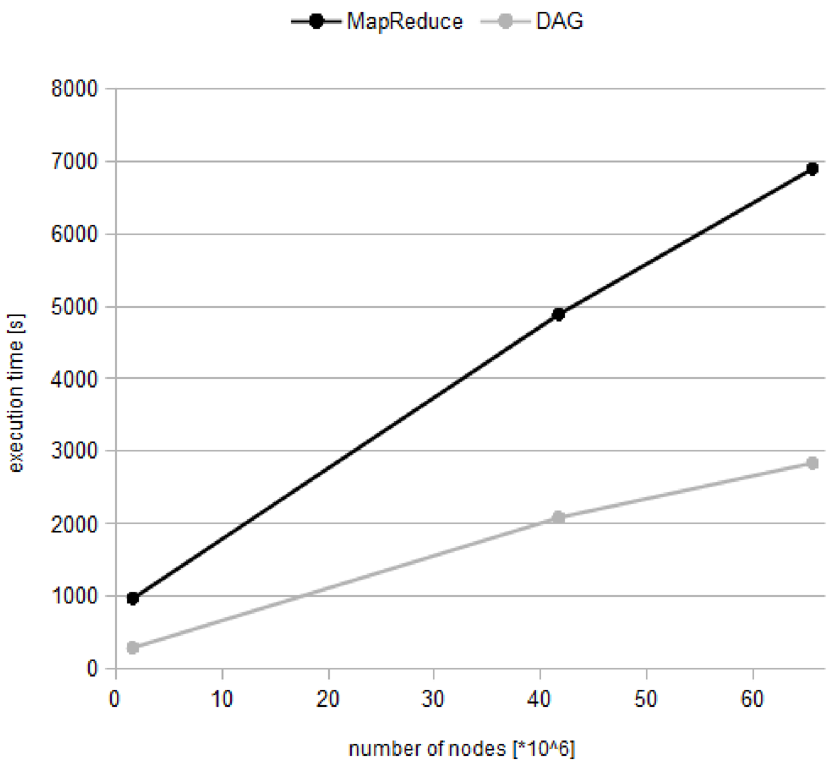


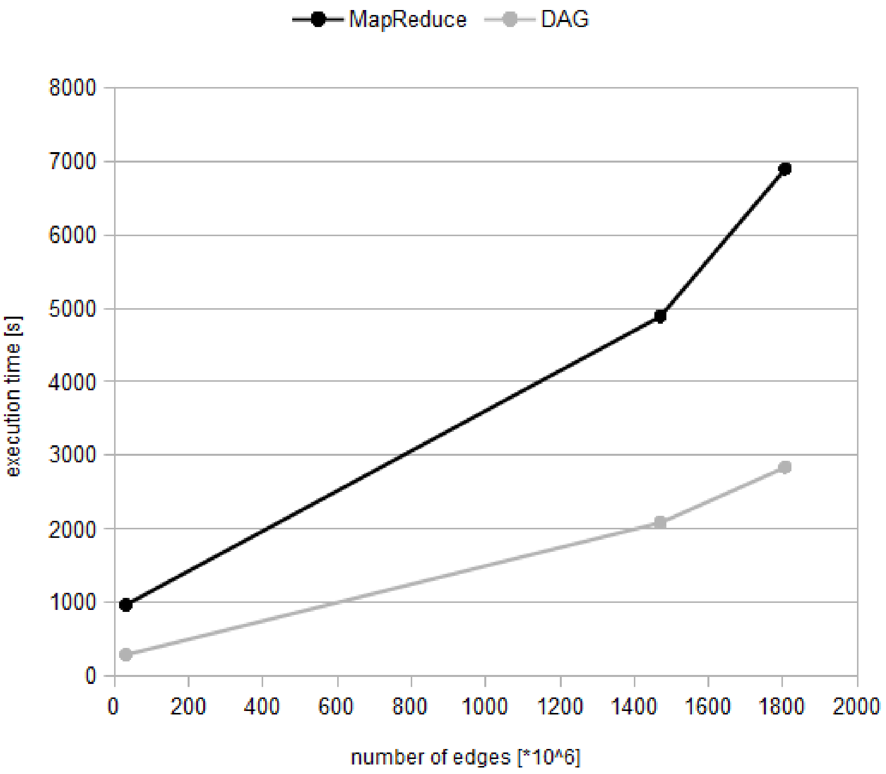
Results Single Source Shortest Paths overview



Results- Scalability of PageRank

- Near linear data-scalability, better scalability for TEZ than MapReduce (in terms of both edges and nodes),





Current status of TEZ

- Still in incubation phase
  - 4.07 – Decision of graduation to „top-level” project
- Development focused on stability of TEZ
- And richer DAG support
  - Better fault tolerance with checkpoints

Difference from Spark

- TEZ focused on large data rather than fast RAM iterations
- Oriented on support for Hive
- IMO TEZ is something between Spark and Impala
  - Oriented on Hive analytic jobs performance
  - Not only for queries
  - Support for complex flows as well as Spark

How to run TEZ immediately

- You can replace MR execution engine with TEZ
- In Hadoop configuration change execution engine „yarn” => „tez-yarn” (at least for Hortonworks)
- Just launch your MapReduce job (it will launch on TEZ)
  - Counters not implemented
  - Worse performance than pure TEZ job

Next steps with TEZ

- Use sandbox! :)
- Create your own DAG
  - At the beginning you can base on Wordcount example

http://lawkarezerwowych.blox.pl/2007/05/Beznadziejny-nawal-pracy.html

- Stinger
  - Delivered with Hive 0.13
    - http://hortonworks.com/labs/stinger/

Thank you!

- Questions?
- Comments?
- Ideas?
- Blames? :)