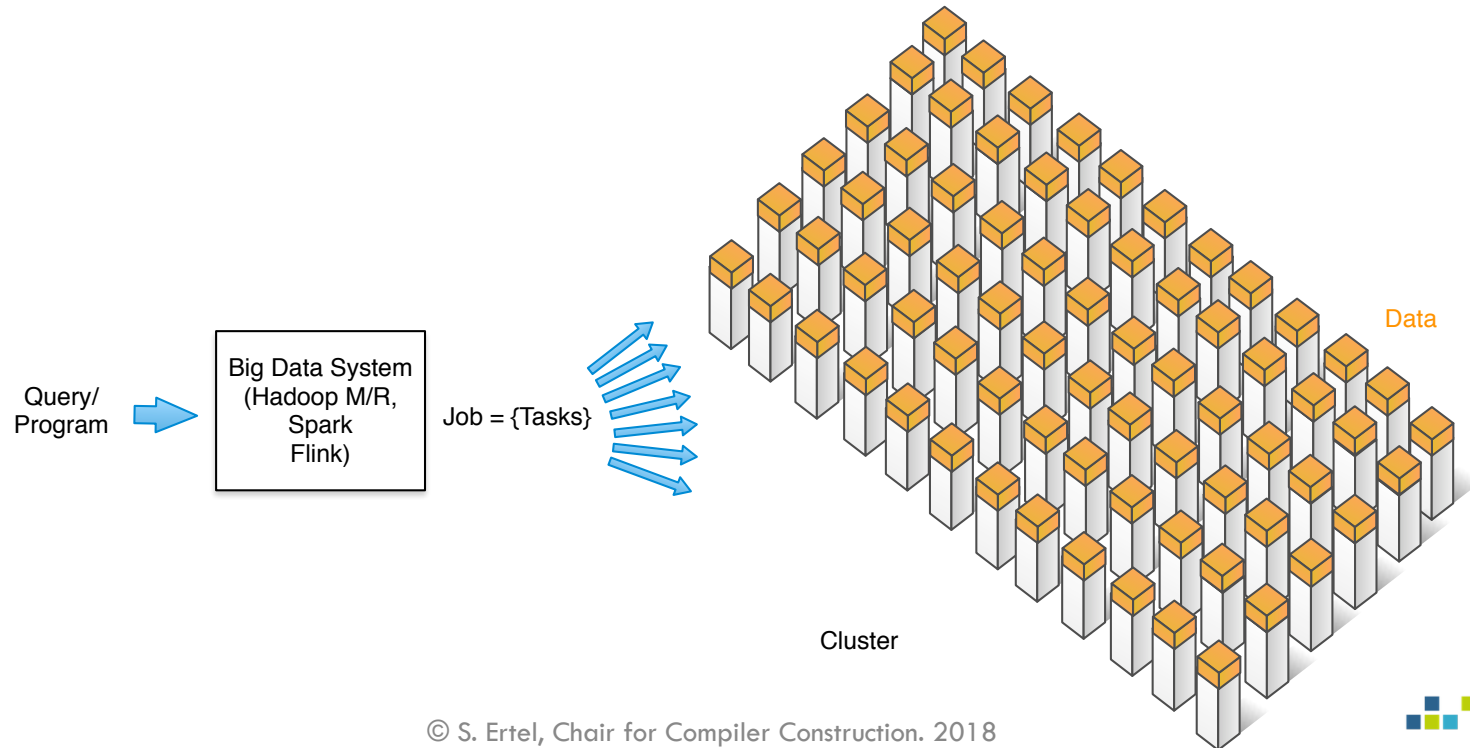


Supporting Fine-grained Dataflow Parallelism in Big Data Systems

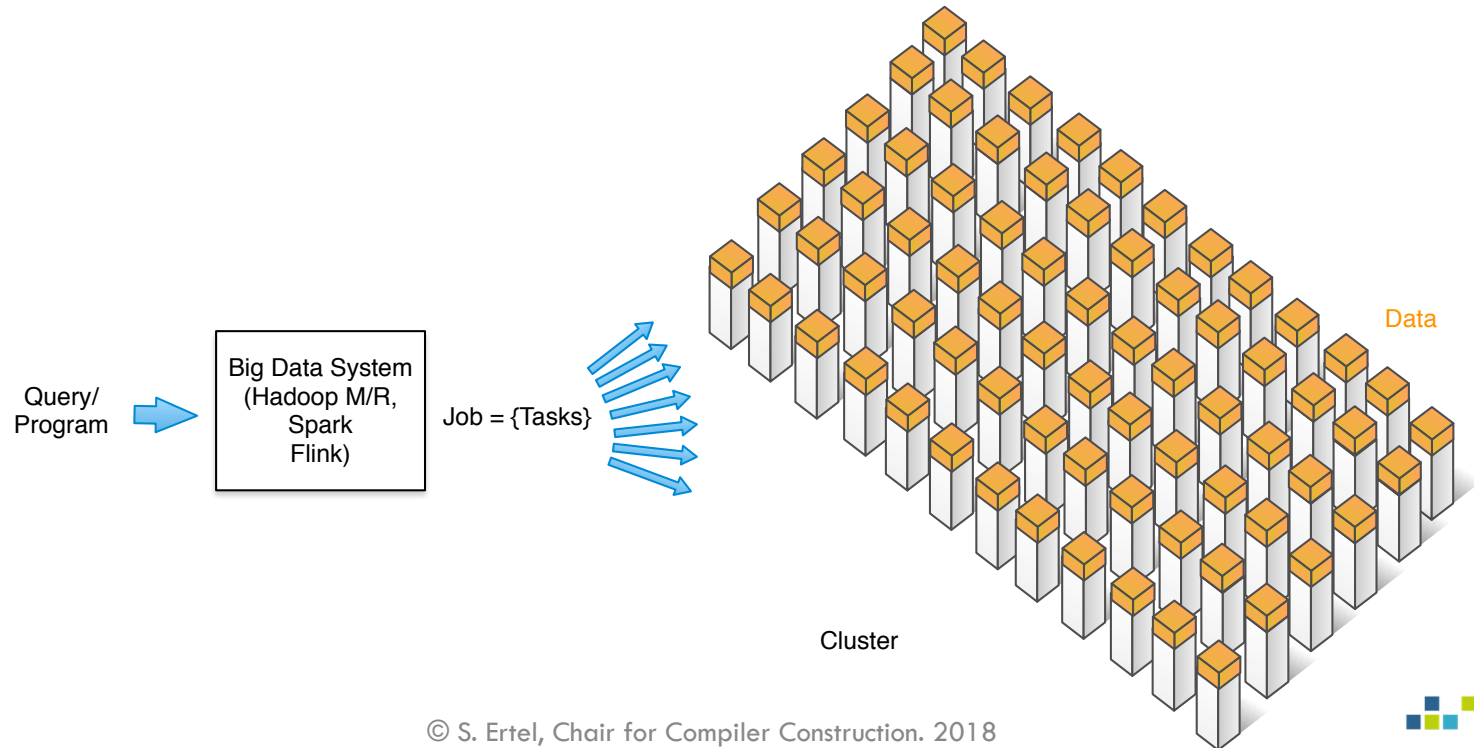
Sebastian Ertel, Justus Adam and Jeronimo Castrillon

Chair for Compiler Construction
TU Dresden

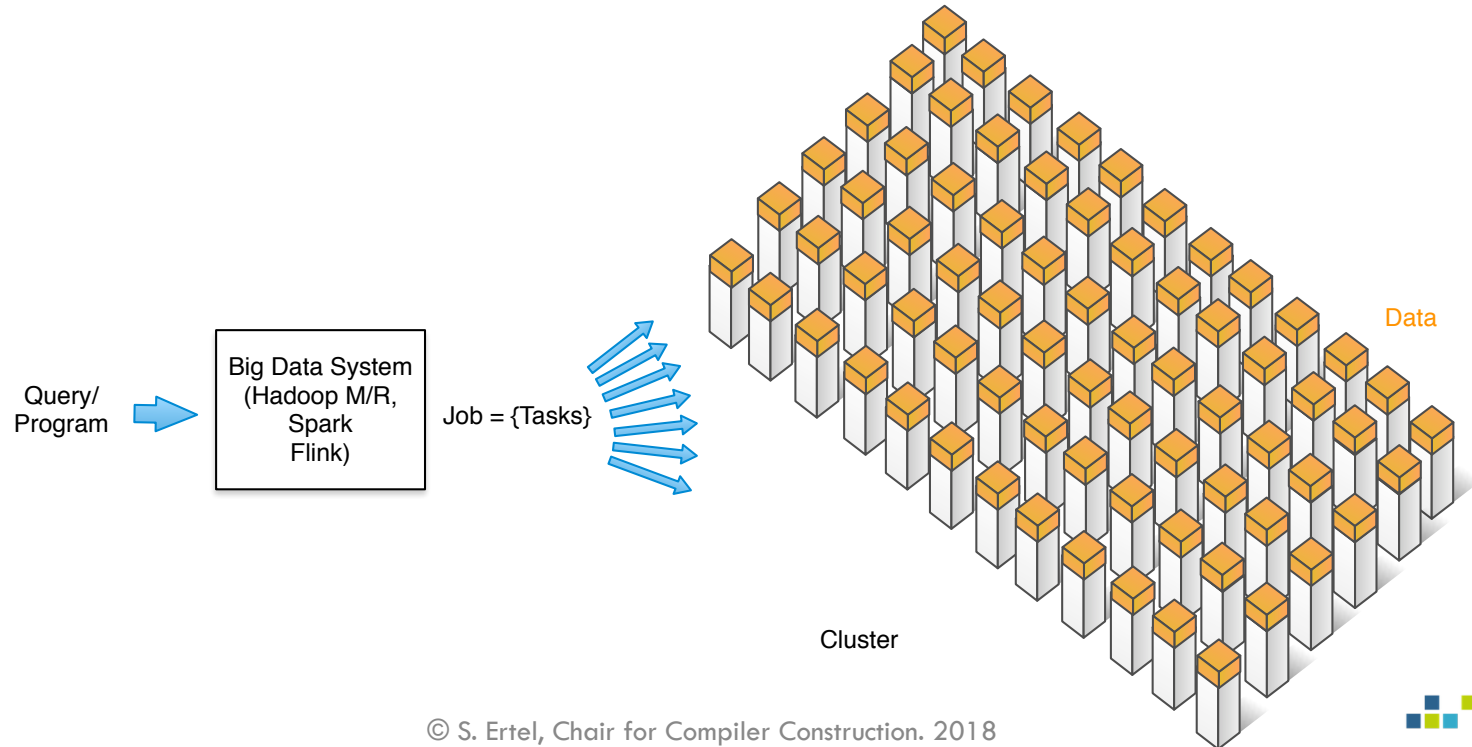
9th International Workshop on Programming Models and Applications for Multicores and Manycores
Vienna, 25.2.2018



1. Big Data Systems (BDSs) scale with the number of cores in the cluster
2. The main bottleneck is I/O (disk and network)



1. Big Data Systems (BDSs) scale with the number of cores in the cluster for independent tasks.
2. The main bottleneck is I/O (disk and network) for simple data.



1. Big Data Systems (BDSs) scale with the number of cores in the cluster for independent tasks.
2. The main bottleneck is I/O (disk and network) for simple data.

Jobs are CPU-bound!

WordCount, Sort
Simple Data Formats
Uncompressed

vs.

Analytics Queries (in Hive)
Complex Data Formats (Parquet, Tables, JSON)
Compressed

Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making sense of performance in data analytics frameworks.(NSDI'15).

1. Big Data Systems (BDSs) scale with the number of cores in the cluster for independent tasks.
2. The main bottleneck is I/O (disk and network) for simple data.

Jobs are CPU-bound!

WordCount, Sort
Simple Data Formats
Uncompressed

vs.

Analytics Queries (in Hive)
Complex Data Formats (Parquet, Tables, JSON)
Compressed

Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making sense of performance in data analytics frameworks. (NSDI'15).

BDS cores/data processing pipelines do not scale on multicores!

- Local optimizations do not solve this problem.
- BDSs do not benefit from new network HW.

➔ Rewrite data processing cores!

Animesh Trivedi, Patrick Stuedi, Jonas Pfefferle, Radu Stoica, Bernard Metzler, Ioannis Koltsidas, and Nikolas Ioannou. 2016. On the [ir]relevance of network performance for data processing. (HotCloud'16).

BDS core study

(Trivedi et.al., 2016) results mostly experimental.

↪ What is the inherent pattern that all these BDS cores suffer from?

Code study: Hadoop M/R (HMR), Spark, Flink

BDS core study → Implicit Parallel Programming

(Trivedi et.al., 2016) results mostly experimental.

↪ What is the inherent pattern that all these BDS cores suffer from?

Code study: Hadoop M/R (HMR), Spark, Flink

BDS cores are already complex:

Framework	Java	Scala	Python
HMR 2.7.3	1402899	0	406
Spark 2.11	99118	344613	28079
Flink 2.10	491339	81549	2814

↪ Threads and Co. make them even more complex and harder to maintain!

Challenge Accepted!

BDS core study → Implicit Parallel Programming → BDS core rewrite

(Trivedi et.al., 2016) results mostly experimental.

↪ What is the inherent pattern that all these BDS cores suffer from?

Code study: Hadoop M/R (HMR), Spark, Flink

BDS cores are already complex:

Framework	Java	Scala	Python
HMR 2.7.3	1402899	0	406
Spark 2.11	99118	344613	28079
Flink 2.10	491339	81549	2814

↪ Threads and Co. make them even more complex and harder to maintain!

If all BDS cores are similar ↪ Representative rewrite(s) for the HMR (map task) core.

BDS core study → Implicit Parallel Programming → BDS core rewrite

```
1 public class Mapper<KEYIN, VALUEIN,  
2     KEYOUT, VALUEOUT> {  
3     /* The default implementation  
4     is the identity function. */  
5     protected  
6     void map(KEYIN key, VALUEIN value,  
7         Context ctxt) {  
8         ctxt.write((KEYOUT) key,  
9             (VALUEOUT) value);  
10    }  
11  
12    public  
13    void run(Context ctxt) {  
14        while (ctxt.nextKeyValue())  
15            map(ctxt.getCurrentKey(),  
16                ctxt.getCurrentValue(),  
17                ctxt);  
18    }}
```

(a) Hadoop

```
1 private[spark] class  
2 ShuffleMapTask(partitionId: Int,  
3     partition: Partition)  
4     extends Task[MapStatus] {  
5  
6     override  
7     def runTask(ctxt: TaskContext)  
8         :MapStatus = {  
9         /* Deserialization and init of  
10        variables omitted for brevity. */  
11        var writer: ShuffleWriter[Any, Any] =  
12            manager.getWriter[Any, Any](  
13                dep.shuffleHandle,  
14                partitionId, ctxt)  
15        writer.write(  
16            rdd.iterator(partition, ctxt)  
17                .asInstanceOf[  
18                Iterator[_<:Product2[Any, Any]]])  
19        writer.stop(success = true).get}}
```

(b) Spark

```
1 public class DataSourceTask<OT>  
2     extends AbstractInvokable {  
3     private  
4         InputFormat<OT, InputSplit> format;  
5     private Collector<OT> output;  
6  
7     @Override public  
8     void invoke() throws Exception {  
9         OT reuse =  
10             serializer.createInstance();  
11         while (!this.taskCanceled &&  
12             !format.reachedEnd()) {  
13             OT returned;  
14             if((returned =  
15                 format.nextRecord(reuse))  
16                 != null)  
17                 output.collect(returned);  
18         }}
```

(c) Flink

Inherent Parallelism in BDS cores

BDS core study → Implicit Parallel Programming → BDS core rewrite

```
1 public class Mapper<KEYIN, VALUEIN,  
2     KEYOUT, VALUEOUT> {  
3     /* The default implementation  
4     is the identity function. */  
5     protected  
6     void map(KEYIN key, VALUEIN value,  
7         Context ctxt) {  
8         ctxt.write((KEYOUT) key,  
9             (VALUEOUT) value);  
10    }  
11  
12    public  
13    void run(Context ctxt) {  
14        while (ctxt.nextKeyValue())  
15            map(ctxt.getCurrentKey(),  
16                ctxt.getCurrentValue(),  
17                ctxt);  
18    }}
```

(a) Hadoop

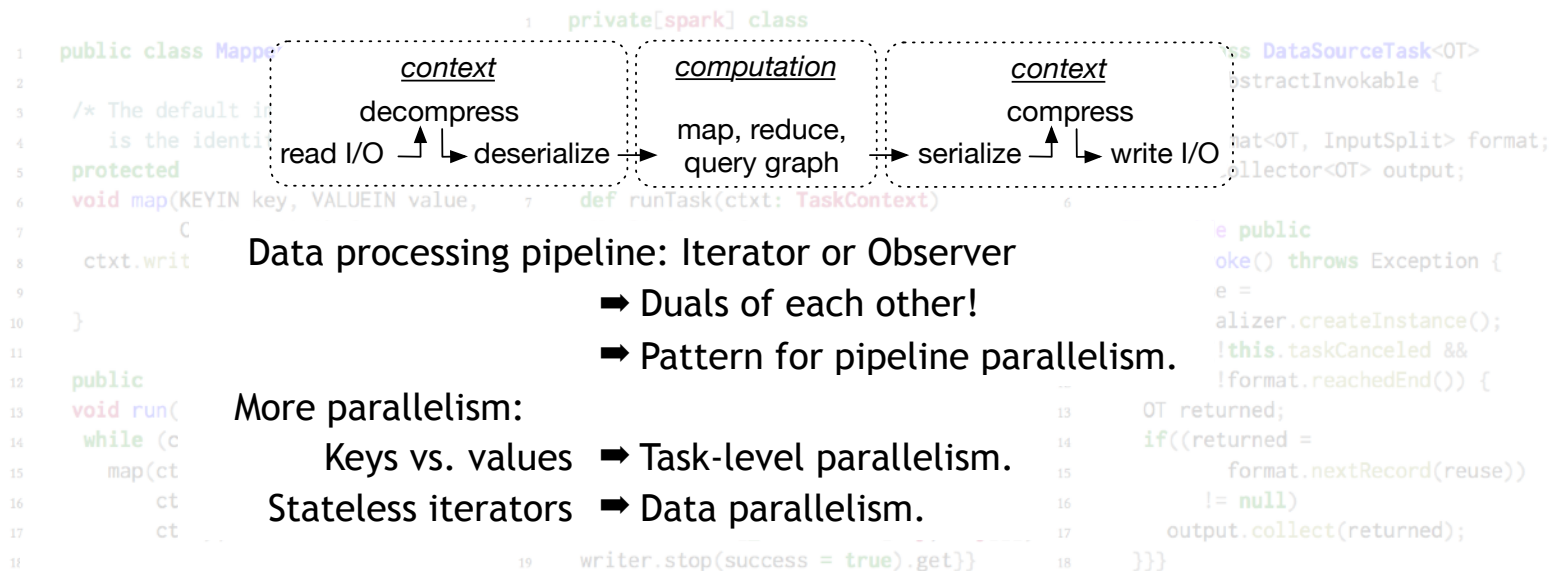
```
1 private[spark] class  
2 ShuffleMapTask(partitionId: Int,  
3     partition: Partition)  
4     extends Task[MapStatus] {  
5  
6     override  
7     def runTask(ctxt: TaskContext)  
8     : MapStatus = {  
9         /* Deserialization and init of  
10        variables omitted for brevity. */  
11        var writer: ShuffleWriter[Any, Any] =  
12            manager.getWriter[Any, Any](  
13                dep.shuffleHandle,  
14                partitionId, ctxt)  
15        writer.write(  
16            rdd.iterator(partition, ctxt)  
17                .asInstanceOf[  
18                Iterator[_<: Product2[Any, Any]]])  
19        writer.stop(success = true).get}}
```

(b) Spark

```
1 public class DataSourceTask<OT>  
2     extends AbstractInvokable {  
3     private  
4         InputFormat<OT, InputSplit> format;  
5         private Collector<OT> output;  
6  
7         @Override public  
8         void invoke() throws Exception {  
9             OT reuse =  
10                 serializer.createInstance();  
11             while (!this.taskCanceled &&  
12                 !format.reachedEnd()) {  
13                 OT returned;  
14                 if ((returned =  
15                     format.nextRecord(reuse))  
16                     != null)  
17                     output.collect(returned);  
18             }}
```

(c) Flink

BDS core study → Implicit Parallel Programming → BDS core rewrite



Erik Meijer. Subject/Observer is Dual to Iterator. 2010. PLDI, Fun Ideas and Thoughts Session.

(c) Flink

BDS core study → Implicit Parallel Programming → BDS core rewrite

Implicit = No concurrency/parallelism abstractions, only functions/algorithms and variables.

DSL

Dataflow Runtime

BDS core study → Implicit Parallel Programming → BDS core rewrite

Implicit = No concurrency/parallelism abstractions, only functions/algorithms and variables.

DSL

$t ::= v$
| (algo [v] t)
| (t t)
| (let [v t] t)
| (if t t t)
| (sf_{JVM} v₁ ... v_n)
| (seq t t)

(smap (algo [v] t) [v₁ ... v_n])

Dataflow Runtime

$v ::= v \in V_{JVM}$
| [v₁ ... v_n]

BDS core study → Implicit Parallel Programming → BDS core rewrite

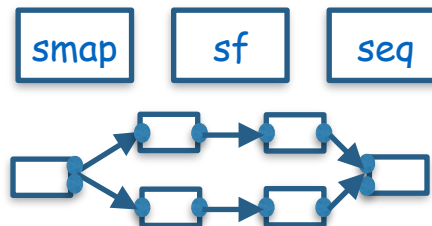
Implicit = No concurrency/parallelism abstractions, only functions/algorithms and variables.

DSL

$\dagger ::= v$ $v ::= v \in V_{JVM}$
 | (algo [v] \dagger) | [$v_1 \dots v_n$]
 | ($\dagger \dagger$)
 | (let [v \dagger] \dagger)
 | (if $\dagger \dagger \dagger$)
 | ($\text{sf}_{JVM} v_1 \dots v_n$)
 | (seq $\dagger \dagger$)
 (smap (algo [v] \dagger) [$v_1 \dots v_n$])

Dataflow Runtime

$\dagger ::=$ \rightarrow (arc)
 | \bullet (port)
 | \square (node)



- Threads/actors/...
- Queues/channels/...
- Scheduler

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

```
1  public class Mapper<KEYIN, VALUEIN,  
2      KEYOUT, VALUEOUT> {  
3      /* The default implementation  
4       is the identity function. */  
5      protected  
6      void map(KEYIN key, VALUEIN value,  
7          Context ctxt) {  
8          ctxt.write((KEYOUT) key,  
9              (VALUEOUT) value);  
10     }  
11  
12     public  
13     void run(Context ctxt) {  
14         while (ctxt.nextKeyValue())  
15             map(ctxt.getCurrentKey(),  
16                 ctxt.getCurrentValue(),  
17                 ctxt);  
18     }}
```

(a) Hadoop

Rewriting the HMR Map Task

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

```
1 public class Mapper<KEYIN, VALUEIN,  
2     KEYOUT, VALUEOUT> {  
3     /* The default implementation  
4     is the identity function. */  
5     protected  
6     void map(KEYIN key, VALUEIN value,  
7         Context ctxt) {  
8         ctxt.write((KEYOUT) key,  
9             (VALUEOUT) value);  
10    }  
11  
12    public  
13    void run((Context ctxt) {  
14        while (ctxt.nextKeyValue())  
15            map(ctxt.getCurrentKey(),  
16                ctxt.getCurrentValue(),  
17                ctxt);  
18    }}
```

(a) Hadoop

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

```
1 public class Mapper<KEYIN, VALUEIN,  
2     KEYOUT, VALUEOUT> {  
3  
4     1 (defn coarse  
5         2 [^org.apache.hadoop.mapreduce.Mapper$Context reader  
6         3 ^org.apache.hadoop.mapreduce.Mapper mapper  
7         4 ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8         5 (let [records-on-disk (new InputIterator reader)]  
9             6 (ohua  
10                7 (smap  
11                    8 (algo compute-and-output [ [line content] ]  
12                        9 (let [kv-pairs (hmr-map line content mapper)]  
13                            10 (smap  
14                                11 (algo output-side [ [k v] ] (output k v writer))  
15                                    12 kv-pairs)))  
16                            13 records-on-disk))))  
17     })
```

(a) Hadoop

Rewriting the HMR Map Task

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

Closure function

```
1 public class Mapper<KEYIN, VAL, KEYOUT> {  
2     KEYOUT  
3  
4     1 (defn coarse  
5         [^org.apache.hadoop.mapreduce.Mapper$Context reader  
6         ^org.apache.hadoop.mapreduce.Mapper mapper  
7         ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8         5 (let [records-on-disk (new InputIterator reader)]  
9             6 (ohua  
10                7 (smap  
11                   8 (algo compute-and-output [ [line content] ]  
12                      9 (let [kv-pairs (hmr-map line content mapper)]  
13                         10 (smap  
14                            11 (algo output-side [ [k v] ] (output k v writer))  
15                             12 kv-pairs)))  
16                             13 records-on-disk))))  
17     }  
18 }
```

(a) Hadoop

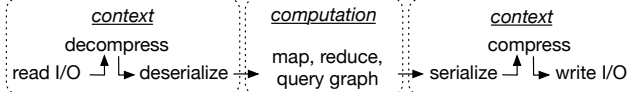
Rewriting the HMR Map Task

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

```
1 public class Mapper<KEYIN, VALUEIN,  
2     KEYOUT, VALUEOUT> {  
3  
4     1 (defn coarse  
5         [^org.apache.hadoop.mapreduce.Mapper$Context reader  
6         ^org.apache.hadoop.mapreduce.Mapper mapper  
7         ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8         5 (let [records-on-disk (new InputIterator reader)]  
9             6 (ohua  
10                7 (smap  
11                    8 (algo compute-and-output [ [line content] ]  
12                        9 (let [kv-pairs (hmr-map line content mapper)]  
13                            10 (smap  
14                                11 (algo output-side [ [k v] ] (output k v writer))  
15                                    12 kv-pairs)))  
16                            13 records-on-disk))))  
17     })  
18 }
```

(a) Hadoop



Rewriting the HMR Map Task

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

```
1 public class Mapper<KEYIN, VALUEIN,  
2     KEYOUT, VALUEOUT> {  
3  
4 1 (defn coarse  
5 2   [^org.apache.hadoop.mapreduce.Mapper$Context  
6 3   ^org.apache.hadoop.mapreduce.Mapper mapper  
7 4   ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8 5   (let [records-on-disk (new InputIterator reader)]  
9 6     (ohua  
10 7       (smap  
11 8         (algo compute-and-output [ [line content] ]  
12 9           (let [kv-pairs (hmr-map line content mapper)]  
13 10             (smap  
14 11               (algo output-side [ [k v] ] (output k v writer))  
15 12                 kv-pairs)))  
16 13             records-on-disk))))  
17  
18 }}
```

BDS core
written in Ohua

(a) Hadoop

BDS core study ➔ Implicit Parallel Programming ➔ BDS core rewrite

Ready! Set! Go!

```
1  public class Mapper<KEYIN, VALUEIN,  
2      KEYOUT, VALUEOUT> {  
3  
4      1  (defn coarse  
5          2  [^org.apache.hadoop.mapreduce.Mapper$Context reader  
6              3  ^org.apache.hadoop.mapreduce.Mapper mapper  
7                  4  ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8          5  (let [records-on-disk (new InputIterator reader)]  
9              6  (ohua  
10                 7  (smap  
11                     8  (algo compute-and-output [ [line content] ]  
12                         9  (let [kv-pairs (hmr-map line content mapper)]  
13                             10  (smap  
14                                 11  (algo output-side [ [k v] ] (output k v writer))  
15                                     12  kv-pairs)))  
16                                     13  records-on-disk))))  
17      14  })  
18  }
```

(a) Hadoop

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

```
1  public class Mapper<KEYIN, VALUEIN,  
2      KEYOUT, VALUEOUT> {  
3  
4  1  (defn coarse  
5  2      [^org.apache.hadoop.mapreduce.Mapper$Context reader  
6  3      ^org.apache.hadoop.mapreduce.Mapper mapper  
7  4      ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8  5      (let [records-on-disk (new InputIterator reader)]  
9  6          (ohua  
10 7              (smap  
11 8                  (algo compute-and-output [ [line content] ]  
12 9                      (let [kv-pairs (hmr-map line content mapper)]  
1310                          (smap  
1411                              (algo output-side [ [k v] ] (output k v writer))  
1512                                  kv-pairs)))  
1613                          records-on-disk))))  
1714      ))  
18  }}
```

(a) Hadoop

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

```
1 public class Mapper<KEYIN, VALUEIN,  
2     KEYOUT, VALUEOUT> {  
3  
4     1 (defn coarse  
5         2 [^org.apache.hadoop.mapreduce.Mapper$Context reader  
6         3 ^org.apache.hadoop.mapreduce.Mapper mapper  
7         4 ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8         5 (let [records-on-disk (new InputIterator reader)]  
9             6 (ohua  
10                7 (smap  
11                    8 (algo compute-and-output [ [line content] ]  
12                        9 (let [kv-pairs (hmr-map line content mapper)]  
13                            10 (smap  
14                                11 (algo output-side [ [k v] ] (output k v writer))  
15                                    12 kv-pairs)))  
16                            13 records-on-disk))))  
17     14  
18     15  
19     16  
20     17  
21     18  
22     19  
23     20  
24     21  
25     22  
26     23  
27     24  
28     25  
29     26  
30     27  
31     28  
32     29  
33     30  
34     31  
35     32  
36     33  
37     34  
38     35  
39     36  
40     37  
41     38  
42     39  
43     40  
44     41  
45     42  
46     43  
47     44  
48     45  
49     46  
50     47  
51     48  
52     49  
53     50  
54     51  
55     52  
56     53  
57     54  
58     55  
59     56  
60     57  
61     58  
62     59  
63     60  
64     61  
65     62  
66     63  
67     64  
68     65  
69     66  
70     67  
71     68  
72     69  
73     70  
74     71  
75     72  
76     73  
77     74  
78     75  
79     76  
80     77  
81     78  
82     79  
83     80  
84     81  
85     82  
86     83  
87     84  
88     85  
89     86  
90     87  
91     88  
92     89  
93     90  
94     91  
95     92  
96     93  
97     94  
98     95  
99     96  
100    97  
101    98  
102    99  
103    100  
104    101  
105    102  
106    103  
107    104  
108    105  
109    106  
110    107  
111    108  
112    109  
113    110  
114    111  
115    112  
116    113  
117    114  
118    115  
119    116  
120    117  
121    118  
122    119  
123    120  
124    121  
125    122  
126    123  
127    124  
128    125  
129    126  
130    127  
131    128  
132    129  
133    130  
134    131  
135    132  
136    133  
137    134  
138    135  
139    136  
140    137  
141    138  
142    139  
143    140  
144    141  
145    142  
146    143  
147    144  
148    145  
149    146  
150    147  
151    148  
152    149  
153    150  
154    151  
155    152  
156    153  
157    154  
158    155  
159    156  
160    157  
161    158  
162    159  
163    160  
164    161  
165    162  
166    163  
167    164  
168    165  
169    166  
170    167  
171    168  
172    169  
173    170  
174    171  
175    172  
176    173  
177    174  
178    175  
179    176  
180    177  
181    178  
182    179  
183    180  
184    181  
185    182  
186    183  
187    184  
188    185  
189    186  
190    187  
191    188  
192    189  
193    190  
194    191  
195    192  
196    193  
197    194  
198    195  
199    196  
200    197  
201    198  
202    199  
203    200  
204    201  
205    202  
206    203  
207    204  
208    205  
209    206  
210    207  
211    208  
212    209  
213    210  
214    211  
215    212  
216    213  
217    214  
218    215  
219    216  
220    217  
221    218  
222    219  
223    220  
224    221  
225    222  
226    223  
227    224  
228    225  
229    226  
230    227  
231    228  
232    229  
233    230  
234    231  
235    232  
236    233  
237    234  
238    235  
239    236  
240    237  
241    238  
242    239  
243    240  
244    241  
245    242  
246    243  
247    244  
248    245  
249    246  
250    247  
251    248  
252    249  
253    250  
254    251  
255    252  
256    253  
257    254  
258    255  
259    256  
260    257  
261    258  
262    259  
263    260  
264    261  
265    262  
266    263  
267    264  
268    265  
269    266  
270    267  
271    268  
272    269  
273    270  
274    271  
275    272  
276    273  
277    274  
278    275  
279    276  
280    277  
281    278  
282    279  
283    280  
284    281  
285    282  
286    283  
287    284  
288    285  
289    286  
290    287  
291    288  
292    289  
293    290  
294    291  
295    292  
296    293  
297    294  
298    295  
299    296  
300    297  
301    298  
302    299  
303    300  
304    301  
305    302  
306    303  
307    304  
308    305  
309    306  
310    307  
311    308  
312    309  
313    310  
314    311  
315    312  
316    313  
317    314  
318    315  
319    316  
320    317  
321    318  
322    319  
323    320  
324    321  
325    322  
326    323  
327    324  
328    325  
329    326  
330    327  
331    328  
332    329  
333    330  
334    331  
335    332  
336    333  
337    334  
338    335  
339    336  
340    337  
341    338  
342    339  
343    340  
344    341  
345    342  
346    343  
347    344  
348    345  
349    346  
350    347  
351    348  
352    349  
353    350  
354    351  
355    352  
356    353  
357    354  
358    355  
359    356  
360    357  
361    358  
362    359  
363    360  
364    361  
365    362  
366    363  
367    364  
368    365  
369    366  
370    367  
371    368  
372    369  
373    370  
374    371  
375    372  
376    373  
377    374  
378    375  
379    376  
380    377  
381    378  
382    379  
383    380  
384    381  
385    382  
386    383  
387    384  
388    385  
389    386  
390    387  
391    388  
392    389  
393    390  
394    391  
395    392  
396    393  
397    394  
398    395  
399    396  
400    397  
401    398  
402    399  
403    400  
404    401  
405    402  
406    403  
407    404  
408    405  
409    406  
410    407  
411    408  
412    409  
413    410  
414    411  
415    412  
416    413  
417    414  
418    415  
419    416  
420    417  
421    418  
422    419  
423    420  
424    421  
425    422  
426    423  
427    424  
428    425  
429    426  
430    427  
431    428  
432    429  
433    430  
434    431  
435    432  
436    433  
437    434  
438    435  
439    436  
440    437  
441    438  
442    439  
443    440  
444    441  
445    442  
446    443  
447    444  
448    445  
449    446  
450    447  
451    448  
452    449  
453    450  
454    451  
455    452  
456    453  
457    454  
458    455  
459    456  
460    457  
461    458  
462    459  
463    460  
464    461  
465    462  
466    463  
467    464  
468    465  
469    466  
470    467  
471    468  
472    469  
473    470  
474    471  
475    472  
476    473  
477    474  
478    475  
479    476  
480    477  
481    478  
482    479  
483    480  
484    481  
485    482  
486    483  
487    484  
488    485  
489    486  
490    487  
491    488  
492    489  
493    490  
494    491  
495    492  
496    493  
497    494  
498    495  
499    496  
500    497  
501    498  
502    499  
503    500  
504    501  
505    502  
506    503  
507    504  
508    505  
509    506  
510    507  
511    508  
512    509  
513    510  
514    511  
515    512  
516    513  
517    514  
518    515  
519    516  
520    517  
521    518  
522    519  
523    520  
524    521  
525    522  
526    523  
527    524  
528    525  
529    526  
530    527  
531    528  
532    529  
533    530  
534    531  
535    532  
536    533  
537    534  
538    535  
539    536  
540    537  
541    538  
542    539  
543    540  
544    541  
545    542  
546    543  
547    544  
548    545  
549    546  
550    547  
551    548  
552    549  
553    550  
554    551  
555    552  
556    553  
557    554  
558    555  
559    556  
560    557  
561    558  
562    559  
563    560  
564    561  
565    562  
566    563  
567    564  
568    565  
569    566  
570    567  
571    568  
572    569  
573    570  
574    571  
575    572  
576    573  
577    574  
578    575  
579    576  
580    577  
581    578  
582    579  
583    580  
584    581  
585    582  
586    583  
587    584  
588    585  
589    586  
590    587  
591    588  
592    589  
593    590  
594    591  
595    592  
596    593  
597    594  
598    595  
599    596  
600    597  
601    598  
602    599  
603    600  
604    601  
605    602  
606    603  
607    604  
608    605  
609    606  
610    607  
611    608  
612    609  
613    610  
614    611  
615    612  
616    613  
617    614  
618    615  
619    616  
620    617  
621    618  
622    619  
623    620  
624    621  
625    622  
626    623  
627    624  
628    625  
629    626  
630    627  
631    628  
632    629  
633    630  
634    631  
635    632  
636    633  
637    634  
638    635  
639    636  
640    637  
641    638  
642    639  
643    640  
644    641  
645    642  
646    643  
647    644  
648    645  
649    646  
650    647  
651    648  
652    649  
653    650  
654    651  
655    652  
656    653  
657    654  
658    655  
659    656  
660    657  
661    658  
662    659  
663    660  
664    661  
665    662  
666    663  
667    664  
668    665  
669    666  
670    667  
671    668  
672    669  
673    670  
674    671  
675    672  
676    673  
677    674  
678    675  
679    676  
680    677  
681    678  
682    679  
683    680  
684    681  
685    682  
686    683  
687    684  
688    685  
689    686  
690    687  
691    688  
692    689  
693    690  
694    691  
695    692  
696    693  
697    694  
698    695  
699    696  
700    697  
701    698  
702    699  
703    700  
704    701  
705    702  
706    703  
707    704  
708    705  
709    706  
710    707  
711    708  
712    709  
713    710  
714    711  
715    712  
716    713  
717    714  
718    715  
719    716  
720    717  
721    718  
722    719  
723    720  
724    721  
725    722  
726    723  
727    724  
728    725  
729    726  
730    727  
731    728  
732    729  
733    730  
734    731  
735    732  
736    733  
737    734  
738    735  
739    736  
740    737  
741    738  
742    739  
743    740  
744    741  
745    742  
746    743  
747    744  
748    745  
749    746  
750    747  
751    748  
752    749  
753    750  
754    751  
755    752  
756    753  
757    754  
758    755  
759    756  
760    757  
761    758  
762    759  
763    760  
764    761  
765    762  
766    763  
767    764  
768    765  
769    766  
770    767  
771    768  
772    769  
773    770  
774    771  
775    772  
776    773  
777    774  
778    775  
779    776  
780    777  
781    778  
782    779  
783    780  
784    781  
785    782  
786    783  
787    784  
788    785  
789    786  
790    787  
791    788  
792    789  
793    790  
794    791  
795    792  
796    793  
797    794  
798    795  
799    796  
800    797  
801    798  
802    799  
803    800  
804    801  
805    802  
806    803  
807    804  
808    805  
809    806  
810    807  
811    808  
812    809  
813    810  
814    811  
815    812  
816    813  
817    814  
818    815  
819    816  
820    817  
821    818  
822    819  
823    820  
824    821  
825    822  
826    823  
827    824  
828    825  
829    826  
830    827  
831    828  
832    829  
833    830  
834    831  
835    832  
836    833  
837    834  
838    835  
839    836  
840    837  
841    838  
842    839  
843    840  
844    841  
845    842  
846    843  
847    844  
848    845  
849    846  
850    847  
851    848  
852    849  
853    850  
854    851  
855    852  
856    853  
857    854  
858    855  
859    856  
860    857  
861    858  
862    859  
863    860  
864    861  
865    862  
866    863  
867    864  
868    865  
869    866  
870    867  
871    868  
872    869  
873    870  
874    871  
875    872  
876    873  
877    874  
878    875  
879    876  
880    877  
881    878  
882    879  
883    880  
884    881  
885    882  
886    883  
887    884  
888    885  
889    886  
890    887  
891    888  
892    889  
893    890  
894    891  
895    892  
896    893  
897    894  
898    895  
899    896  
900    897  
901    898  
902    899  
903    900  
904    901  
905    902  
906    903  
907    904  
908    905  
909    906  
910    907  
911    908  
912    909  
913    910  
914    911  
915    912  
916    913  
917    914  
918    915  
919    916  
920    917  
921    918  
922    919  
923    920  
924    921  
925    922  
926    923  
927    924  
928    925  
929    926  
930    927  
931    928  
932    929  
933    930  
934    931  
935    932  
936    933  
937    934  
938    935  
939    936  
940    937  
941    938  
942    939  
943    940  
944    941  
945    942  
946    943  
947    944  
948    945  
949    946  
950    947  
951    948  
952    949  
953    950  
954    951  
955    952  
956    953  
957    954  
958    955  
959    956  
960    957  
961    958  
962    959  
963    960  
964    961  
965    962  
966    963  
967    964  
968    965  
969    966  
970    967  
971    968  
972    969  
973    970  
974    971  
975    972  
976    973  
977    974  
978    975  
979    976  
980    977  
981    978  
982    979  
983    980  
984    981  
985    982  
986    983  
987    984  
988    985  
989    986  
990    987  
991    988  
992    989  
993    990  
994    991  
995    992  
996    993  
997    994  
998    995  
999    996  
1000   997  
1001   998  
1002   999  
1003   1000  
1004   1001  
1005   1002  
1006   1003  
1007   1004  
1008   1005  
1009   1006  
1010   1007  
1011   1008  
1012   1009  
1013   1010  
1014   1011  
1015   1012  
1016   1013  
1017   1014  
1018   1015  
1019   1016  
1020   1017  
1021   1018  
1022   1019  
1023   1020  
1024   1021  
1025   1022  
1026   1023  
1027   1024  
1028   1025  
1029   1026  
1030   1027  
1031   1028  
1032   1029  
1033   1030  
1034   1031  
1035   1032  
1036   1033  
1037   1034  
1038   1035  
1039   1036  
1040   1037  
1041   1038  
1042   1039  
1043   1040  
1044   1041  
1045   1042  
1046   1043  
1047   1044  
1048   1045  
1049   1046  
1050   1047  
1051   1048  
1052   1049  
1053   1050  
1054   1051  
1055   1052  
1056   1053  
1057   1054  
1058   1055  
1059   1056  
1060   1057  
1061   1058  
1062   1059  
1063   1060  
1064   1061  
1065   1062  
1066   1063  
1067   1064  
1068   1065  
1069   1066  
1070   1067  
1071   1068  
1072   1069  
1073   1070  
1074   1071  
1075   1072  
1076   1073  
1077   1074  
1078   1075  
1079   1076  
1080   1077  
1081   1078  
1082   1079  
1083   1080  
1084   1081  
1085   1082  
1086   1083  
1087   1084  
1088   1085  
1089   1086  
1090   1087  
1091   1088  
1092   1089  
1093   1090  
1094   1091  
1095   1092  
1096   1093  
1097   1094  
1098   1095  
1099   1096  
1100   1097  
1101   1098  
1102   1099  
1103   1100  
1104   1101  
1105   1102  
1106   1103  
1107   1104  
1108   1105  
1109   1106  
1110   1107  
1111   1108  
1112   1109  
1113   1110  
1114   1111  
1115   1112  
1116   1113  
1117   1114  
1118   1115  
1119   1116  
1120   1117  
1121   1118  
1122   1119  
1123   1120  
1124   1121  
1125   1122  
1126   1123  
1127   1124  
1128   1125  
1129   1126  
1130   1127  
1131   1128  
1132   1129  
1133   1130  
1134   1131  
1135   1132  
1136   1133  
1137   1134  
1138   1135  
1139   1136  
1140   1137  
1141   1138  
1142   1139  
1143   1140  
1144   1141  
1145   1142  
1146   1143  
1147   1144  
1148   1145  
1149   1146  
1150   1147  
1151   1148  
1152   1149  
1153   1150  
1154   1151  
1155   1152  
1156   1153  
1157   1154  
1158   1155  
1159   1156  
1160   1157  
1161   1158  
1162   1159  
1163   1160  
1164   1161  
1165   1162  
1166   1163  
1167   1164  
1168   1165  
1169   1166  
1170   1167  
1171   1168  
1172   1169  
1173   1170  
1174   1171  
1175   1172  
1176   1173  
1177   1174  
1178   1175  
1179   1176  
1180   1177  
1181   1178  
1182   1179  
1183   1180  
1184   1181  
1185   1182  
1186   1183  
1187   1184  
118
```


Rewriting the HMR Map Task

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

```
1 public class Mapper<KEYIN, VALUEIN,  
2     KEYOUT, VALUEOUT> {  
3  
4     1 (defn coarse  
5     2     [^org.apache.hadoop.mapreduce.Mapper$Context reader  
6     3     ^org.apache.hadoop.mapreduce.Mapper mapper  
7     4     ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8     5     (let [records-on-disk (new InputIterator reader)]  
9     6     (ohua  
10    7     (smap  
11    8     (algo compute-and-output [ [line content] ]  
12    9     (let [kv-pairs (hmr-map line content mapper)]  
13    10     (smap  
14    11     (algo output-side [ [k v] ] (output k v writer))  
15    12     kv-pairs)))  
16    13     records-on-disk))))  
17  
18    }}
```

(a) Hadoop

```
1 public class Output {  
2     @defsf public  
3     void output(Object key, Object value, Context ctxt) {  
4         ctxt.write(key, value); }}
```

Java

Ohua

o.output(k, v, writer); (output k v writer)

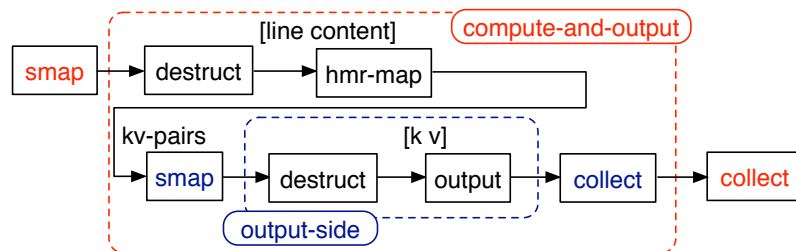
Rewriting the HMR Map Task

BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

```
1 public class Mapper<KEYIN, VALUEIN,  
2   KEYOUT, VALUEOUT> {  
3  
4   1 (defn coarse  
5     2 [^org.apache.hadoop.mapreduce.Mapper$Context reader  
6       3 ^org.apache.hadoop.mapreduce.Mapper mapper  
7         4 ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8       5 (let [records-on-disk (new InputIterator reader)]  
9         6 (ohua  
10          7 (smap  
11            8 (algo compute-and-output [ [line content] ]  
12              9 (let [kv-pairs (hmr-map line content mapper)]  
13                10 (smap  
14                  11 (algo output-side [ [k v] ] (output k v writer))  
15                    12 kv-pairs)))  
16                  13 records-on-disk))))  
17  })  
18 }
```

(a) Hadoop



```
1 public class Output {  
2   @defsf public  
3   void output(Object key, Object value, Context ctxt) {  
4     ctxt.write(key, value); }  
}
```

Java

Ohua

o.output(k, v, writer); (output k v writer)

Rewriting the HMR Map Task

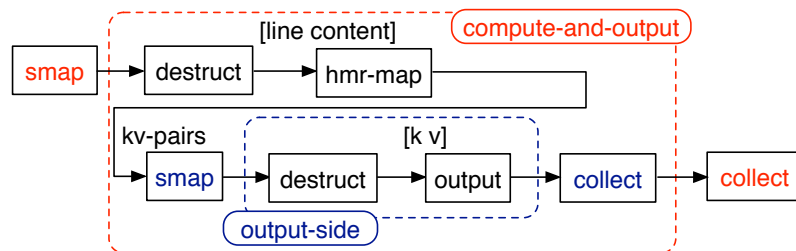
BDS core study → Implicit Parallel Programming → BDS core rewrite

Ready! Set! Go!

4 variants: Coarse (C) - Coarse Input Fine Output (CIFO) -
Fine Input Coarse Output (FICO) - Fine (F)

```
1 public class Mapper<KEYIN, VALUEIN,  
2     KEYOUT, VALUEOUT> {  
3  
4     (defn coarse  
5         [^org.apache.hadoop.mapreduce.Mapper$Context reader  
6         ^org.apache.hadoop.mapreduce.Mapper mapper  
7         ^org.apache.hadoop.mapreduce.Mapper$Context writer]  
8         (let [records-on-disk (new InputIterator reader)]  
9             (ohua  
10                (smap  
11                    (algo compute-and-output [ [line content] ]  
12                        (let [kv-pairs (hmr-map line content mapper)]  
13                            (smap  
14                                (algo output-side [ [k v] ] (output k v writer))  
15                                kv-pairs)))  
16                        records-on-disk))))  
17         })  
18 }
```

(a) Hadoop



```
1 public class Output {  
2     @defsf public  
3     void output(Object key, Object value, Context ctxt) {  
4         ctxt.write(key, value); }  
5 }
```

Java

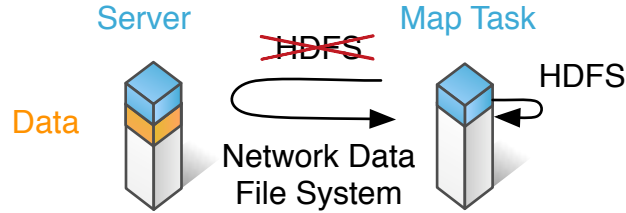
Ohua

o.output(k, v, writer); (output k v writer)

Evaluation - Setup

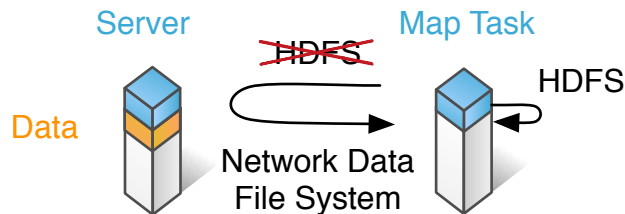
Intel NUMA 2.6 GHZ

- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM

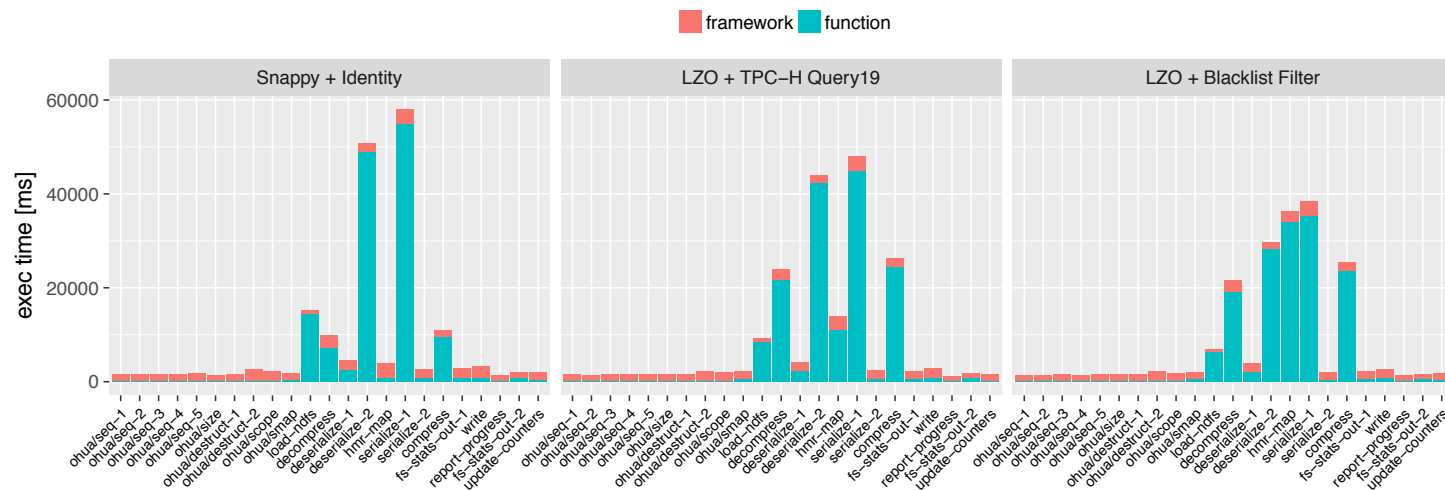


- SequenceFile
- JSON
- Snappy, LZ0
- TPC-H Parts Table

- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



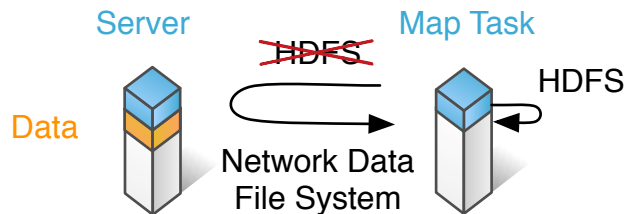
- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table



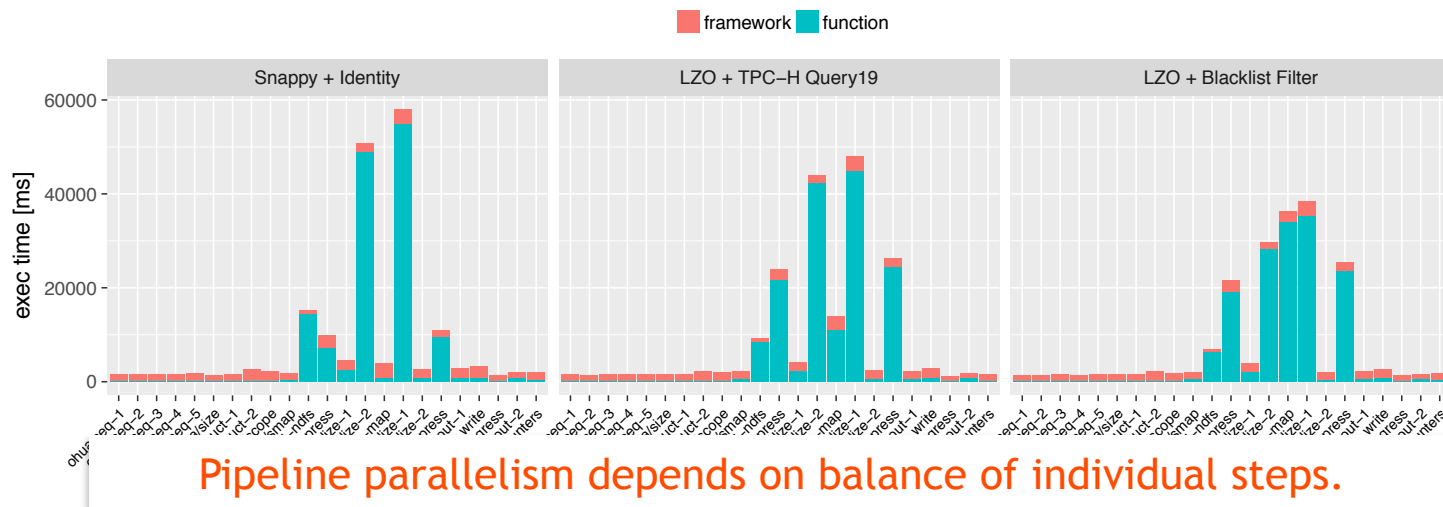
Evaluation - Execution Breakdown

Intel NUMA 2.6 GHZ

- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



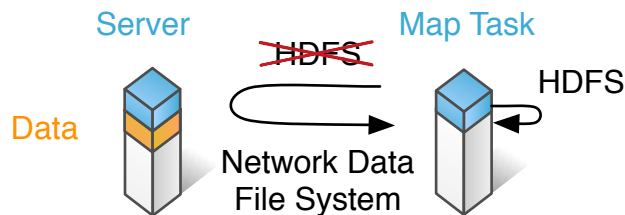
- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table



Evaluation - Throughput Analysis

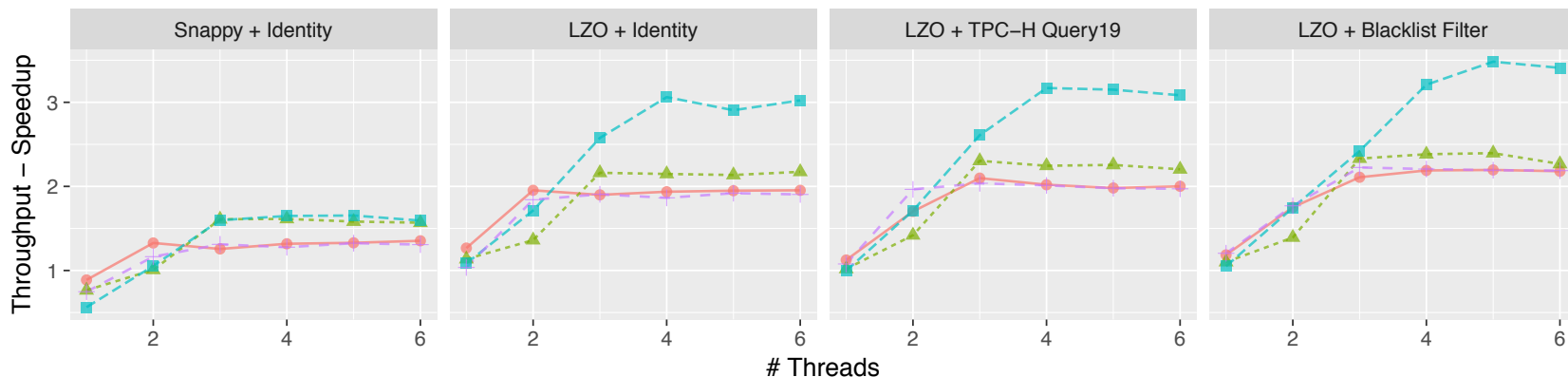
Intel NUMA 2.6 GHZ

- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table

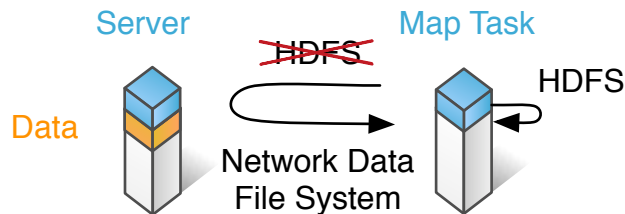
—●— C —▲— CIFO —■— F —+— FICO



Evaluation - Throughput Analysis

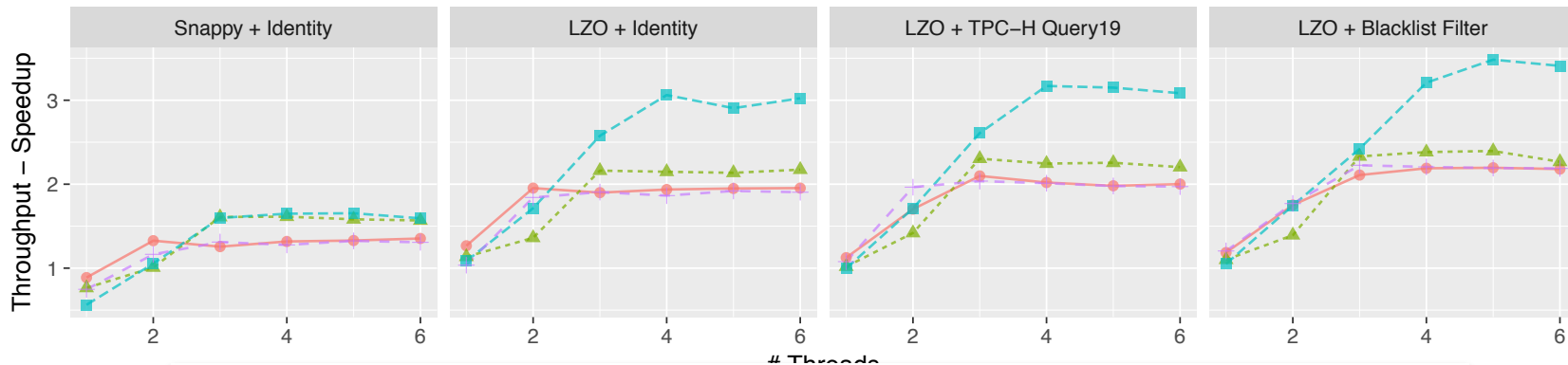
Intel NUMA 2.6 GHZ

- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table

—●— C —▲— CIFO —■— F —+— FICO

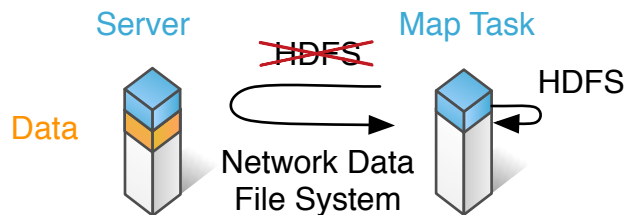


Speedup of up to 3.5x for compute-intensive configurations!

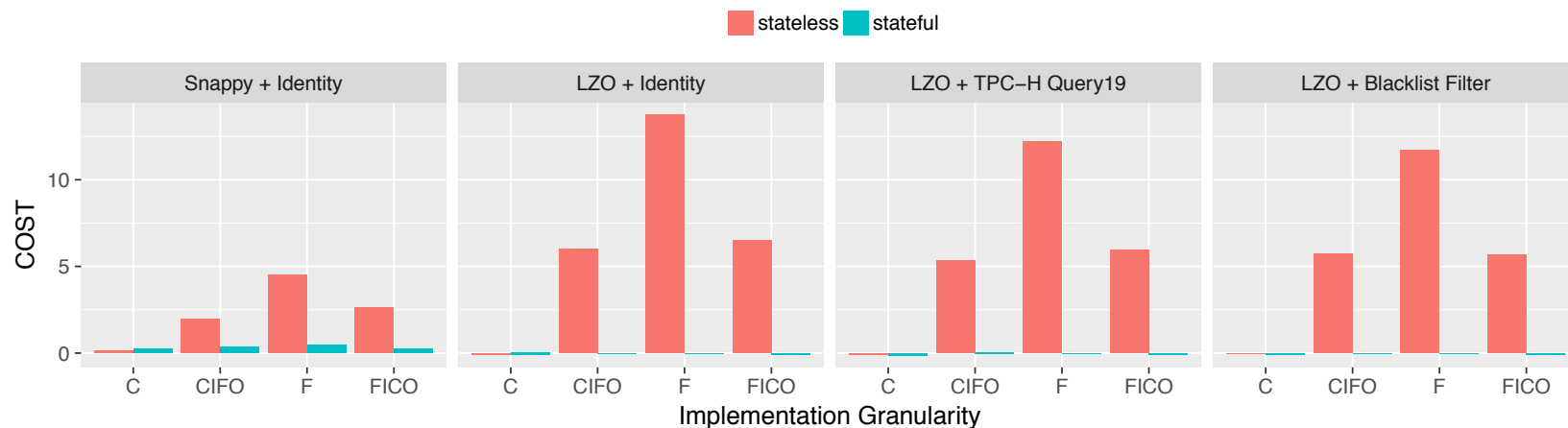
Evaluation - COST Analysis

Intel NUMA 2.6 GHZ

- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table

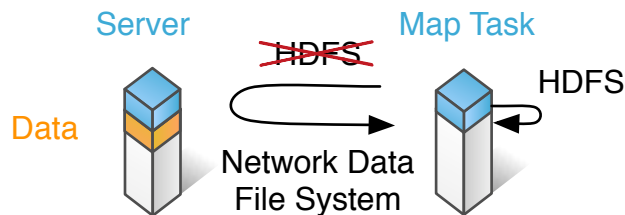


Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost? (HotOS'15).

Evaluation - COST Analysis

Intel NUMA 2.6 GHZ

- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table



Processing is inherently stateful! → Cache for (de)compression.

Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability: but at what cost? (HOTOS 15).

- Data processing cores of state-of-the-art BDS are all similar,
- and be rewritten with low effort to scale on multi- and manicures.

If you do or (write a new BDS):
Consider using an implicit parallel programming language such as Ohua!

- ☑ Clean, concise and modular code structure,
- ☑ without concurrency abstractions.
- ☑ Use the associated compiler and runtime system to adapt to scale to new/heterogenous HW