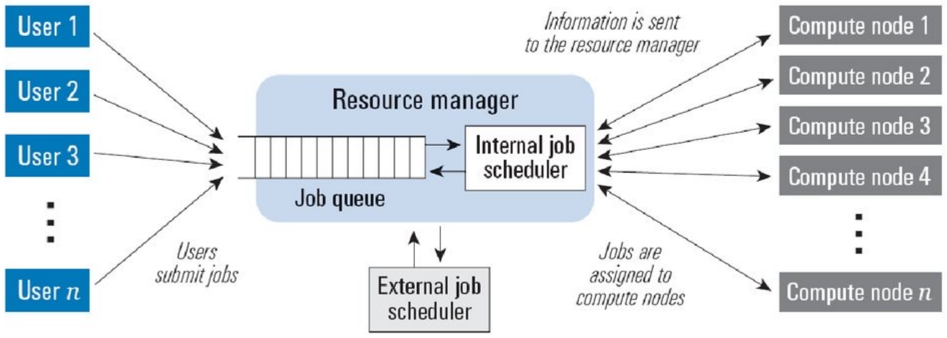
Study of scheduling approaches for batch processing in Big Data cluster

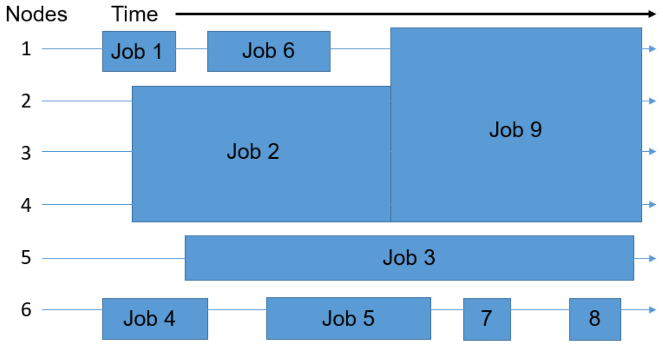
Ilya Timokhin – is.timokhin@hse.ru (Higher School of Economics),
Aleksey Teplov – aleksey.teplov@huawei.com (Advanced Software Technologies Laboratory of Huawei Moscow Research Center)

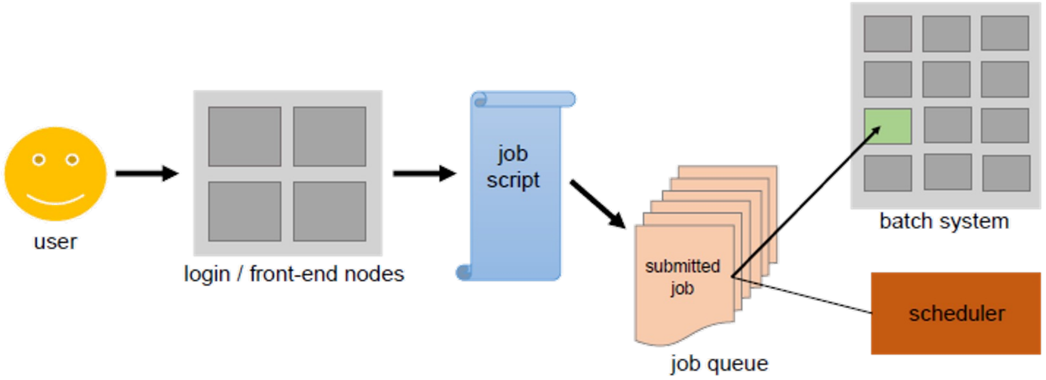
HUAWEI TECHNOLOGIES CO., LTD.

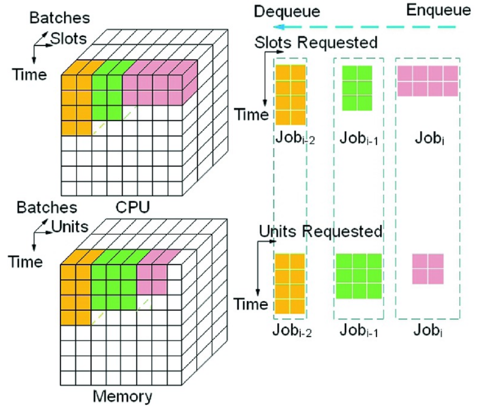
Background

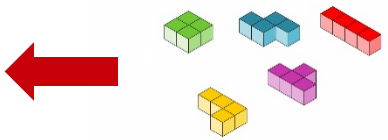
- Scheduling and resource allocation for the tasks are basic operations for distributed systems task execution with typical goal to increase resource utilization rate;
- The main concept of the scheduling approach is to spread processing tasks between available hardware resources and define the execution order;
- Depending on the application usage scenario and goals processing strategies can also vary and use different approaches to schedule tasks and allocated resources for internal logic;
- Different scheduling goals require different metrics optimizations that reflect how efficient scheduling approach is.





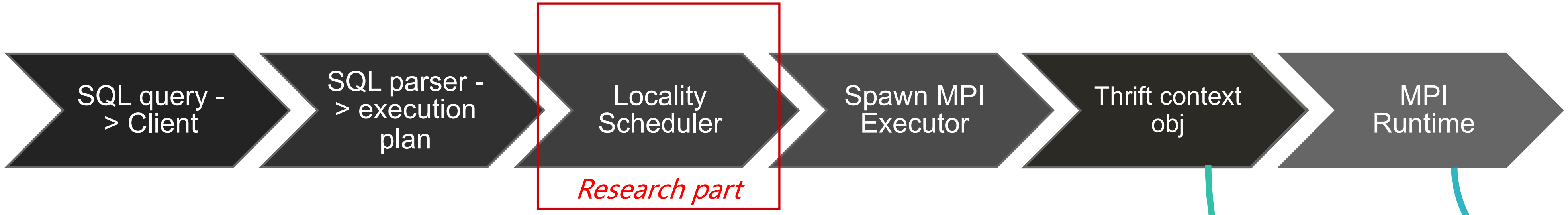




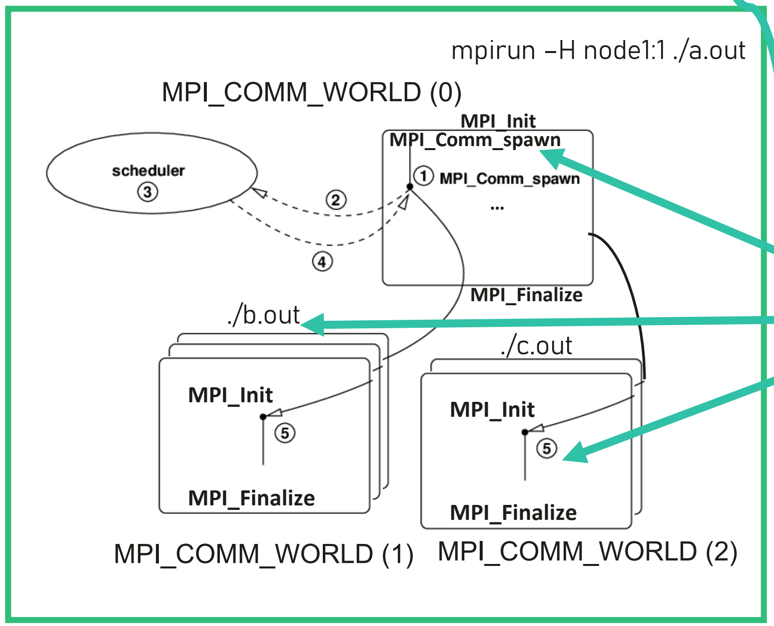


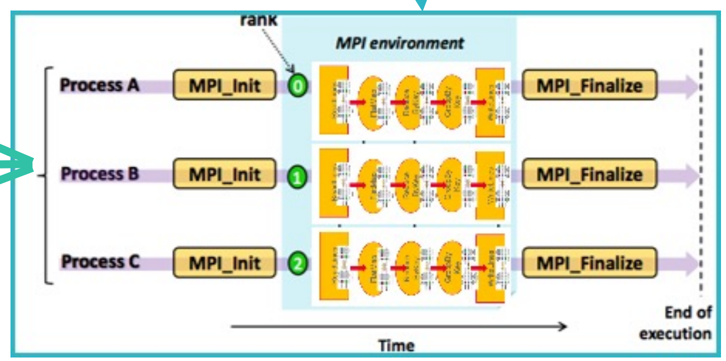
HUAWEI

Our framework for Big Data batch computations



- This framework combines MPI distributed approach with high-performance APIs to provide fast batch calculations.
- It use Thrift to provide SQL context and HDFS to store the data;
- It supports client/executor/SQL parser modules for simultaneously batch processing;
- **But the problem of data location in HDFS and optimal number of workers for MPI batch is still open!**

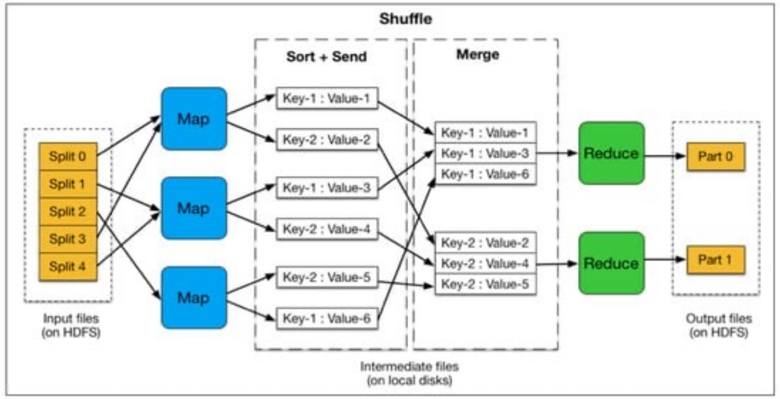


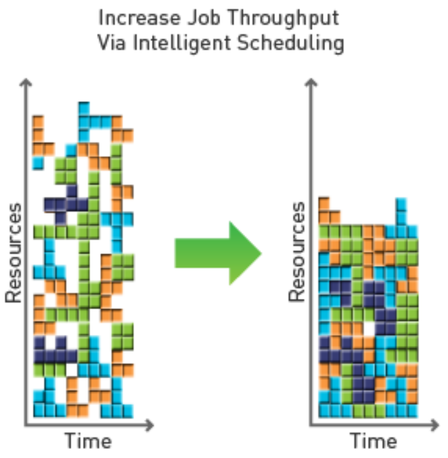


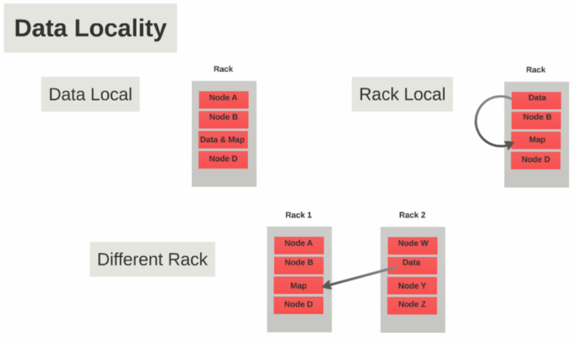
Data locality principle

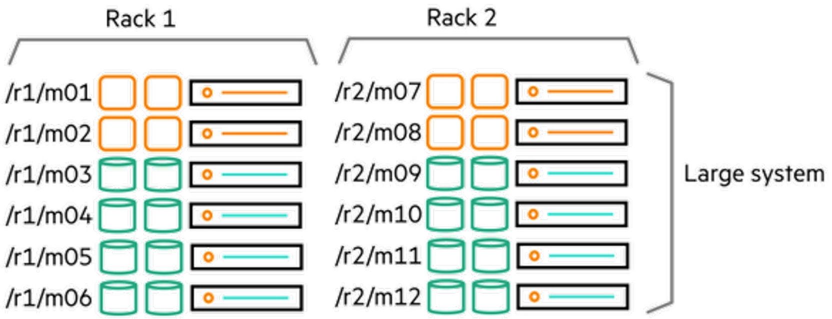
Data movement is costly operation that is better to avoid in distributed systems; Hadoop-like data storage provide computations and data on the same distributed HQ cluster.

1. ***Data-to-blocks:*** the whole system is designed to store incredibly large files (>10GB) across different machines (nodes) in a cluster. HDFS stores and split each file as a sequential set of blocks;
2. ***Computing-to-data:*** we should bring the computing to the data instead bringing the data to the computing as another file storages;
3. ***Fault tolerance:*** If any node get down the data continues processing in the other copies (replications) and the part of failing computations should be restarted in one of "health" nodes;
4. ***Replica parametrization:*** The user can set any discrete value for the replica factor (number of data copies) and the block size. Default value in HDFS setting file for replica factor is 3, each block has size of 128MB;
5. ***Nodes topology/roles:*** NameNode (master), Secondary NameNode (contains the latest file system changes), DataNode (stores blocks of files);
6. Optimized to support ***high-streaming read***.

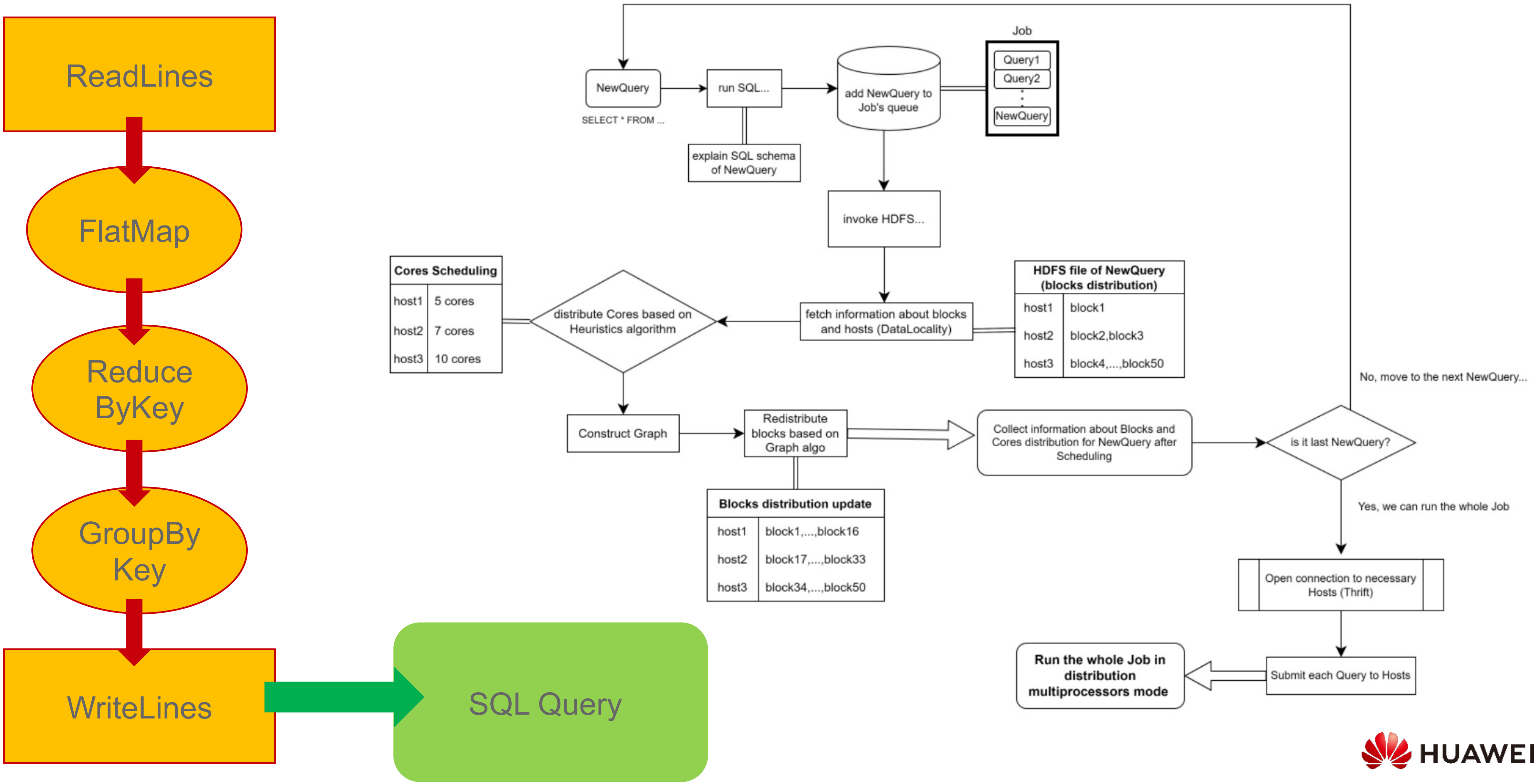








HDFS + MPI + Data Locality principle for scheduling



HUAWEI

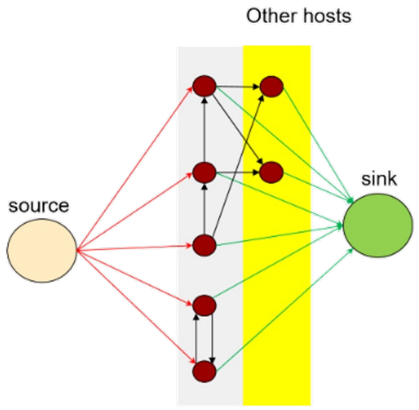
Graph approach: single-task scheduler

Assume that we have HDFS file with *B* blocks (with the same size) distributed between *H* hosts;

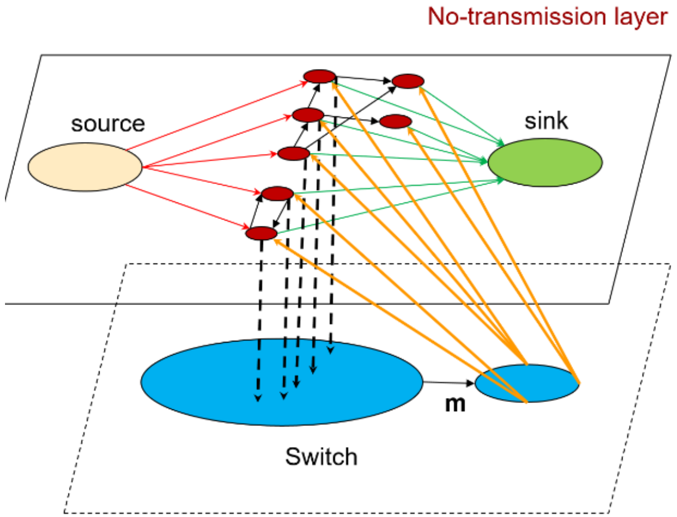
Each block is stored with replica factor *R* (same for all blocks);

Problem #1: All of blocks should be distributed most evenly: only *L* blocks per host (*L* is max-load value).

Problem #2: Consider switch engine (ability to transfer blocks between hosts dynamically) with capacity *m*;



Graph without switch
(Problem #1)



Graph with switch
(Problem #2)

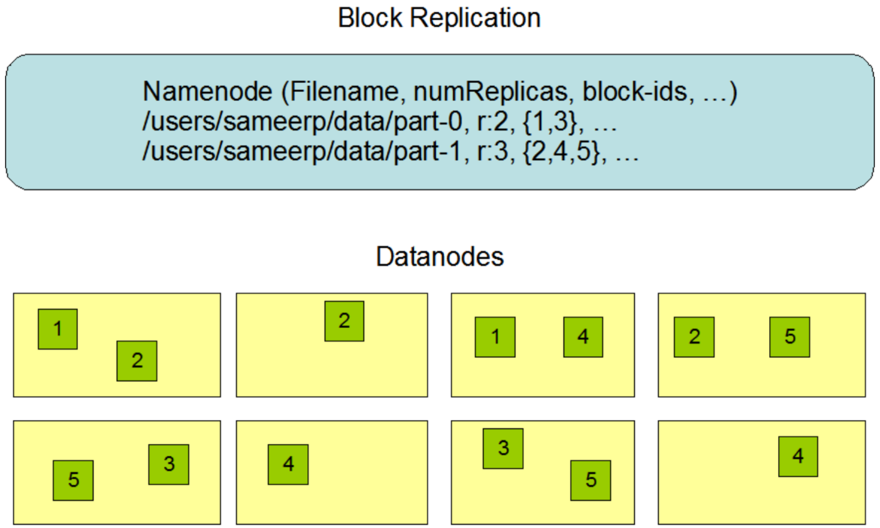


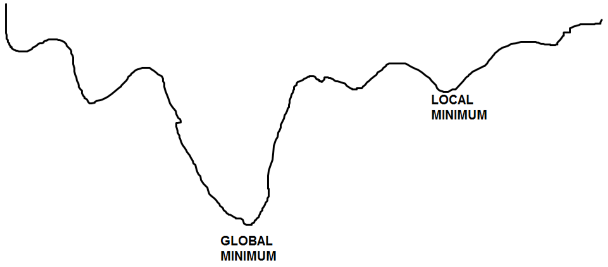
Illustration of data distribution in HDFS with B=5, H=8, R=2 for blocks 1,3 and R=3 for blocks 2,4,5)

HUAWEI

Optimization problem

- To optimize both L and m values let's describe two additional parameters: τ_c – cost of computation for one block and τ_t – cost for transferring of one block.
- We consider **cost function**: $C(L,m) = L\tau_c + m\tau_t$
- Instead of optimization only L parameter, we will optimize C(L,m) value with given cost metrics;
- To achieve the better results, we will use combination of binary search and Dinic's algorithm (for obtaining max-load value).
- We also should consider trade-off between number of transfers and minimal load of one host.

T_c	T_t	*ratio*	*L*	*m*
1	1	1	100	0
1	0.5	2	75	25
1	0.25	4	75	25
0.5	1	0.5	100	0



m	L	C(L,m)
15000	150	15300
936	150	1236
467	152	771
116	154	424
57	155	367
28	155	338
6	155	316
0	155	310

HUAWEI

Test cases result

- Several cases were tested with the same system and data configuration: 200 hosts, 30000 blocks, replica factor R = 3;

- This test describes a file with 3.5 TB of data;
- 10 tests has different topology and unevenly distributed data.

Case	Graph (no switch)		Equilibrium		Baseline	
	L	Time (secs)	L	Time (secs)	L	Time (secs)
0	155	0.154	156	164.06	170	0.136
1	155	0.149	155	169.36	165	0.132
2	156	0.156	156	172.76	169	0.135
3	154	0.153	154	170.12	166	0.139
4	153	0.154	153	165.00	163	0.131
5	153	0.140	153	165.97	168	0.133
6	155	0.159	155	172.54	164	0.131
7	154	0.155	154	170.87	165	0.136
8	154	0.140	154	170.17	165	0.133
9	155	0.147	155	172.43	168	0.132

Equilibrium algorithm is another concept based on data shuffling and resorting blocks;

Baseline is the basic HDFS scheduler tool;

Optimal area search is algorithm based on binary search via optimal (L,m) values;

Forward search use huge steps for defining of local (L,m) optimum.

Case	Optimal area search				Forward search			
	L	m	$C(L,m)$	Time (secs)	L	m	$C(L,m)$	Time (secs)
0	150	34	150034	1.0365	150	34	150034	0.5301
1	150	50	150050	1.0536	150	50	150050	0.6891
2	150	57	150057	1.3422	151	28	151028	0.5992
3	150	46	150046	1.0632	150	46	150046	0.7385
4	150	45	150045	1.0314	150	45	150045	0.7191
5	150	60	150060	1.0072	151	12	151012	0.6977
6	150	50	150050	1.4023	150	50	150050	0.8845
7	150	45	150045	1.2511	150	45	150045	0.7673
8	150	64	150053	1.3352	151	23	151023	0.8266
9	150	62	150062	1.4827	151	31	151031	0.7256

Data-driven approach: multi-task scheduler

- This strategy describes the method to allocate resources based on task sizes, number of blocks and scoring metric;
- It should schedule cores (workers) for each task and order of tasks execution in batch before graph scheduler;
- The batch contains of *Y* tasks in batch and *h* hosts, *C* described a maximum number of cores per host;
- Each task in batch mapped with label of size from empty (0) to large (4).

Algorithm 2 Simplified data-driven description

Assume there are N tasks in batch Δ and array W of weights for task sizes' labels from 0 to 4 (according to aliases "empty", "small",...,"large") for each task in Δ, $|W| = N$; Set the batch cardinality parameter λ and the σ-density parameter as well.
Reorder tasks based on block sizes (descending order);
Create new batch: choose 1 task from "head" of the queue and x tasks from "tail", where $\sum W(x) \approx \sigma$ and repeat it until the end of batch;
Calculate mean number of blocks M for all tasks;
for each task q in Job **do**
    $Cores = \frac{NumBlocks(q)}{M} \times \frac{C}{H} + \tau_c(q)\frac{W(q)}{\tau_t(q)}$;
    if $Cores > \frac{C}{\sigma}$ **then**
        $Cores = \frac{C}{\sigma}$;
    end if
    Assign $Cores$ for task q and distribute them uniformly between all h nodes;
    Do a single task scheduling for q (Graph strategy);
end for

σ	τ	time	Pure CPU usage	Tail CPU usage
1	5	592	50.93	76.58
1	10	578	62.21	77.80
1	15	570	65.14	68.85
2	5	522	65.42	51.33
2	10	517	68.92	80.21
2	15	500	71.93	70.28
3	5	477	65.66	50.28
3	10	456	76.39	80.66
3	15	428	77.52	71.04
5	5	515	53.45	67.43
5	10	504	59.68	69.66
5	15	498	62.77	65.02
10	5	588	77.24	50.25
10	10	560	70.51	48.54
10	15	531	66.01	41.00

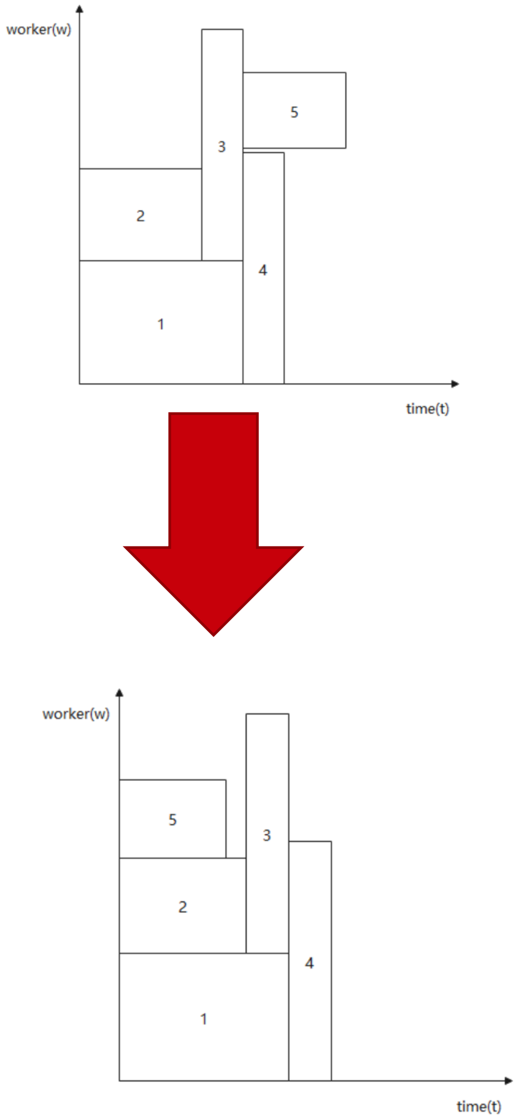
Other multi-task schedulers

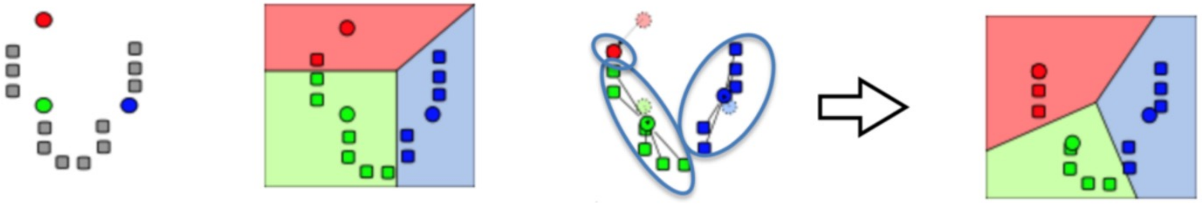
- **Annealing**: the purpose of scheduling is how to make the ”box” fit the tightest, that is, the shortest horizontal coordinate via avoiding the local minimums;

- **Dynamical annealing:** runtime recalculation of the optimal permutation based on its historical execution time;

- **Multiple batch:** optimal stacking mechanism for several batches execution based on K-Means and clusterization. No historical information.

- **Greedy division**: sorts all tasks by file size, the maximum number of cores occupies by 50% of the maximum task and for other tasks the number of allocated cores based on the ratio of the file size to the first task.



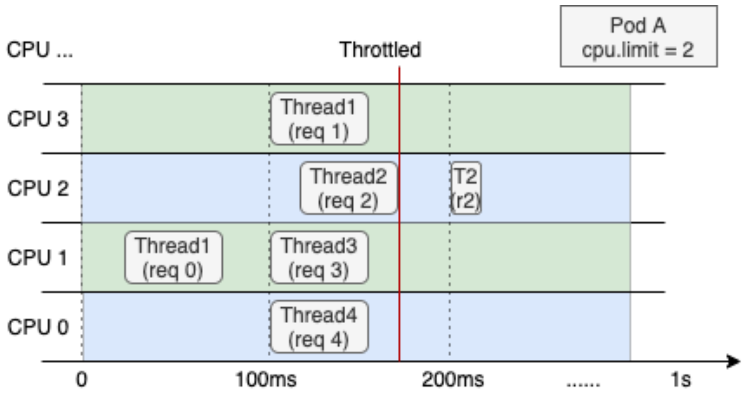


HUAWEI

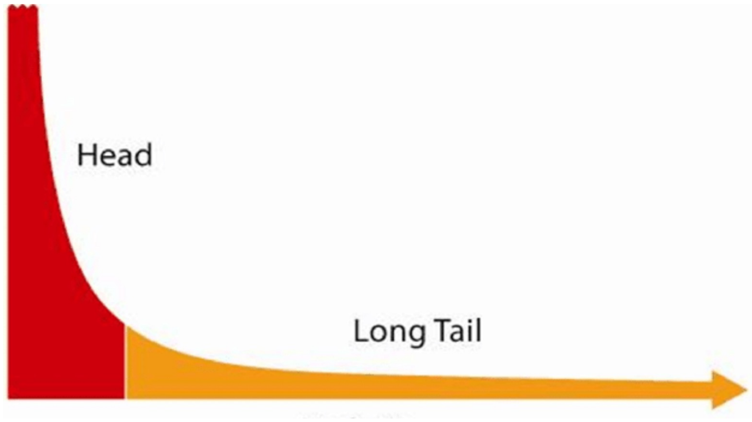
Scheduling efficiency: how to measure?

Problem: *How can it indicate the batch execution process is effective? How can it measure the idle part and non-idle part? What is the target resource for the cluster?*

The whole batch can be divided into two parts: the main part and the tail part. Each of them represents a useful and idle proportion of execution.



- **Execution time**: the whole time of scheduler work (basic metric);
- **Main part time**: execution time without tail part (from 0 seconds to $\hat{t}$);
- **Tail time**: execution time of tail part (from $\hat{t}$ to the end);
- **Pure CPU usage**: square of the diagram in the main part;
- **Tail CPU usage**: square of the diagram only on the tail (starts from $\hat{t}$);
- **Tail overhead**: the ratio between tail time and median time value among all of the tasks;
- **Efficiency capacity**: summation of the execution time of each task multiplied by the total used cores for current tasks.



Long tail problem: huge idle part

HUAWEI

Summary of the results: performance

Strategy	Time (secs)	Efficiency capacity
MultiBatch	996.851594	33792.451026
Greedy	520.334221	30531.777603
Data-driven	482.04614	25617.325310
Dyn.annealing	442.392825	24944.507181
Annealing	445.535692	25340.807954
Simple d.-driven	428.740834	22410.507381

Performance of different schedulers

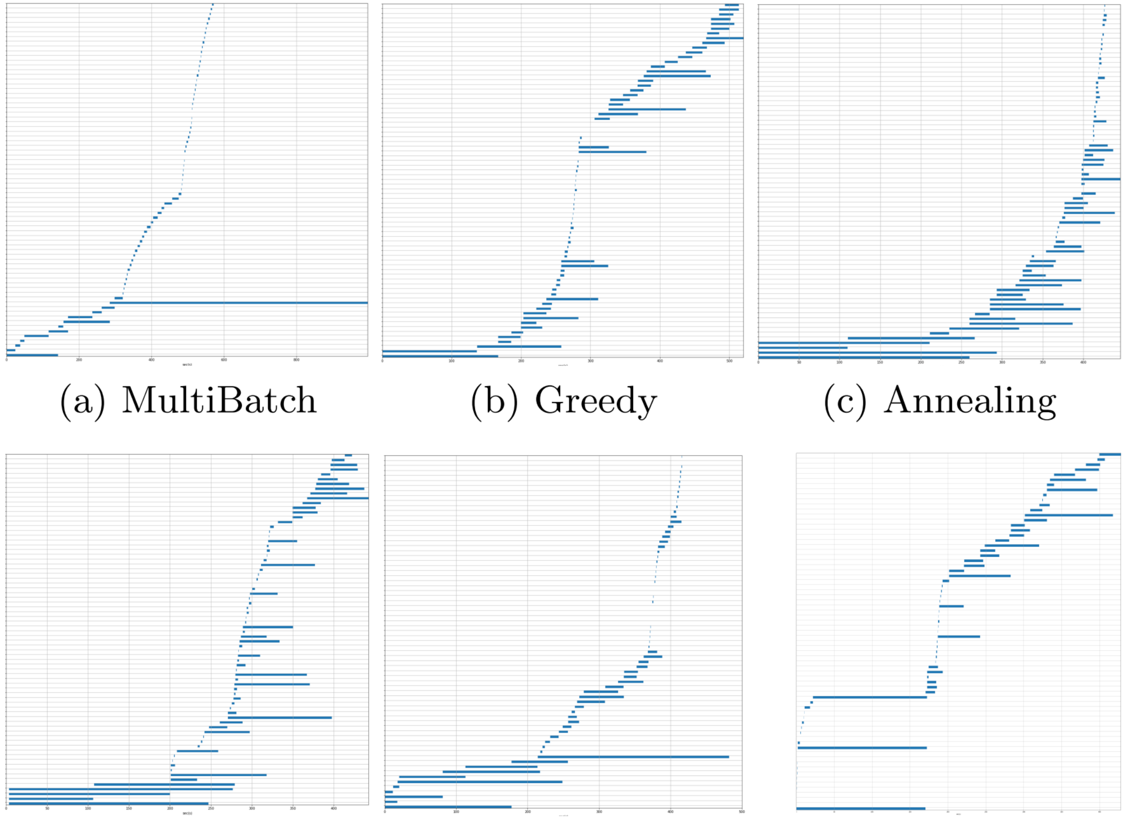
Method	Main part time	Tail time	Pure CPU usage (%)	Tail overhead	Tail CPU usage (%)
Annealing	417.083	28.452	79.54	2.82	36.70
Dyn. Annealing	321.111	121.281	76.17	6.81	68.67
MultiBatch	567.060	429.790	96.07	129.69	2.64
Greedy	485.182	35.151	82.96	1.87	64.92
Data-driven	413.250	68.795	75.62	10.38	27.55
Simple d.-driven	377.707	51.033	84.01	20.52	78.77

Tail metrics

Benchmark: SEQ – Service Experience Quality Analyst
Dynamic scheduling of processes, tasks and resources is required
71 tasks in batch, the total size of the batch is 584 GBs and it contains ≈ 3.6 billions of data rows;
Hardware configuration: 6 nodes at the cluster, CPU with Intel Xeon Gold 6230N (2.30GHz/20cores), RAM: 8*32G DDR4 ECC

After SQL operations (filter, select, group by...) it was obtained 38 GBs of output data with ≈ 0.3 billions of rows.

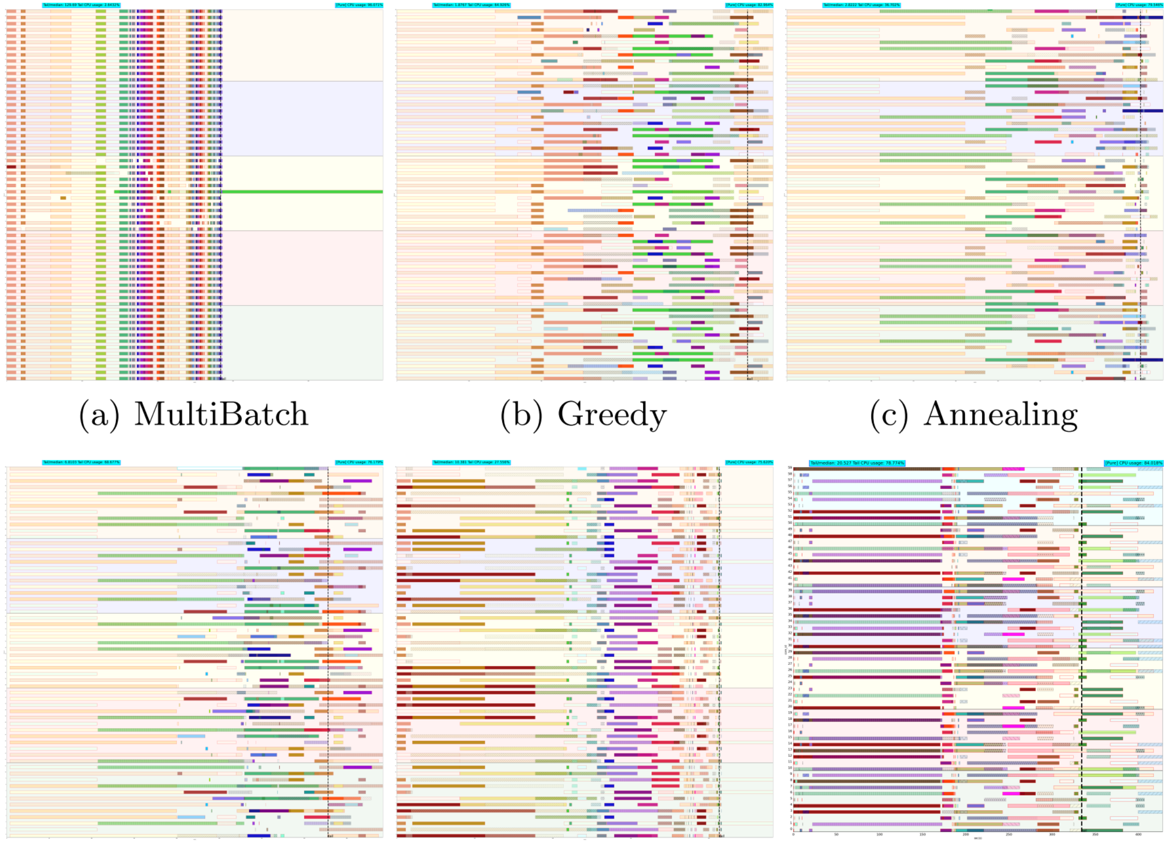
Summary of the results: visualization



(a) MultiBatch (b) Greedy (c) Annealing

(d) Dyn.Annealing (e) Data-driven (f) Simple d.-driven

Gantt chart



(a) MultiBatch (b) Greedy (c) Annealing

(d) Dyn.Annealing (e) Data-driven (f) Simple d.-driven

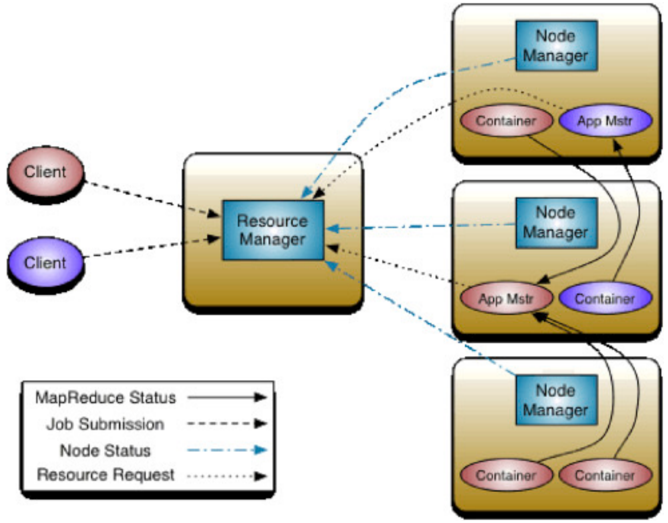
CPU utility diagrams

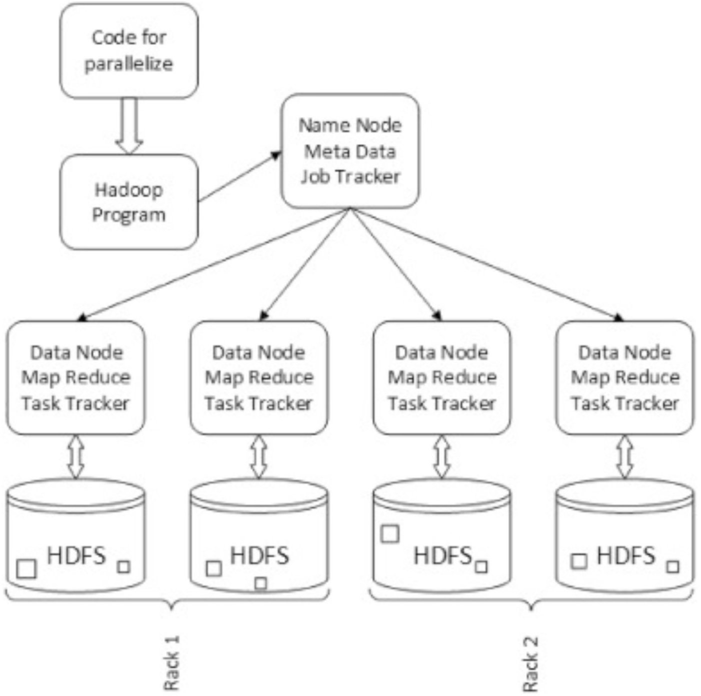
HUAWEI

Conclusion

- An approach based on data locality and HDFS-file representation feature was implemented and proved its high efficiency in comparison with other strategies to reduce batch total execution time;
- The optimal parameterization for tuning of the batch was discovered, and it was shown in experiments that the derived metrics perfectly demonstrated their applicability;
- Heuristics is the most applicable solution for MPI-based batch and experimental framework;

- Data-driven heuristics is **5-10%** faster than annealing algorithms;
- It's also provides the highest efficiency capacity (**20%** better than annealing) and the highest tail CPU usage (**10%** better than annealing);
- Multibatch and greedy strategies are inefficient by **40-70%** by time and CPU usage in comparison with heuristics approach.





HUAWEI

Thank you

Presenter: Ilya Timokhin
is.timokhin@hse.ru