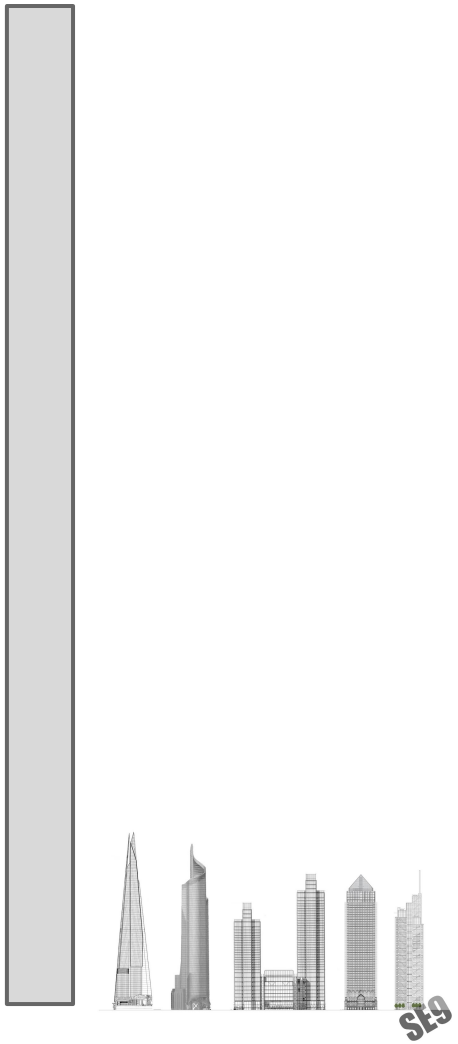
Imperial College London

Dron - an Integration Job Scheduler

Ionel Corneliu Gog

The big data problem

- Facebook Data Warehouse (100 PB)
- 120 MB/user
- 2048 meters of 1 TB HDDs
- Google processes 20 PB/day (2008)

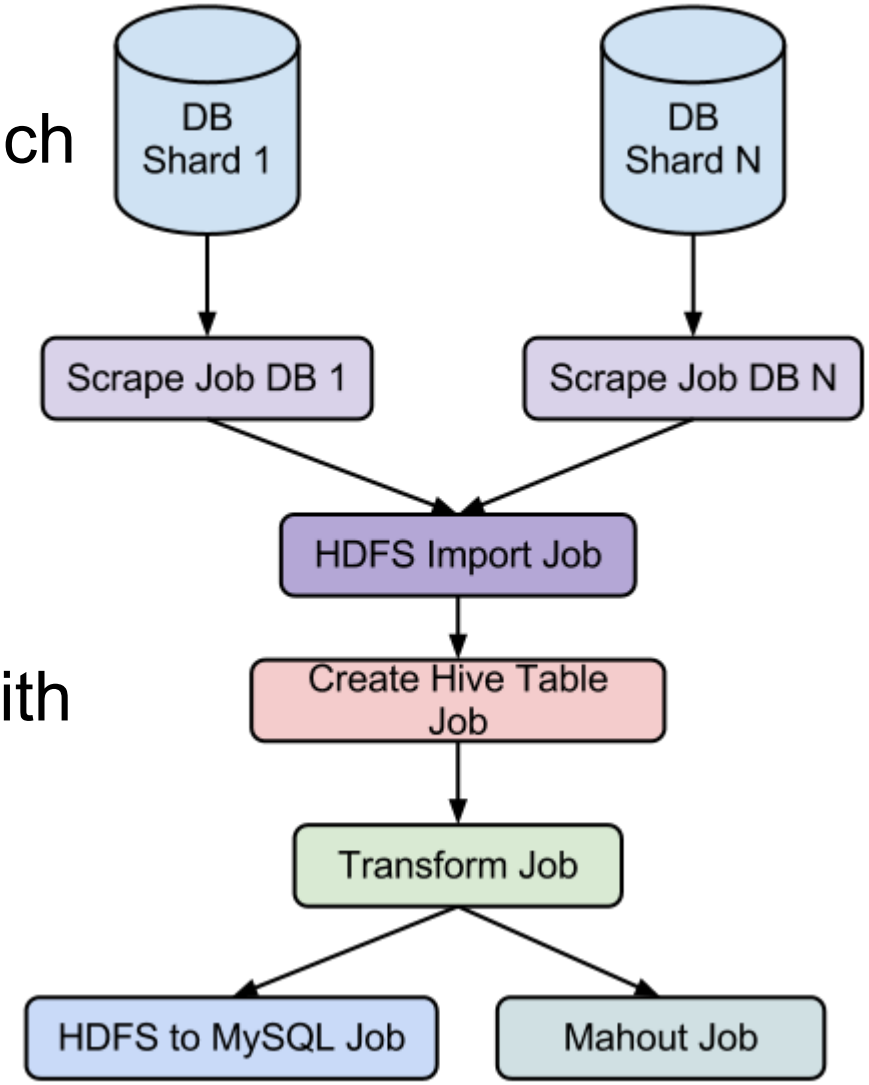


How do they do it?

- **MapReduce** / **Hadoop -** programming model and distributed framework
- **Google File System** (GFS) / **Hadoop File System** (HDFS)
- **BigTable** / **HBase** - data storage systems
- **Sawzall** / **Pig** / **Hive -** languages for data analysis
- **Mahout -** machine learning library
- **Spark**, **HaLoop**, **Shark**...

Problem \ Facebook use case

- No easy way to create such workflows
- Setup own cron jobs
- Handle communication with frameworks
- Difficult to handle failures



Dron

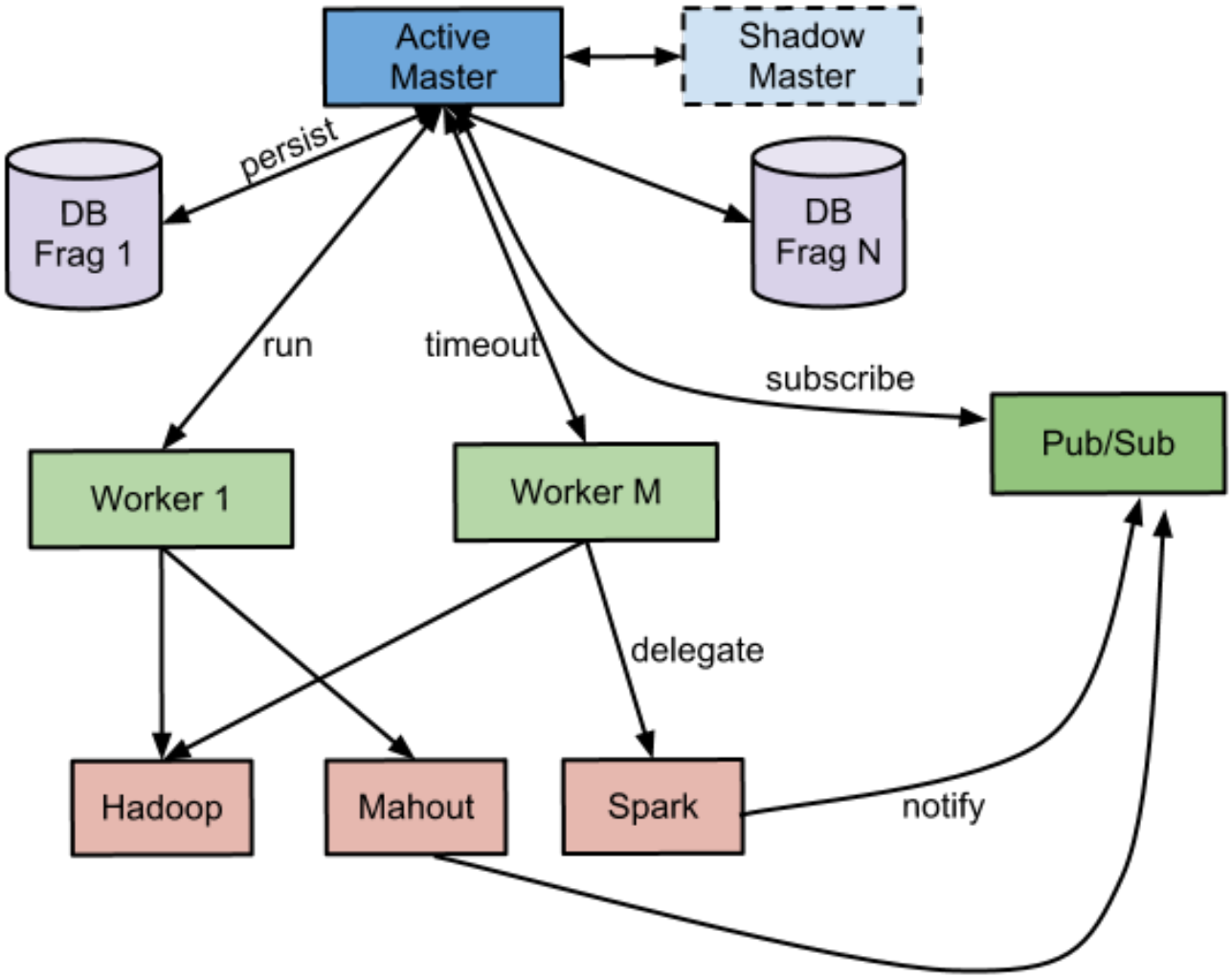
- Scalable job scheduler (thousands jobs / second)
- Possibility of building workflows
- Easy to integrate and use many frameworks
- Fault tolerant

Dependencies Language

```
dron_api:register_job(#job{name="scrape_users",
                           cmd_line="sh scrape_users.sh",
                           start_time={{2012,06,23},{12,0,0}},
                           frequency = 3600,
                           timeout = 600,
                           max_retries = 3,
                           dependencies = [],
                           deps_timeout = 10}).

dron_api:register_job(#job{name="scrape_query",
                           cmd_line="hive -e 'select users.name from table
                               users'",
                           start_time={{2012,06,23},{12,1,0}},
                           frequency = 3600,
                           timeout = 600,
                           max_retries = 3,
                           dependencies = [{"scrape_users", {hour, {2, 1}}}
                               ],
                           deps_timeout = 3600}).
```

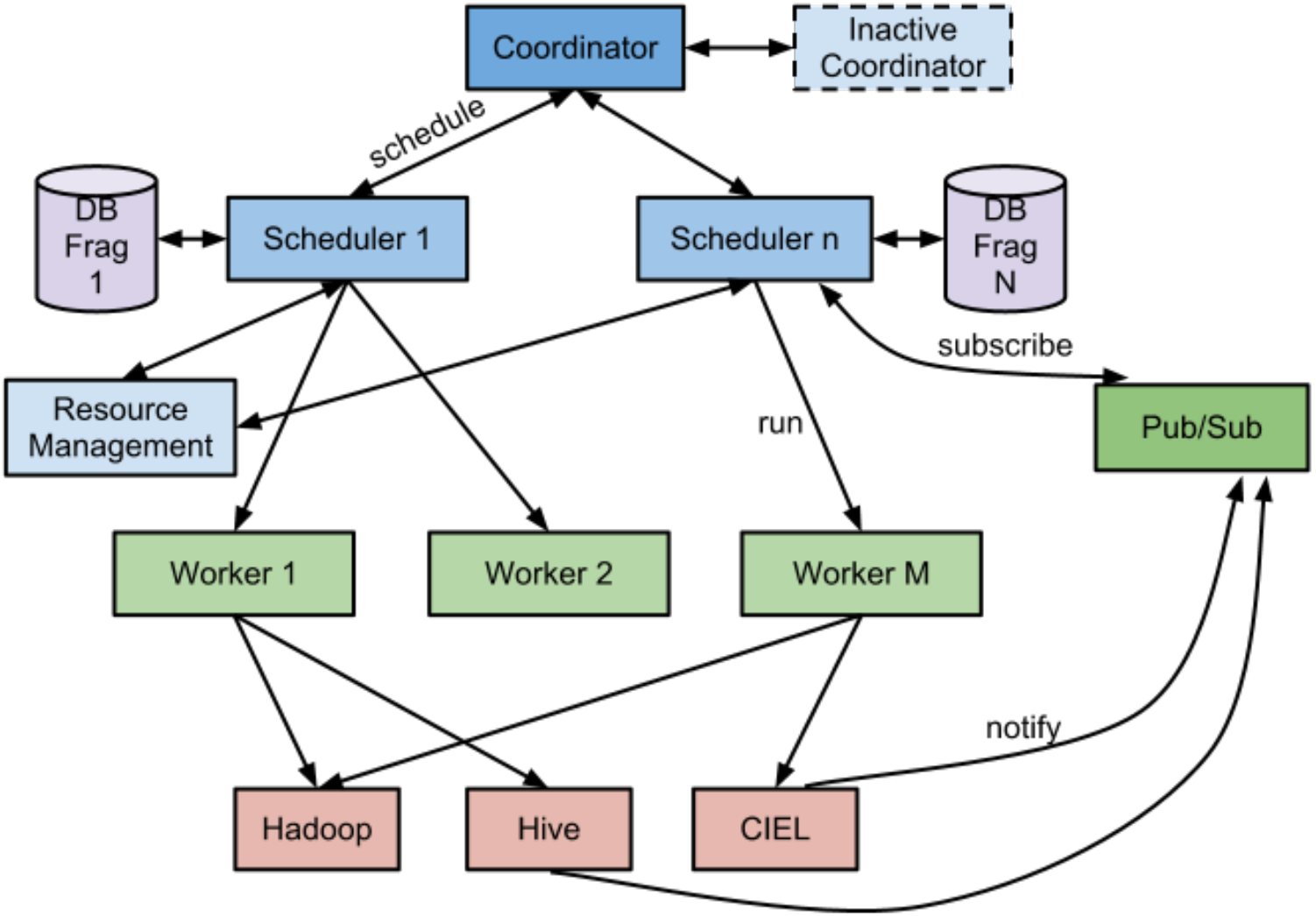
First Architecture



Bottlenecks

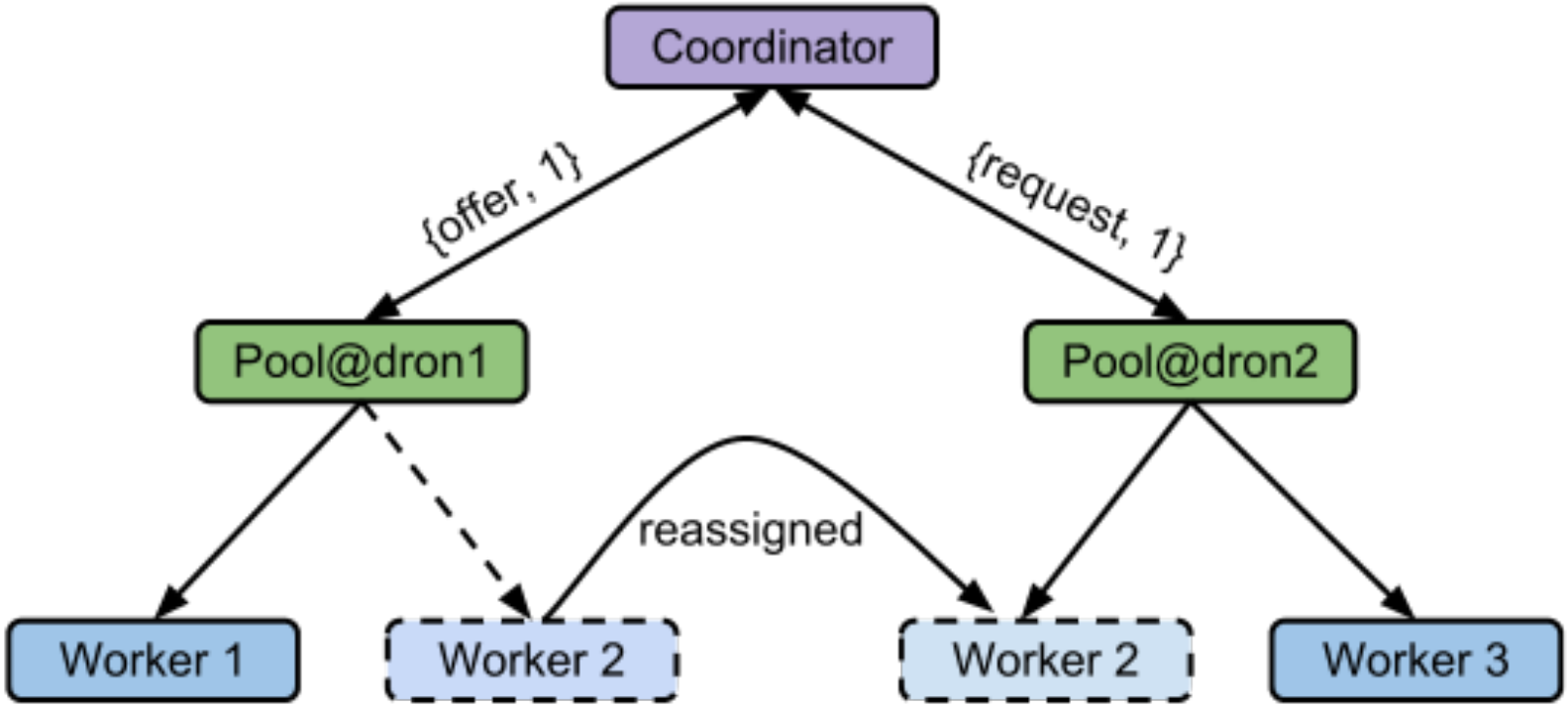
- Initially, around 100 jobs / second
- Reduced number of database transactions
- Tweaked Erlang Runtime
- Moved data about workers into memory
- Finally, at around 500 jobs / second the load on the machine was too high

Second Architecture



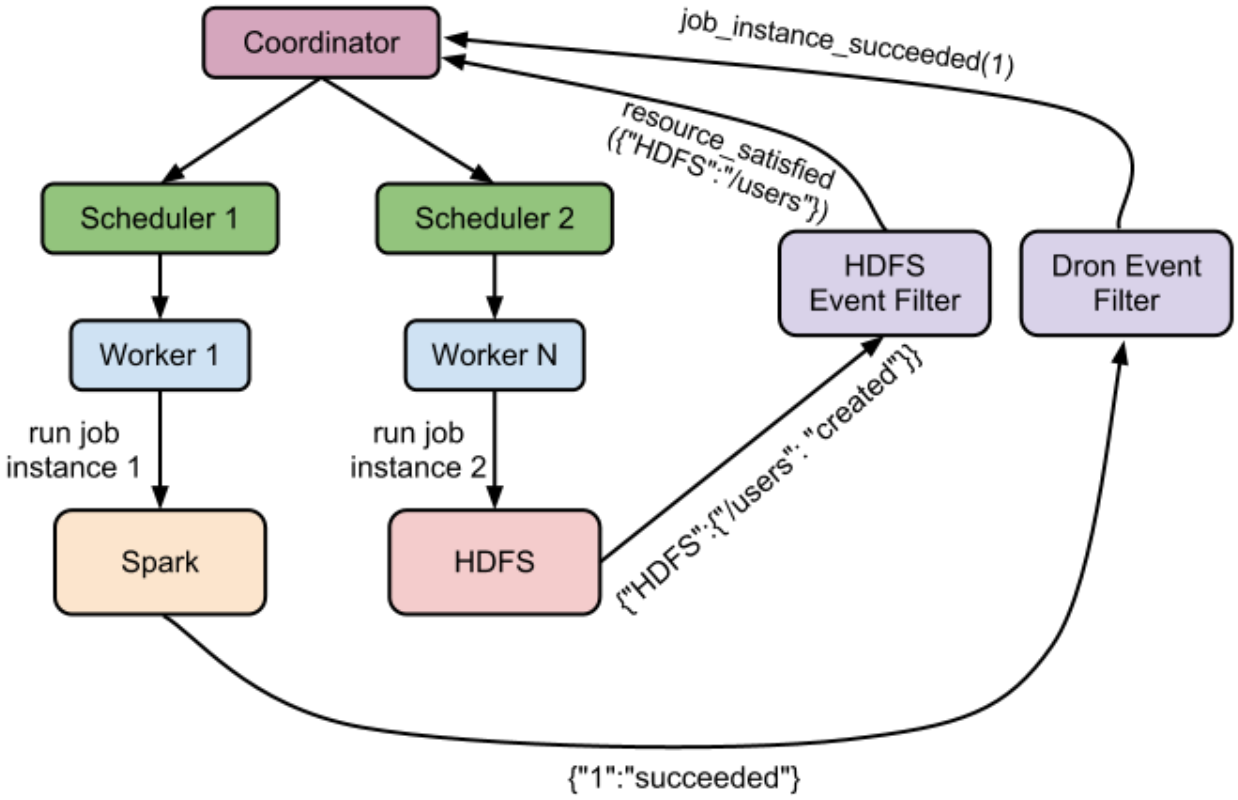
Workers Balancing

- Schedulers can have different sharing policies



Framework Integration

- Adapted MapReduce, Spark, Mahout



Evaluation

- How does it compare with other workflow managers?
- How is the scheduling delay affected as the number of jobs is increased?
- How many jobs can be run at the same time?
- Can we use the job dependencies data to improve scheduling?

Dron versus Oozie

- **Scalability**
  - **Oozie -** 3.57 jobs / sec
  - **Dron -** 370 jobs / sec

- **Scheduling Delay**

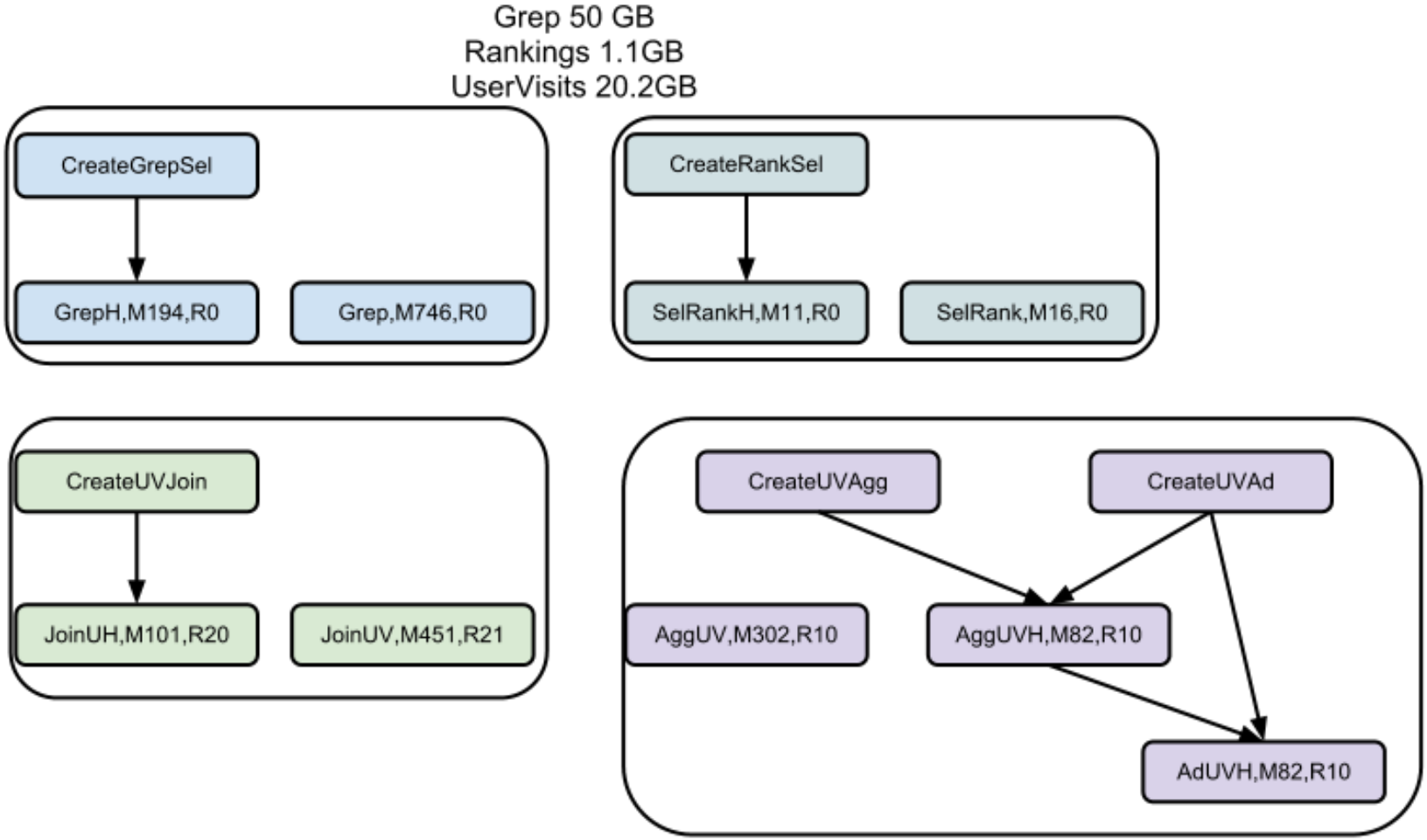
Number Jobs	Oozie (sec)	Dron (sec)
100	15.33	0.78
1 000	68.28	0.35

Dron Scalability

- **Scheduling delay**
  - 4 scheduler and 2 worker nodes
  - Under 2 sec for loads < 850 jobs / sec
  - Exponential increase on loads > 850 jobs / sec
- **Jobs running at the same time**
  - *"sleep 600"* jobs
  - 1 scheduler and 6 worker nodes
  - Exhausted job slots (150 000)

Benchmark

- 21 nodes Hadoop cluster

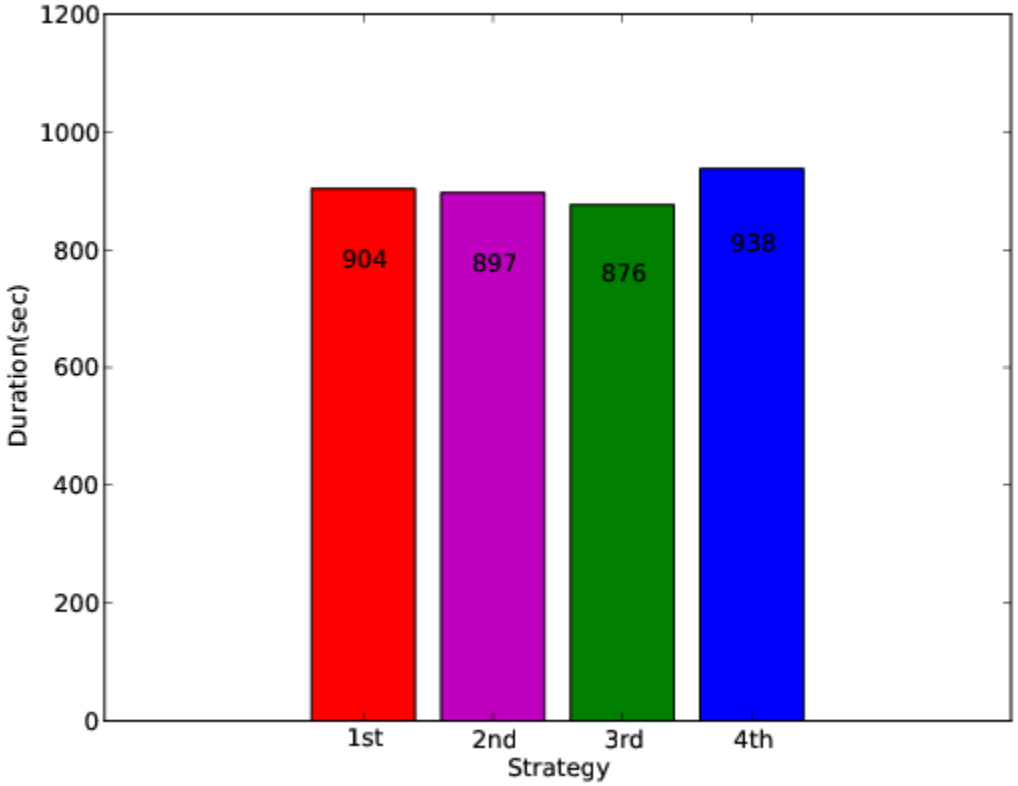


Benchmark Strategies

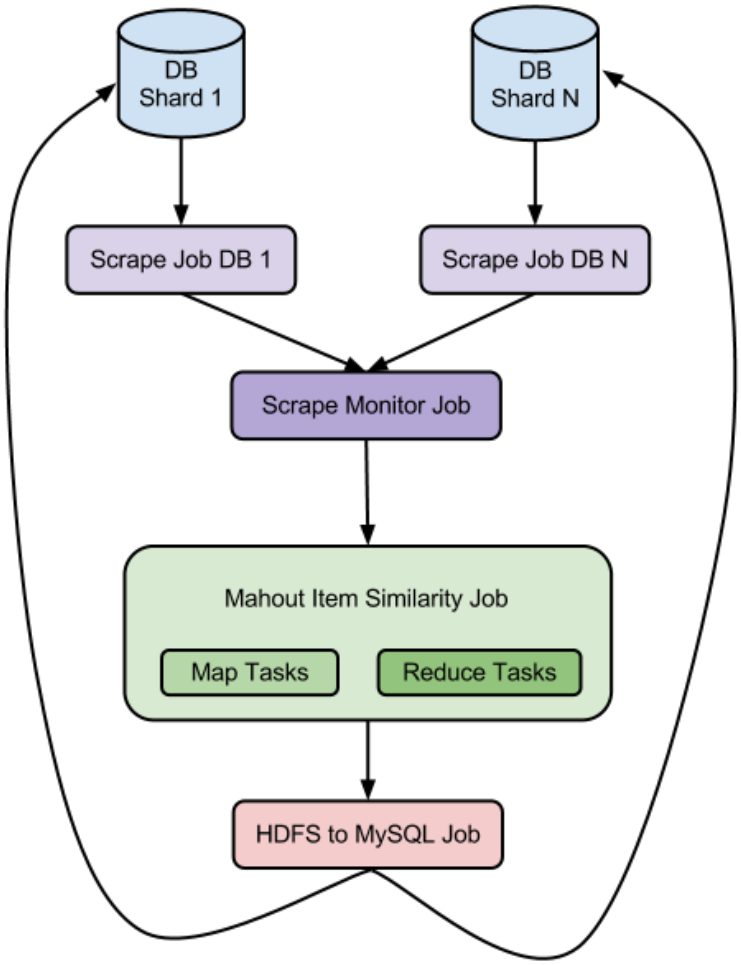
- **1st Strategy** - register all jobs at the same time
- **2nd Strategy** - prioritize jobs with highest acyclic graph depending on them
- **3rd Strategy** - extended 2nd strategy with knowledge about intermediate job steps
- **4th Strategy** - prioritize only a part of the jobs that have dependants

Benchmark Results

- 3rd strategy 6.6% faster than 4th
- No significant resource utilisation differences (except 4th)



Recommendation Engine



Further Work

- Extend dependency language
- Improve scalability
- Refine scheduling
- Resource based scheduling
- Data locations and provenience
- Improve user experience

Questions