高知工科大学
KOCHI UNIVERSITY OF TECHNOLOGY

Functional Models of Hadoop MapReduce with Application to Scan

Kiminori Matsuzaki
Kochi University of Technology

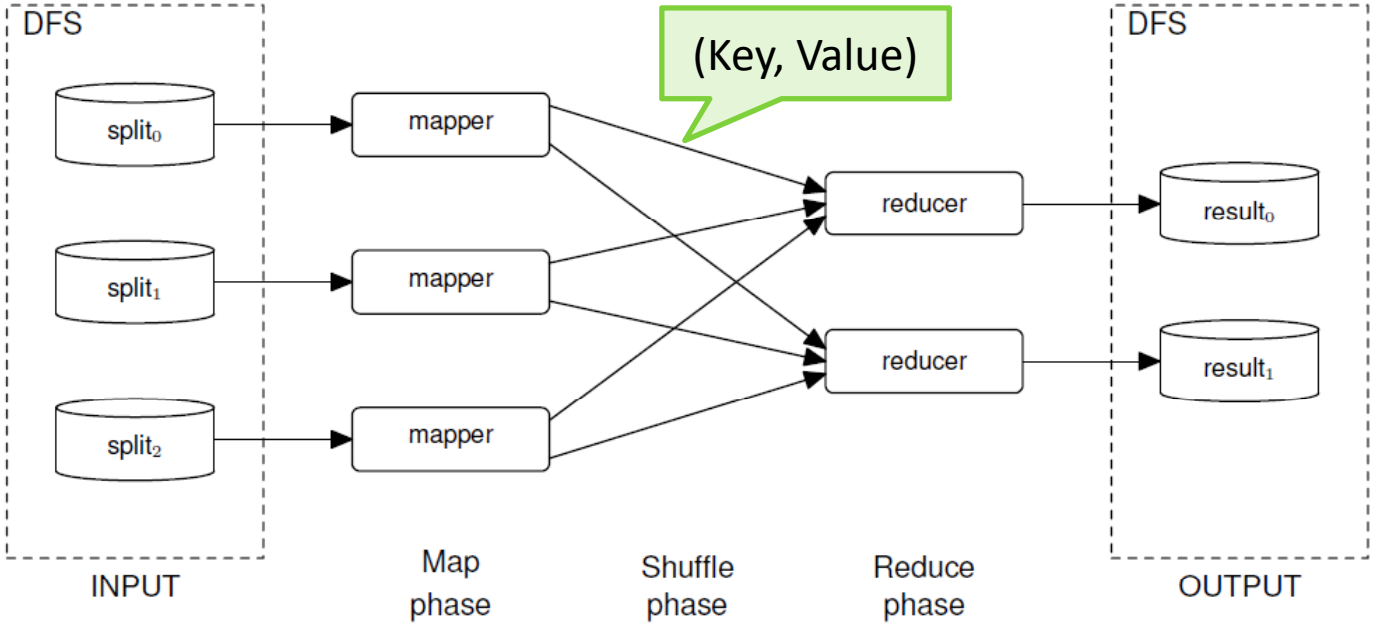
Brief History of (Hadoop) MapReduce

- 2004: Google proposed MapReduce [OSDI 2004]
- 2004-2006: Open-Source MapReduce in Nutch
- 2006-: Hadoop project
- 2011 Dec.: Hadoop 1.0.0
- 2012: Industry standard in distributed processing
- 2013 Oct.: Hadoop 2.2.0 (first stable ver. 2.x)

[http://research.yahoo.com/files/cutting.pdf]
[http://www.guruzon.com/6/introduction/map-reduce/history-of-map-reduce]
[http://hadoop.apache.org/releases.html]

MapReduce in a NutShell

- 3 phases, 2 user-defined functions



Misunderstandings

高知工科大学
KOCHI UNIVERSITY OF TECHNOLOGY

- The OSDI paper said: “map/reduce were inspired from those in functional programming”

Functional Community

- Map/reduce in MapReduce differs in terms of
  - target data (set-like data, not lists or trees)
  - how they work (map/reduce are applied independently)
  - no associativity needed in reduce

DB Community

- “MapReduce: A major step backwards” (2008)
  - No indexing, poor impl., DBMS-incompatibility, etc.

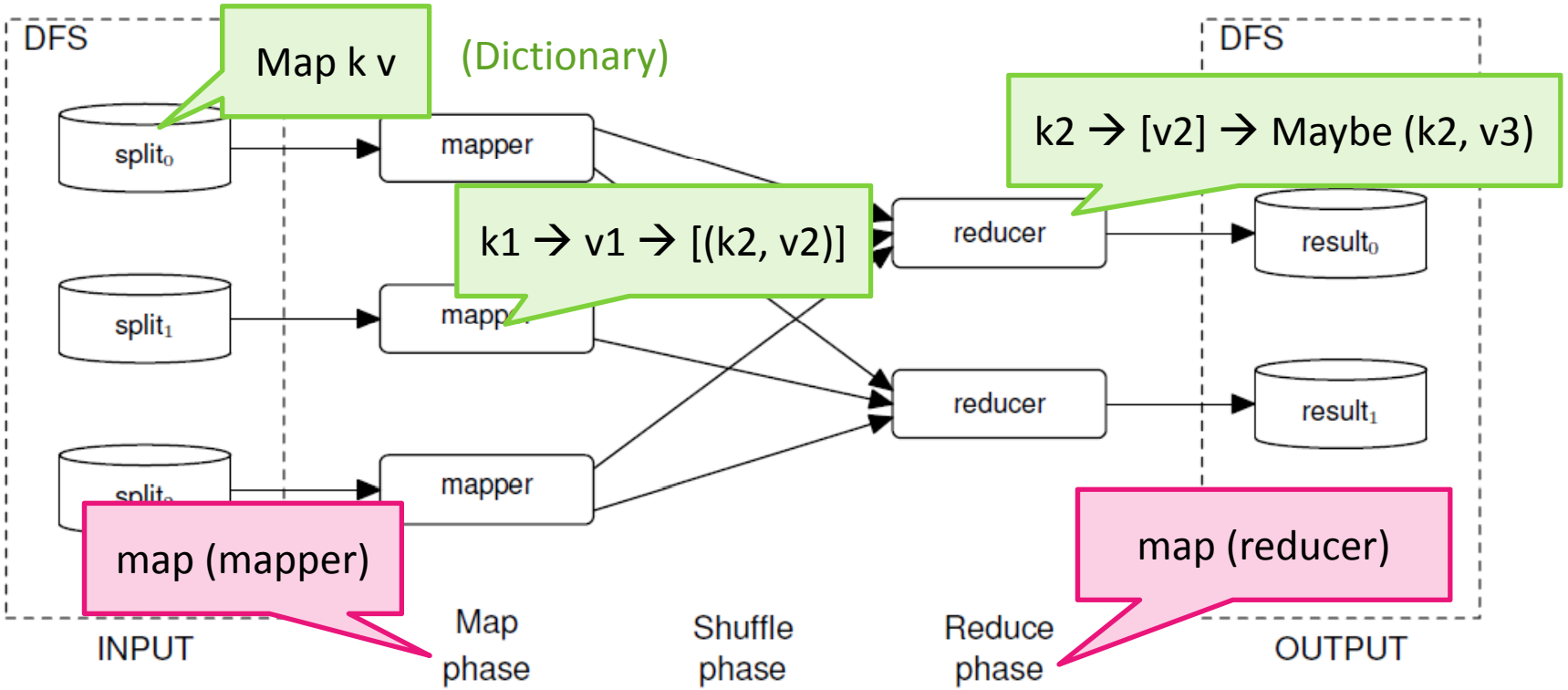
Functional Models

高知工科大学
KOCHI UNIVERSITY OF TECHNOLOGY

A functional model describes clearly the computation of the framework, especially by using the types

- R. Lämmel:
  "Google's MapReduce programming model -- revisited."
  Science of Computer Programming, 2008
  - Provides a functional model of Google's MapReduce
  - Model is written in Haskell

Lämmel's Functional Model



Why Functional Model Matters?

高知工科大学
KOCHI UNIVERSITY OF TECHNOLOGY

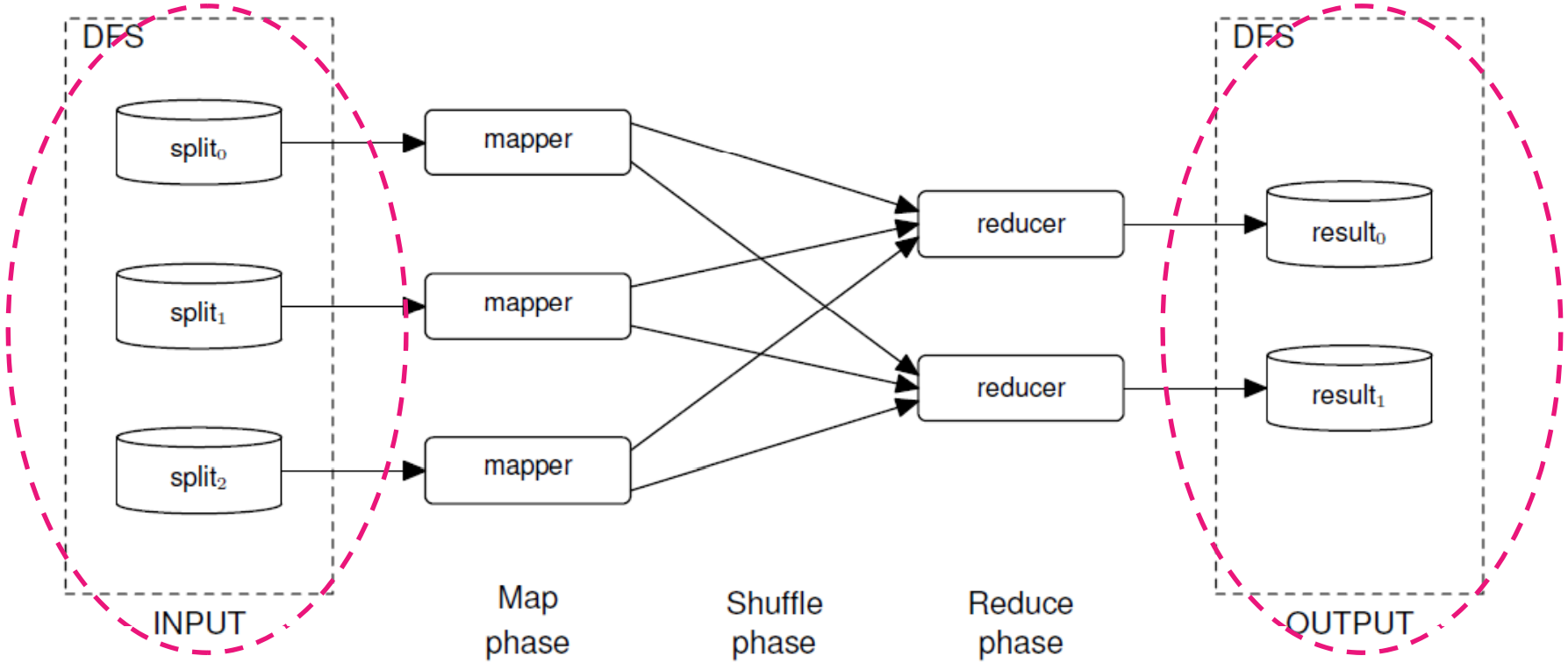
- Understanding the computation
  - Avoid misunderstandings
- Proof of Correctness
  - Developing functional code to check
  - Proof using Coq (Related work [Ono 2011], [Jiang 2014])
- Program Calculation
  - Developing program-transformation rules
- Cost Model
  - Performance tuning (Related work [Dörre 2014])

Contributions In The Paper

高知工科大学
KOCHI UNIVERSITY OF TECHNOLOGY

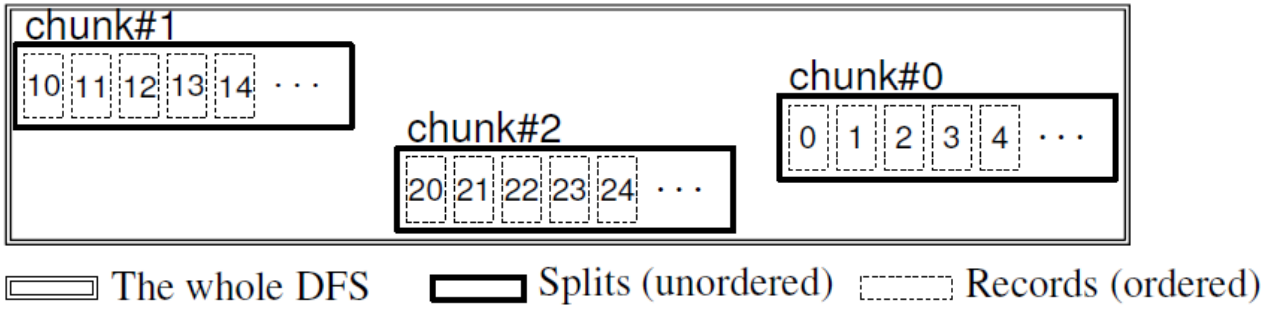
- Two functional models of Hadoop MapReduce
  - Low-level model (based on implementation)
    - Nested input/output
    - Stateful mapper/reducer
    - Detailed modeling of Shuffling phase
  - High-level model (user-friendly specification)
    - for “secondary-sorting” technique
- Scan (prefix-sums) algorithm on the models
  - Three-phase algorithm (L-reduce, G-scan, L-scan)
  - BSP-based 2-superstep algorithm

Nested Input/Output in Hadoop



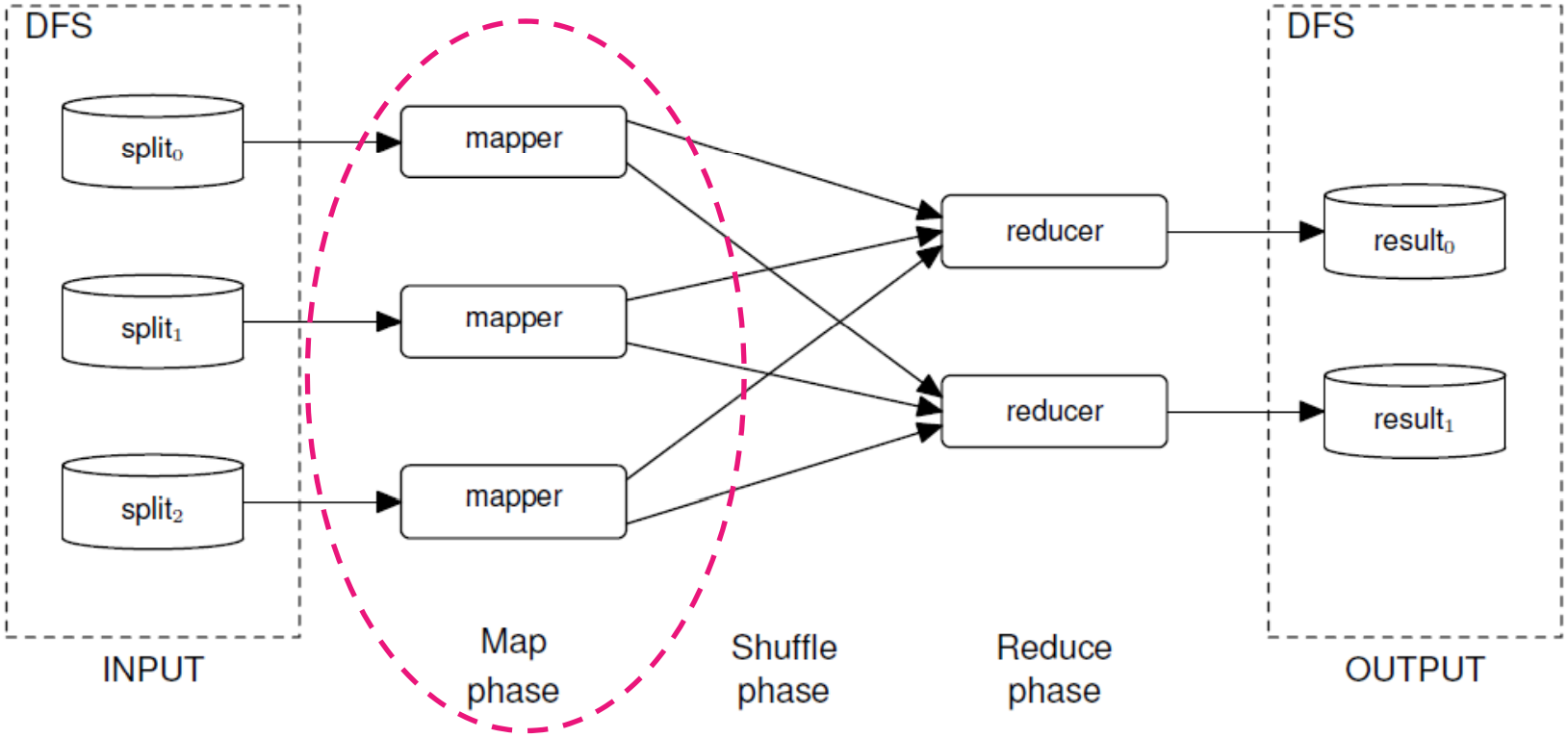
Nested Input/Output in Hadoop

- Data → Split → Record



- Split → Unorderd → Bag (multi-set)
  - Parallel (independently / order may change)
- Record →Ordered → List
  - Sequential (one-by-one / order preserved)

Mapper Class in Hadoop



Mapper Class in Hadoop

高知工科大学 Kochi University of Technology

- Definition in Hadoop

```
class Mapper {
   void setup(Context);
   void map(Key, Value, Context);
   void cleanup(Context);

   void run(Context c) {
      setup(c);
      for (kv : split) {
          map(kv.k, kv.v, c);
      }
      cleanup(c);
   }
}
```

- A simple case == map

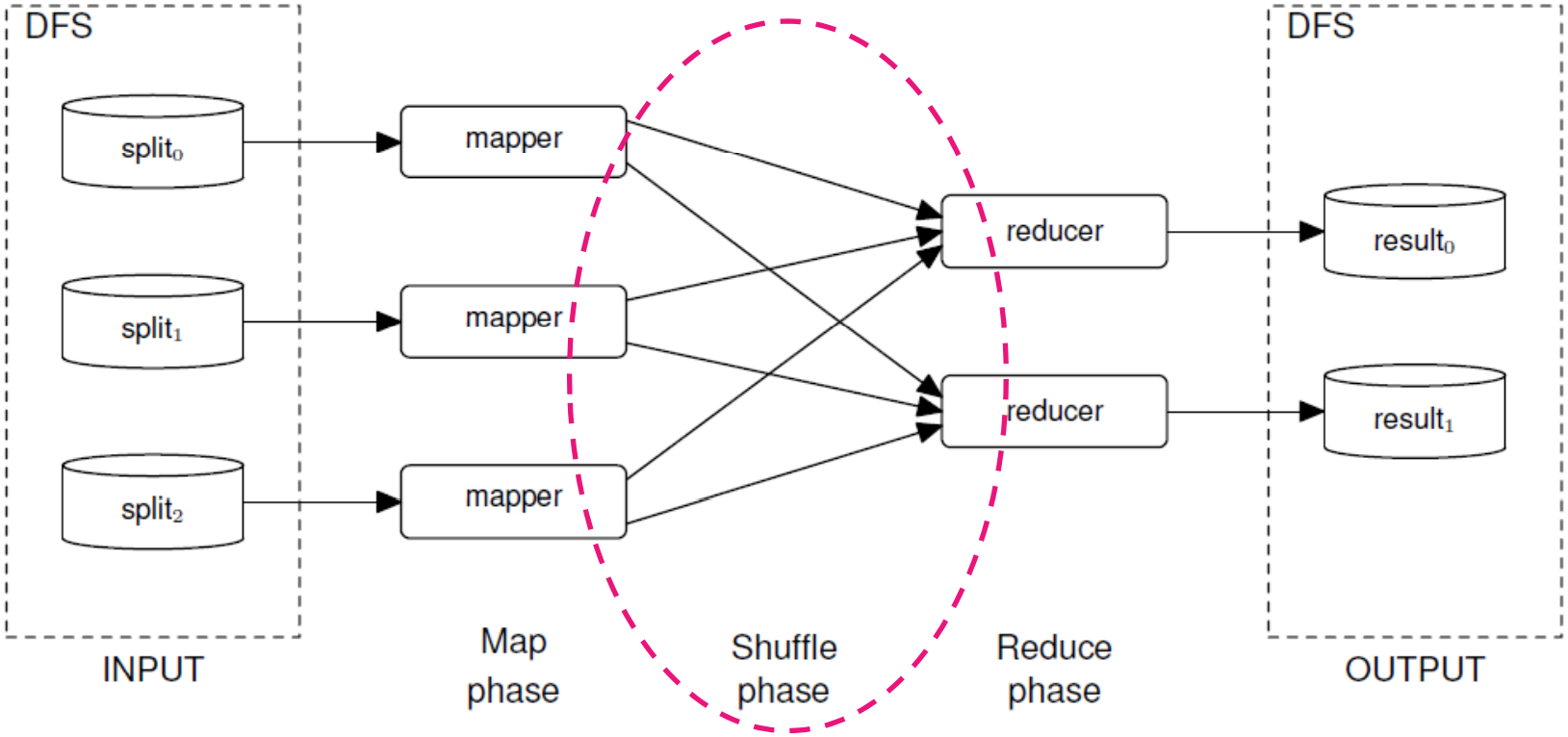
$$
\begin{aligned}
&mkMapper1 :: ((k_1, v_1) \to [(k_2, v_2)]) && \text{-- } f_{\text{map}} \\
&\qquad \to [(k_1, v_1)] \to [(k_2, v_2)] && \text{-- input/output} \\
&mkMapper1\ f_{\text{map}} = \mathit{flatten} \circ \mathit{map}\ f_{\text{map}}
\end{aligned}
$$

- A stateful case == foldl

$$
\begin{aligned}
&mkMapper2 :: att && \text{-- } f_{\text{setup}} \\
&\qquad \to (att \to (k_1, v_1) \to att) && \text{-- } f_{\text{map}} \\
&\qquad \to (att \to [(k_2, v_2)]) && \text{-- } f_{\text{cleanup}} \\
&\qquad \to [(k_1, v_1)] \to [(k_2, v_2)] && \text{-- input/output} \\
&mkMapper2\ f_{\text{setup}}\ f_{\text{map}}\ f_{\text{cleanup}} = f_{\text{cleanup}} \circ \mathit{foldl}\ f_{\text{map}}\ f_{\text{setup}}
\end{aligned}
$$

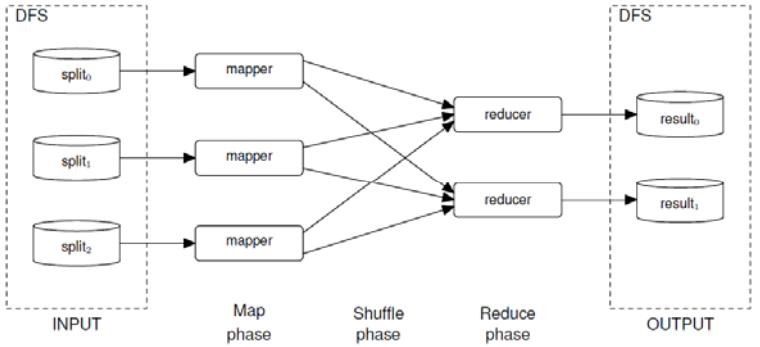
- A state-monadic map (in paper)

Shuffle Phase in Hadoop



Shuffle Phase in Hadoop

- 3 sub-phases
  1. Partitioning (cf. reduce job)
  2. Sorting
  3. Grouping (cf. reducer func.)



$$
\begin{aligned}
&shuffleMR :: ((k_2, v_2) \to Integer) && \text{-- } hashP\\
&\quad \to (k_2 \to k_2 \to Ordering) && \text{-- } compS\\
&\quad \to (k_2 \to k_2 \to Ordering) && \text{-- } compG\\
&\quad \to Bag[(k_2, v_2)] && \text{-- } input\\
&\quad \to Bag[(k_2, [v_2])] && \text{-- } output\\
&shuffleMR\ hashP\ compS\ compG\ input\\
&\quad = \textbf{let}\ aftP = grpByID\ \$\ map_{\mathrm{B}}\ (applyW\ \ hashP)\ \$\ flatten_{\mathrm{B}}\ input\\
&\qquad\ aftS = map_{\mathrm{B}}\ (sortByKey\ \ compS)\ \ aftP\\
&\qquad \textbf{in}\ \ map_{\mathrm{B}}\ (grpByKey\ \ compG)\ \ aftS
\end{aligned}
$$

Toward High-Level Model

- 3-phase Implementation
  1. Partitioning (cf. reduce job)
  2. Sorting
  3. Grouping (cf. reducer func.)

Sorting and grouping should be consistent

- Possibly any comparator

Let comp a b = |a – b| < 2
then [3, 4, 5, 7] ➔ [3, 4, 5], [7]

- Sorting after grouping
  1. Partitioning (cf. reduce job)
  2. Grouping (cf. reducer func.)
  3. Sorting

cf. secondary-sorting

- Explicit keys

Use (kP, kG, kS) for keys

Difference from Low-Level Model

高知工科大学 Kochi University of Technology

Low-Level Model

$$
\begin{aligned}
shuffleMR :: & ((k_2, v_2) \rightarrow Integer) && \text{-- } hashP \\
& \rightarrow (k_2 \rightarrow k_2 \rightarrow Ordering) && \text{-- } compS \\
& \rightarrow (k_2 \rightarrow k_2 \rightarrow Ordering) && \text{-- } compG \\
& \rightarrow Bag[(k_2, v_2)] && \text{-- } input \\
& \rightarrow Bag[(k_2, [v_2])] && \text{-- } output
\end{aligned}
$$

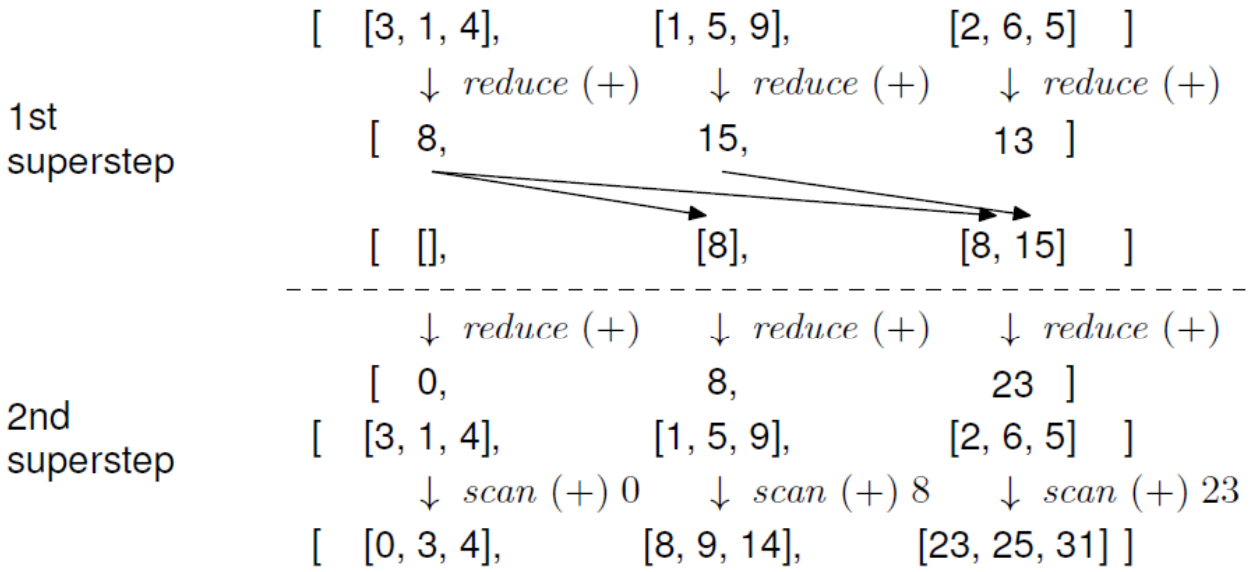
Comparator ➔ Type Class

High-Level Model

$$
\begin{aligned}
shuffleMR2 :: & \text{ Eq } k_P \Rightarrow \text{Ord } k_G \Rightarrow \text{Ord } k_S && \text{-- (requirements)} \\
& \Rightarrow Bag\ [((k_P, k_G, k_S), v_2)] && \text{-- input} \\
& \rightarrow Bag\ [((k_P, k_G, k_S), [v_2])] && \text{-- output}
\end{aligned}
$$

Explicit 3-keys

Case Study: BSP-based Scan



- Challenge

  A sequential program ➔ A MapReduce program

A Program for BSP-scan

- Specification: based on nested-lists

$$
\begin{aligned}
&scanBSP :: (a \to a \to a) \to a \to Int \to [[a]] \to [[a]] \\
&scanBSP\ (\oplus)\ e\ p\ xss \\
&\quad = \mathbf{let}\ ys = map\ (foldl\ (\oplus)\ \iota_{\oplus})\ xss \\
&\qquad\quad zss = map\ (\lambda p.\ take\ p\ ys)\ [0..(p-1)] \quad \text{-- 1st superstep} \\
&\qquad \mathbf{in}\ map\ scan'\ (zip\ xss\ zss) \quad \text{-- 2nd superstep} \\
&\qquad \mathbf{where}\ scan'\ (xs, zs) = scan\ (\oplus)\ (foldl\ (\oplus)\ e\ zs)\ xs
\end{aligned}
$$

communication

A MapReduce Program for BSP-scan

- 1-pass MapReduce with state-monadic Mapper

```
scanBSPMR (⊕) e pp xb
  = mapReduceH mapper reducer xb
    where
      mapper = mkMapper3 fsetup fmap fcleanup
      fsetup − (0, ι⊕)
      fmap (_, s) (k, v) = let p = div k 3
                           in ((p, s ⊕ v), [((p, 1, k), v)])
      fcleanup (p, s) = map (\p' → ((p', 1, −1000 + p), s)) [p + 1..pp − 1]
      reducer = mkReducer1 freduce
      freduce ((p, _, _), vs) = let zs = take p vs
                                    xs = drop p vs
                                in zip [(3 ∗ p)..(3 ∗ p + 2)]
                                       (scan (⊕) (foldl (⊕) e zs) xs)
```

Send input-list to reducer

Design 3-keys (indices)

Separate input-list

Related Work

- K. Ono et al.: *SEFM 2011*.
- F. Jiang et al.: *IEICE Japanese Journal* 2014.
  - Proof with Coq system and code extraction

- J. Dörre, et al.: *Concurrency and Computation: Practice and Experience,* 2014.
  - Developed a cost model and performance tuning

Conclusion

- Summary
  - Two functional models of Hadoop MapReduce
    - Low level: based in the implementation of Hadoop
    - High level: User-friendly Shuffle phase
  - Scan (prefix-sum) algorithms on MapReduce
- Future Work
  - Coq library with code generation
  - Comparison of two scan algorithms
  - Functional models for big-graph processing (Pregel)