Paradigma MapReduce a Apache Hadoop

Marek Rychlý

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů

Doplňující přednáška pro GJA
2. prosince 2020

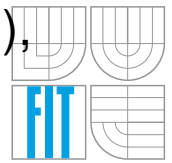
Obsah

OLTP/OLAP a BigData

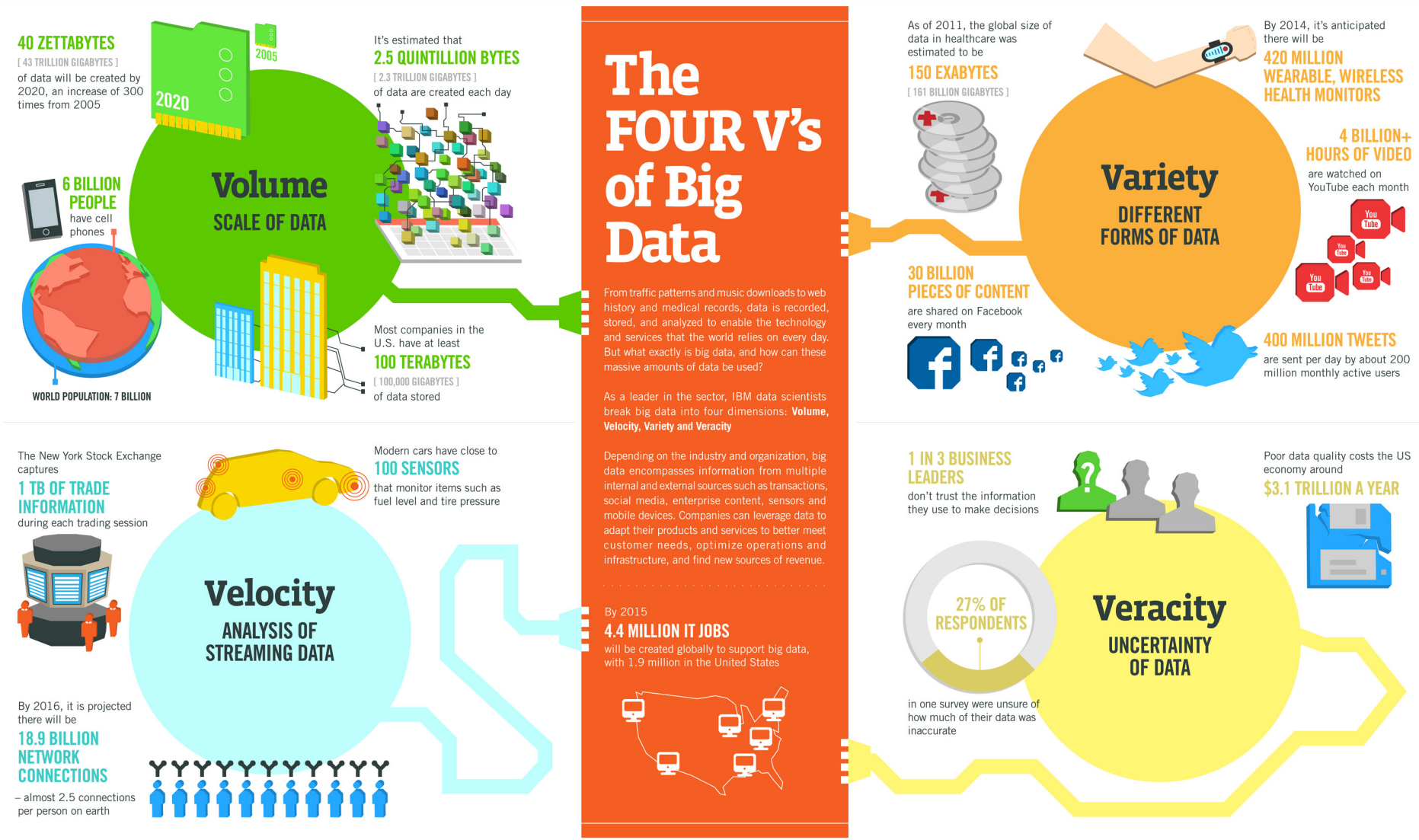
- IT bylo zvyklé pracovat s pevně strukturovanými daty.
  (např. relační a post-relační databáze s jasným schématem)
- Na nižší úrovni použití OLTP systémy, na vyšší OLAP systémy.
  (tj. „online transaction/analytical processing“, běžná práce vs. celková analýza dat)
- S nestrukturovanými daty snaha o řešení v NoSQL databázích.
  (tj. databáze bez schéma, většinou jen úložiště „klíč:hodnota“)
- Absence databázového schéma však není jediný problém.
  (např. proudy dat zpracovávané sekvenčně a real-time, tj. bez možnosti náhodného přístupu či pozastavení proudu)
- Práce s takovými daty se pak vymyká přístupu OLTP/OLAP $\Rightarrow$ BigData.
  (BigData doplňují OLTP/OLAP, nenahrazují; OLTP/OLAP stále běžně potřeba)

Proč BigData?

- Velké, nestrukturované a velmi rychle rostoucí kolekce dat.
  (pro jejich vlastnosti je není možné zpracovat běžnými přístupy)
- Vyžadují nové přístupy pro uložení, zpracování i zobrazení dat.
  (zachycení, předzpracování, uložení, hledání, sdílení/přesun, analýza, vizualizace)
- Nutné použít paralelní a distribuovaná úložiště a algoritmy/cloud.
  (data nelze uchovávat/zpracovávat centrálně pro velikost, místa zdrojů dat, výkon)
- Paralelní a distribuované zpracování přináší další problémy.
  (jak zaručit vhodnou distribuci dat a výpočtu, jak řešit nespolehlivost/výpadky zapojené infrastruktury, jak a kam zajistit doručení výsledků výpočtu, atp.)
- BigData jsou potřeba pro zpracování a dotazování dat
  - ze sociálních sítí a zpráv (Facebook, Twitter, . . .),
  - z rozsáhlých měření (data neustále generovaná tisíce senzory, statistické údaje o používání různých služeb miliony uživatelů, atp.),
  - z neustále se měnících a nestrukturovaných množin dat (telefonní hovory, internetová komunikace, video či audio proudy dat, atp.).



Proč BigData?



(diagram převzat z „The Four V's of Big Data, IBM“)

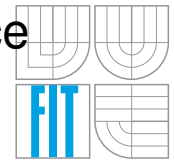
Paradigma MapReduce

- V roce 2004 Dean&Ghemawat z Google publikovali příspěvek „MapReduce: Simplified Data Processing on Large Clusters“.
- Paradigma založeno na funkcích Map a Reduce.

  (inspirováno funkcemi v jazyce Lisp a ostatních funkcionálních jazycích)

```
;; (map unary-op list1 [list2 list3 ...])
(map square '(1 2 3 4)) ;; result = (1 4 9 16)
;; (reduce binary-op list1 [list2 list3 ...])
(reduce + '(1 4 9 16)) ;; result = (+ 16 (+ 9 (+ 4 1) ) ) = 30
```

- Souběžné provádění několika Map a Reduce úloh.
  1. vstup se rozdělí na části, každá přiřazena jednomu výpoč. uzlu,
     (Lisp: ze vstupních dat vznikne několik seznamů)
  2. každý uzel souběžně spustí Map pro každý prvek vst. seznamů,
     (Lisp: paralelní provedení fce map pro každý seznam)
  3. výsledky se posbírají z uzlů a přitom seskupí podle daného klíče
     (Lisp: příprava seznamů pro fce reduce, jeden pro každou hodnotu klíče)
  4. skupiny se rozdělí mezi uzly dle hodnot klíče a každý spustí Reduce
     (Lisp: paralelní provedení fce reduce pro každý seznam)
  5. výsledky všech provedení reduce se posbírají a uloží na výstup



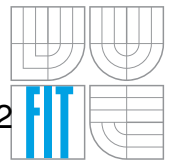
Funkce Map a Reduce

MapReduce aplikace jsou složeny z funkcí Map a Reduce definovaných nad daty reprezentovanými dvojicemi „klíč:hodnota“

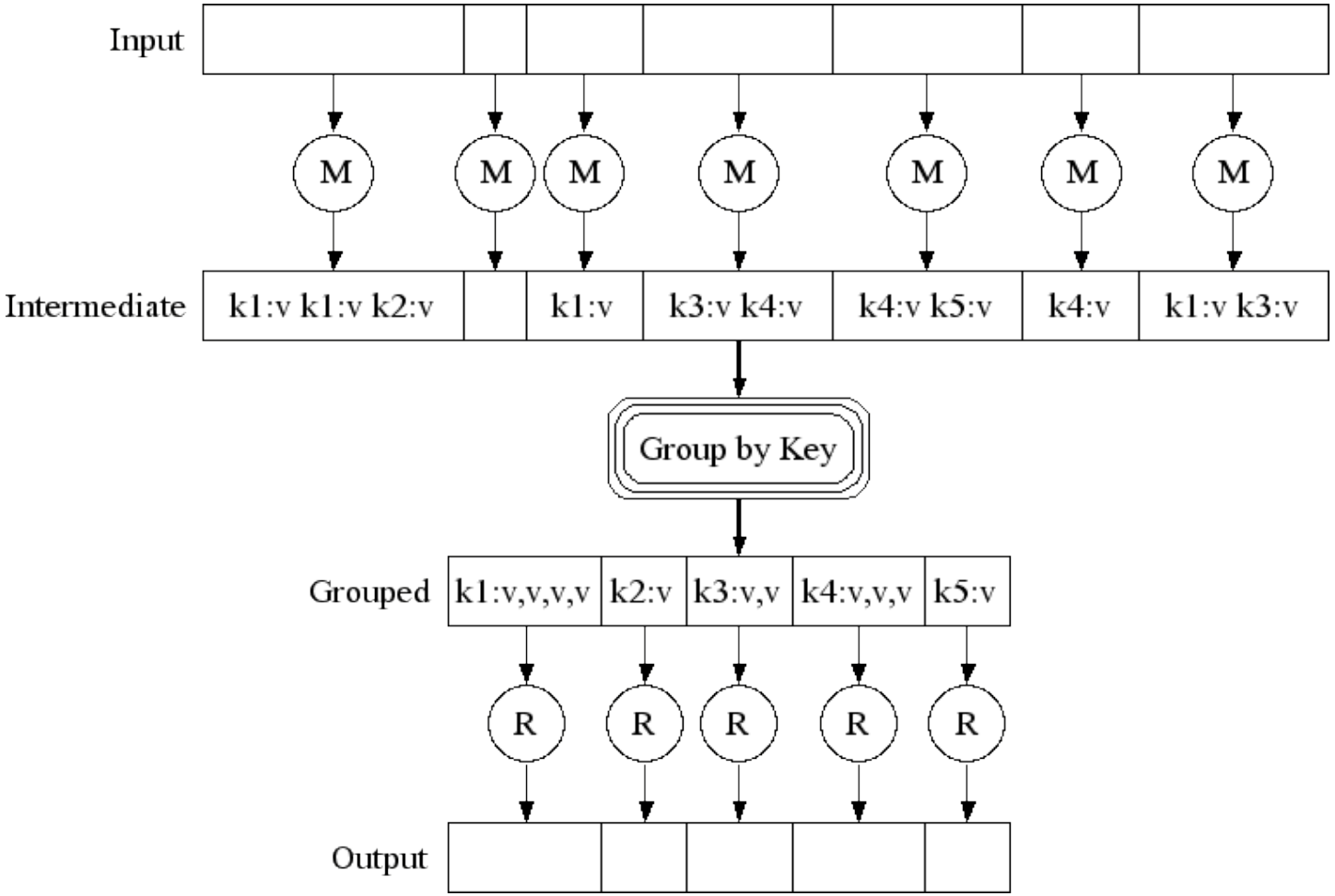
$Map(k_1, v_1) \rightarrow list(k_2, v_2)$ na každá jednotlivá vstupní data k_1:v_1 je aplikována Map, která vstupy zpracuje a jako výsledek vytvoří pro každý vstup seznam výstupů k_2:v_2

$Reduce(k_2, list(v_2)) \rightarrow list(k_3, v_3)$ hodnoty ve výstupech všech aplikací Map jsou sdruženy podle klíče a na každý takový seznam hodnot pro jednotlivý klíč je aplikována Reduce, která z nich vytvoří seznam výstupních hodnot.

- k_1 hodnota klíče položky vst. dat, např. pořadí, určuje rozdělení mezi uzly
- v_1 vlastní data položky vst. dat, tj. to, co se má zpracovat
- k_2, k_3 hodnota klíče položky výst. dat, např. název výsledku zpracování dat
- v_2 mezistupeň výsledku zpracování dat, získáno z jednotlivých vstupů
- v_3 celkový výsl. zpracování dat agregací mezivýsledků pro každý klíč k_2

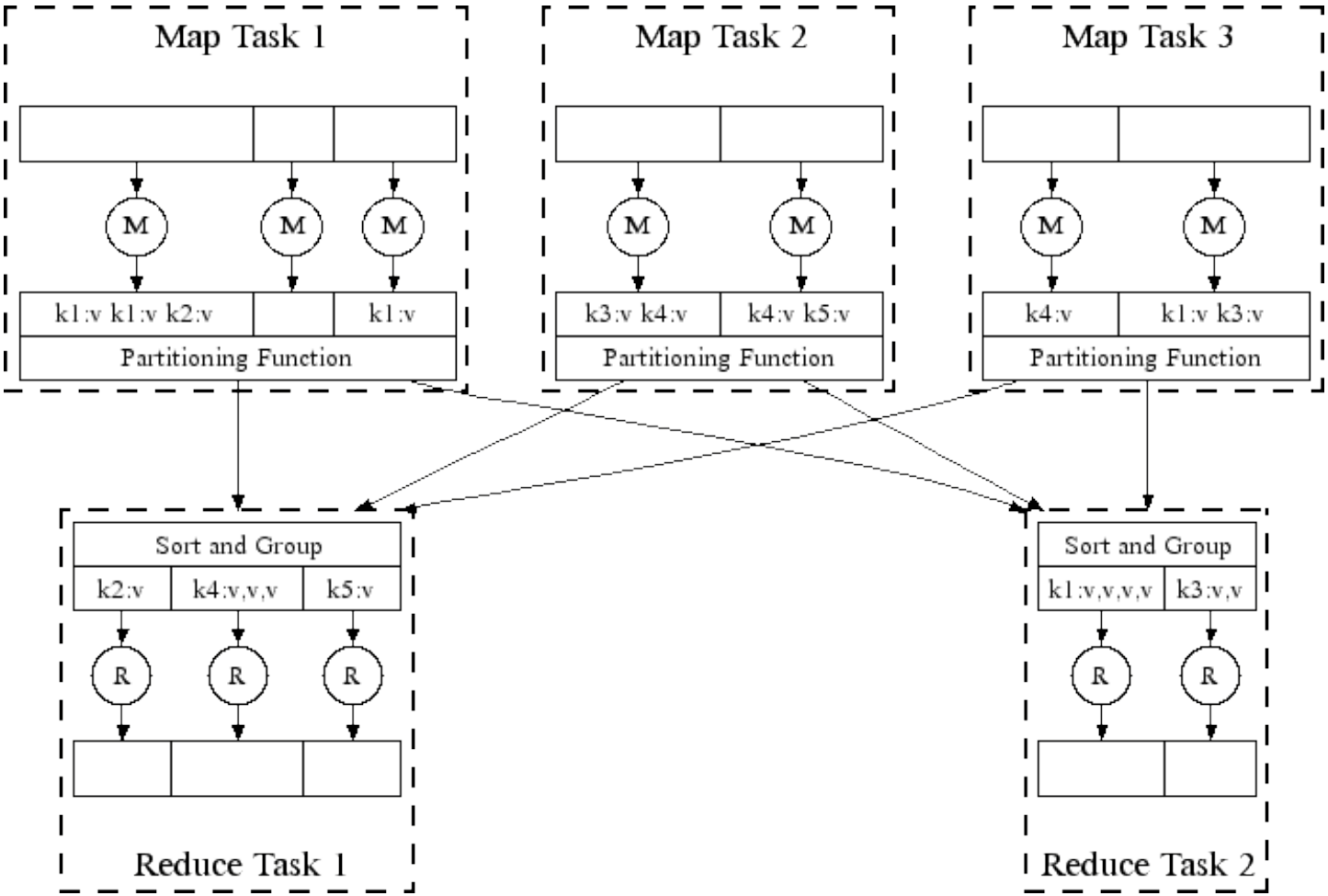


Provádění MapReduce



(diagram převzat z „MapReduce: Simplified Data Processing on Large Clusters“)

Souběžný průběh MapReduce



(diagram převzat z „MapReduce: Simplified Data Processing on Large Clusters“)

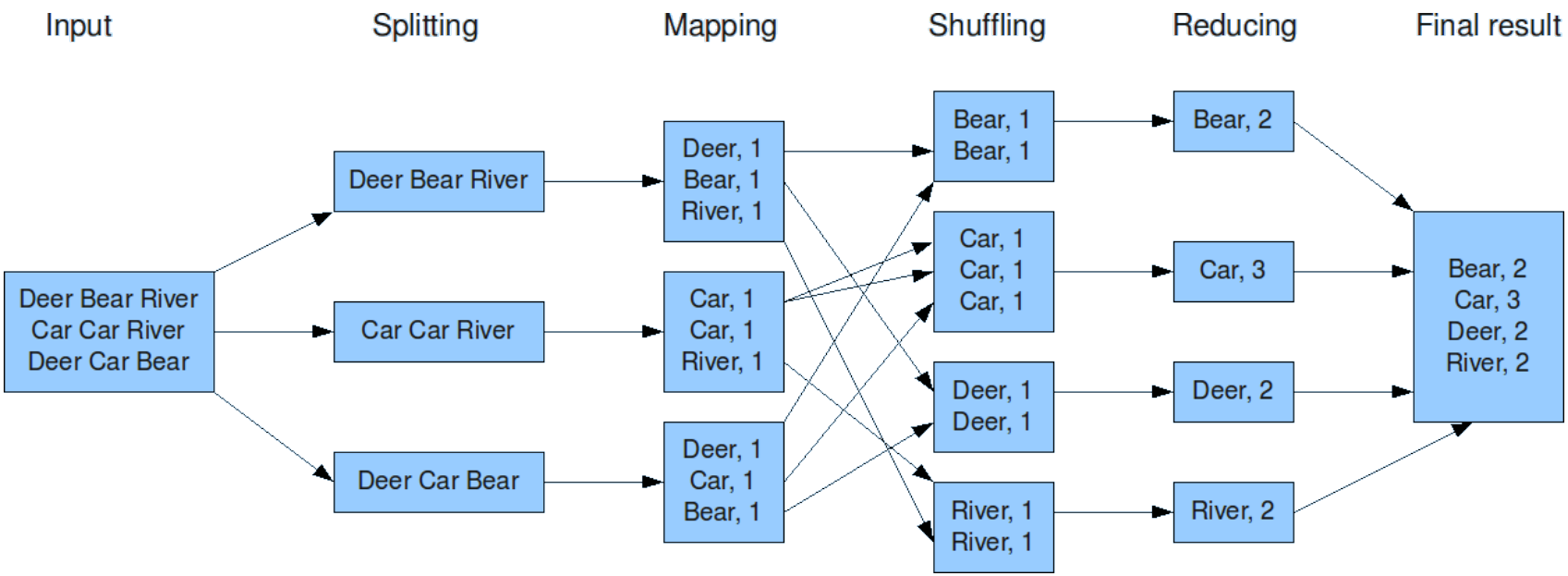
Příklad funkcí Map a Reduce (ze článku Google2004)

```
map(String input_key, String input_value):
  // input_(key,value): (document name,document contents)
  for each word w in input_value:
    EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
  // output_(key,values): (a word,a list of counts)
  int result = 0;
  for each val in intermediate_values:
    result += ParseInt(val);
  Emit(AsString(result));
```

- uvedená aplikace počítá četnost slov ve vstupních datech
- Map je spuštěno pro každý dokument (řádek) vstupu
  a pro každé slovo v dokumentu (na řádku) vytvoří výstup
  (klíč vstupu je název dokum. (číslo řádku), hodnota je obsah dokumentu (řádku);
  klíč výstupu je slovo, hodnota je četnost „1“)
- Reduce dostane na vstupu slovo a seznam jeho četností
  a jako výstup vrátí celkovou četnost slova na vstupu
  (klíč vstupu je slovo, hodnota je seznam obsahující prvek „1“ pro každý jeden výskyt slova; např. pro 3 výskyty slova je seznam $[1, 1, 1]$;
  výstupní hodnota je součet všech prvků vstupního seznamu)

Výstupy fází provádění MapReduce příkladu

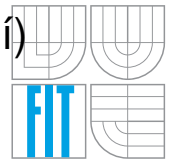


(diagram převzat z „Data Mining 2.0: Mine your data like Facebook, Twitter and Yahoo“)

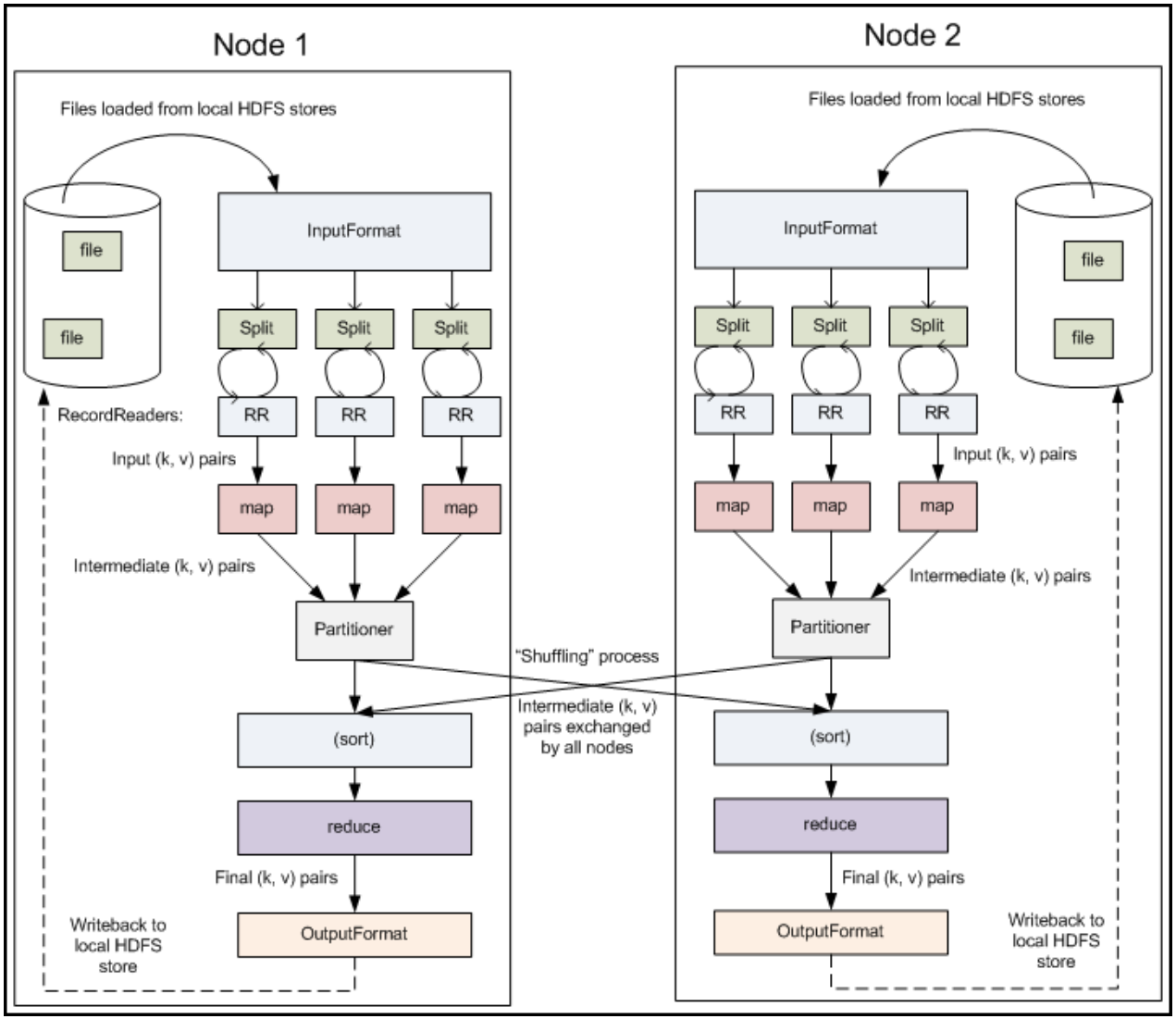
Zpracování dat a programování MapReduce

1. input&splitting
   (načtení dat ze zdroje a rozdělení mezi uzly provádějící Map)
2. Map funkce
   (vlastní provedení Map pro jednotlivé části vstupů)
3. shuffling (partitioning&comparing)
   (setřídění výstupů Map, rozdělění mezi uzly Reduce a přenesení dat)
4. Reduce funkce
   (vlastní provedení Reduce pro jednotlivé části mezivýsledků)
5. output
   (posbírání výsledků Reduce a zápis do výstupu)

- Programátor většinou řeší jen input&output a Map&Reduce.
- Splitting&partioning prováděno automaticky implementací rámce.
  (většinou rozdělení mezi uzly podle hash fce klíče, snaha o rovnoměrné rozložení)
- Comparing, tj. třídění mezidat, se provádí automaticky dle klíčů.

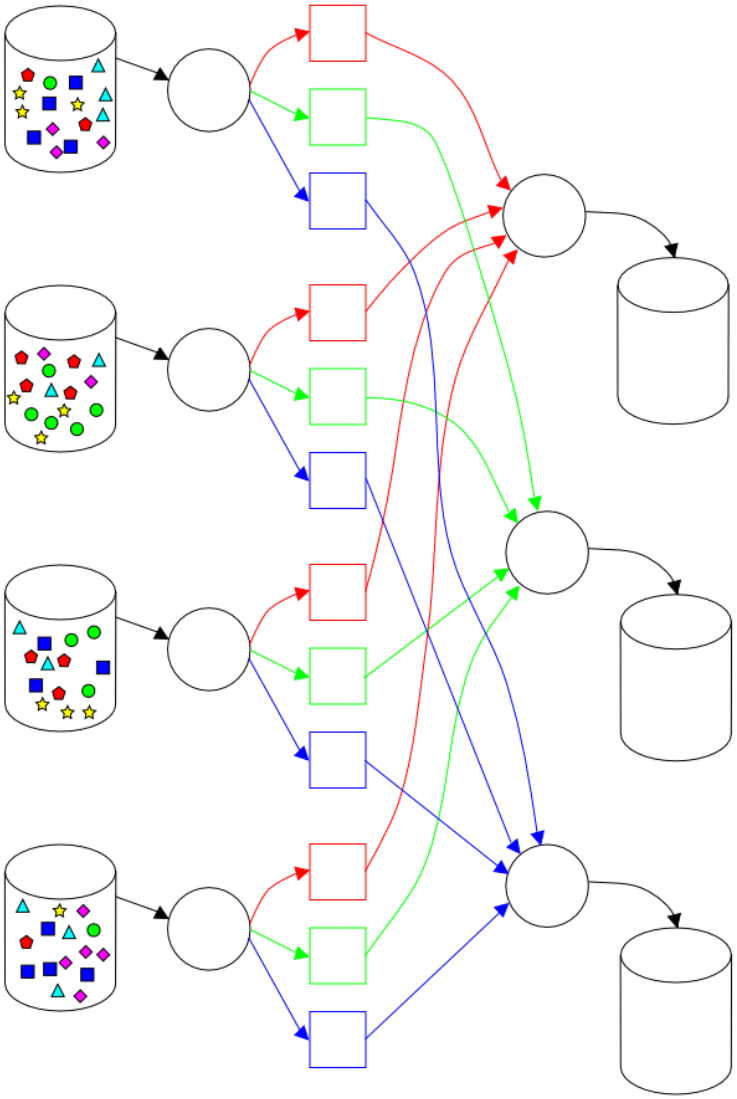


Bližší pohled na provádění MapReduce



(diagram převzat z „Apache Hadoop, Module 4: MapReduce“)

Příklad provádění MapReduce

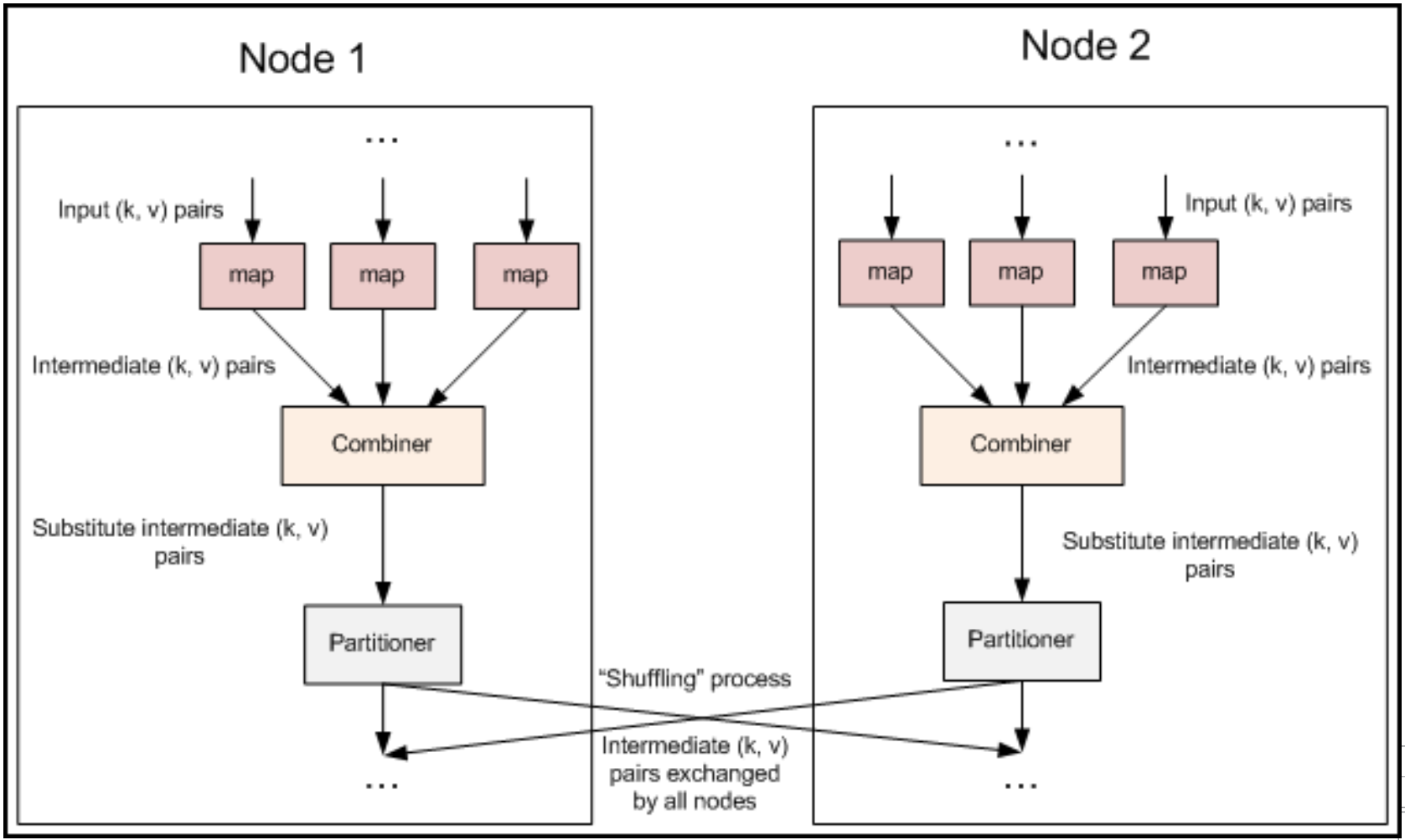


(převzato z „MapReduce Animation, SYSTEMS Deployment, 2012“)

- Map zde přiřadí každému vstupnímu záznamu kategorii.
  (v příkladu vlevo přiřadí vstupnímu grafickému objektu kategorii dle jeho barvy)
- Shuffle seskupí záznamy dle jejich přiřazené kategorie.
  (v příkladu vlevo grafické objekty dle kategorie podle jejich barev)
- Reduce spočítá/uloží záznamy jednotlivých kategorií.
  (v příkladu vlevo grafické objekty patřící do dané kategorie dle barvy)

Rozšíření MapReduce o „Combiner“

„mini-reduce“ proces zpracovávající data generovaná Map procesy v rámci jednoho stroje



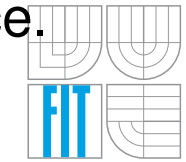
(diagram převzat z „Apache Hadoop, Module 4: MapReduce“)

Distribuovaný souborový systém

- Kromě modelu výpočtu potřebujeme i distribuované uložení dat.
  (MapReduce je model výpočtu, GFS/HDFS distribuovaný souborový systém)
- Google navrhl MapReduce nad Google File System (GFS).
- Při implem. MapReduce v Hadoop byl GFS inspirací pro HDFS.
  (HDFS = Hadoop Distribute File System; dále popisován jen HDFS)
- Distribuovaný souborový systém distribuuje data (a metadata).
  (distribuce skrz IT infrastrukturu, dislokované uzly globálního úložiště)
- Řeší optimální uložení dat, výkon a odolnost vůči výpadkům.
  (různé strategie umístění, např. blízko vniku či spotřeby dat; nutná redundance, ne všechny uzly musí být vždy dostupné či mít poslední verzi dat)

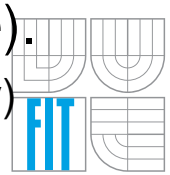
Souborový systém HDFS

- Virtuální distribuovaný souborový systém.
  (vybudován nad běžnými souborovými systémy jednotlivých uzlů, řeší problém nalezení úložiště a přístupu k datům, nikoliv fyzické uložení na uzlu)
- Navržen pro sekvenční přístup k souborů, nikoliv náhodný.
  (MapReduce je dávkové zpracování, čte a zapisuje vst./výst. data sekvenčně)
- Navržen pro velmi velké soubory (BigData).
  (většina režie je pro nalezení úložiště, vlastní čtení/zápis jsou rychlé)
- Data souborů uložena v HDFS v blocích fixní velikosti.
  (typicky 64 či 128 MB, rychlé; implementováno jako skupina bloků lokálních fs. z různých uzlů, tj. lze uložit větší data než je kapacita jednotlivých uzlů; částečně zaplněné HDFS bloky zabírají jen nutný počet bloků lokálních fs., neplýtvá se)
- Jednotlivé HDFS bloky jsou distribuovány a jednotkami replikace.
  (tj. redundance na úrovni HDFS bloků, jeden blok uložen na několika uzlech)

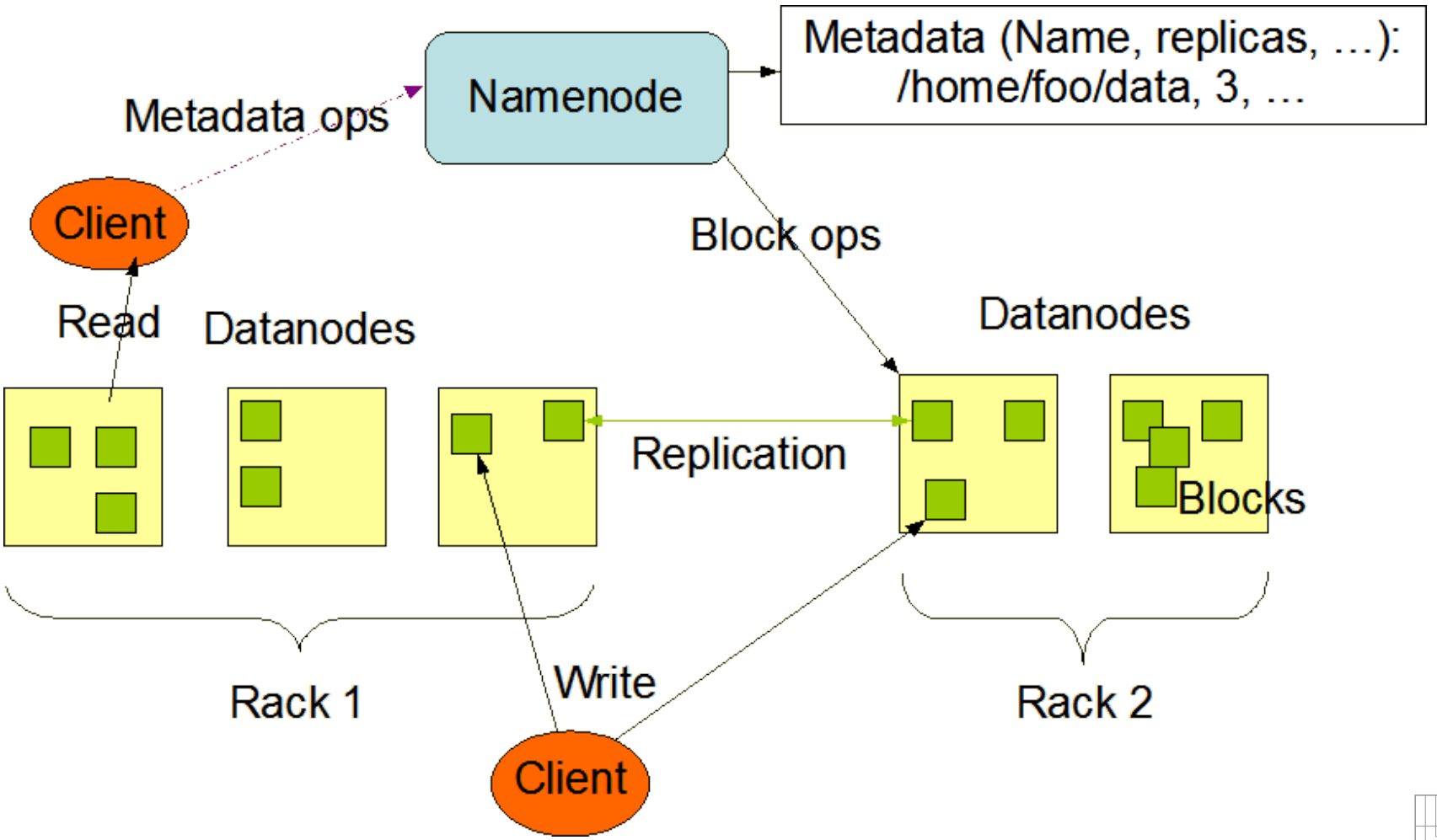


Architektura HDFS

- V HDFS jsou dva typy uzlů
  - **NameNode** spravuje souborový systém a metadata souborů, (adresáře, cesty k souborů, jejich atributy a místa uložení)
  - **DataNode** hostují data, tedy jednotlivé HDFS bloky souborů. (na kterých DataNode jsou které bloky souborů ví NameNode)
- NameNode obvykle jedinný, měl by být výkonný a spolehlivý. (tzv. „single point of failure“, zálohovat na „secondary NameNode“, atp.)
- Více DataNode, díky redundanci nemusí být výkonné a spolehlivé. (stejný HDFS blok je uložen vícekrát na různých DataNode)
- Uzly úložiště jsou fyzicky uspořádaný do „racků“. (rack je v jedné lokalitě/serverovně, jeho uzly jsou lépe/rychleji propojeny)
- NameNode umísťuje instance bloku na různé racky (redundance). (počet instancí záleží na faktoru replikace, obvykle 3 instance na celkem 2 racky)



Architektura HDFS a operace se soubory



(diagram převzat z „HDFS Architecture Guide, Apache.“)

Rámec Apache Hadoop

- Rámec pro distrib., škálovatelné a dávkové výpočty MapReduce.

  Hadoop MapReduce implementace MapReduce paradigma
  Hadoop YARN správa zdrojů a plánování distrib. úloh
  Hadoop DFS distribuovaný souborový systém
  Hadoop Common pomocné knihovny

- Kromě výše uvedených komponent také pomocné nástroje.

  Apache Pig(Latin) hi-level programování MapReduce (Yahoo)
  Apache Hive platforma pro dolování dat nad Hadoop (Facebook)
  Apache HBase distribuovaná databáze nad Hadoop (Google)
  Apache/IBM Jaql dotazovací jazyk pro JSON data
  Apache Flume služba pro řízení toku dat nad Hadoop
  a další ...

  (Mahout, Cassandra, HCatalog, Zookeeper, Oozie, Sqoop, ...)

- Apache Hadoop a většina nástrojů naprogramována v Java.

  (běží nad JVM, v rámci jednoho uzlu může běžet více instancí)

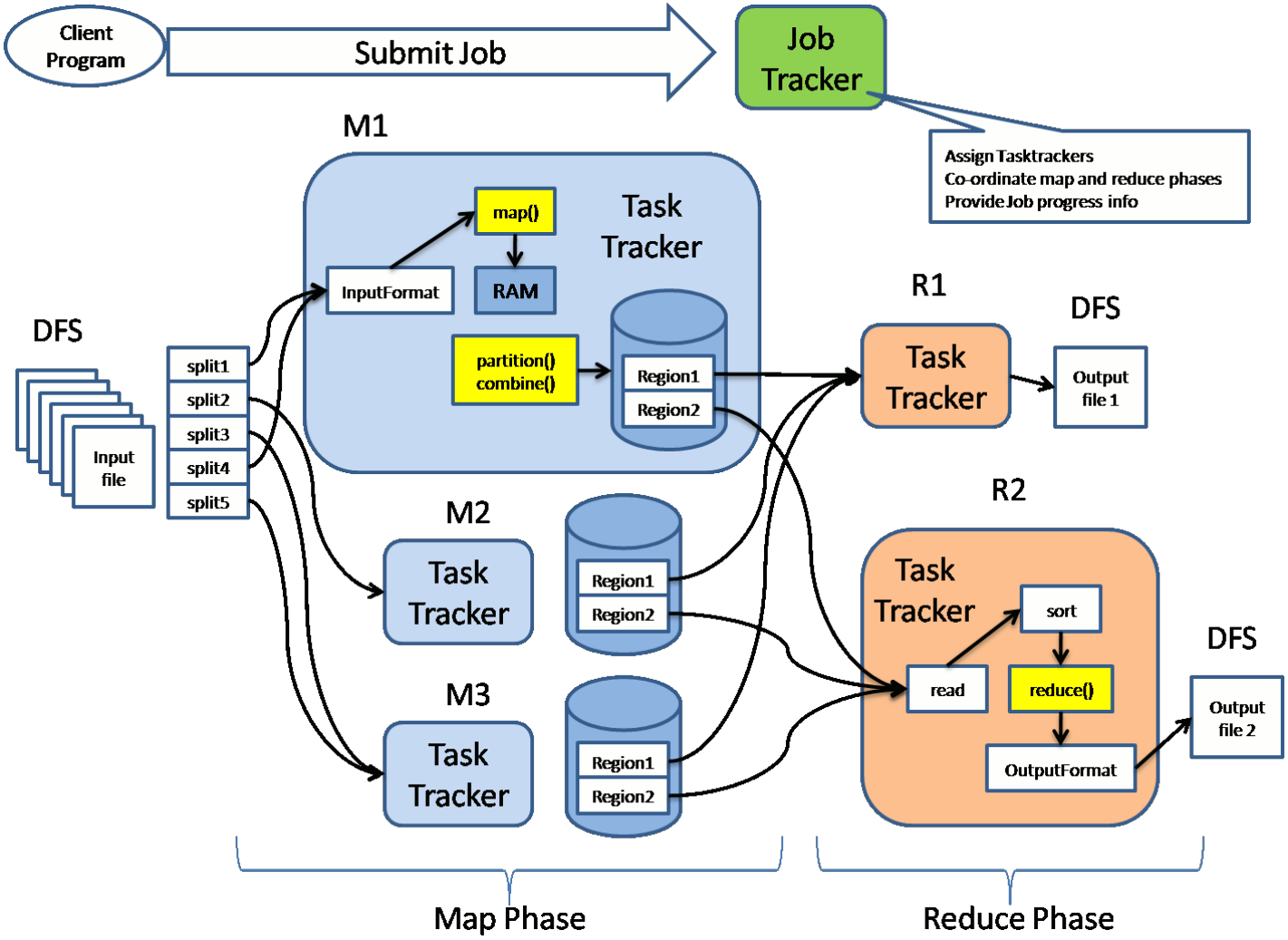
Distribuce Hadoop

- Apache Hadoop
- IBM InfoSphere BigInsights
- MapR M3/M5/M7
- Hortonworks Data Platform
- Intel HPC Distribution for Apache Hadoop
- Cloudera CDH
- EMC Pivotal HD
- DataStax Enterprise
- Microsoft Windows Azure HDInsight

Architektura Hadoop

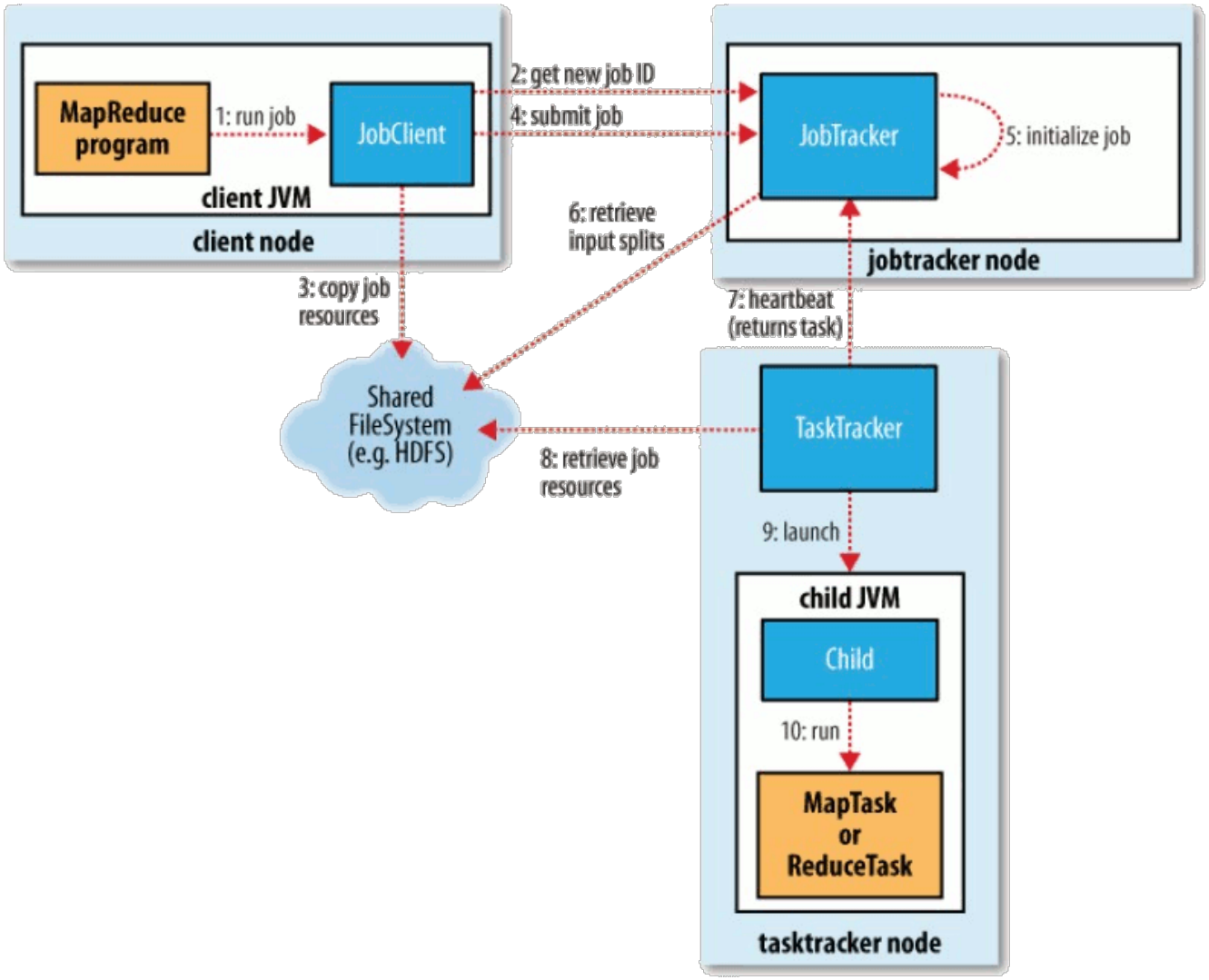
- Kromě HDFS uzlů v Hadoop další dva typy uzlů pro MapReduce
  - **JobTracker** přijímá požadavky na a řídí MapReduce aplikace,
    (pouze jeden na cluster, zadává úkoly a řídí TaskTrackers)
  - **TaskTracker** spouští jednotlivé MapReduce operace.
    (alespoň jeden na uzlu, spouští úlohy v samostatných JVM)
- JobTracker spouští MapReduce aplikace zadané klientem.
  (rozdělí Map a Reduce mezi různé TaskTrackers, hlídá jejich dokončení)
- TaskTrackers dostanou úlohy pracující s jim blízkými daty.
  (nejlépe s daty umístěnými v DataNode na stejném uzlu či racku, jako TT)
- TaskTracker nemusí být spolehlivý, JobTracker musí.
  (přestane-li TaskTracker posílat „heartbeat“, JobTracker jeho úlohy zopakuje)
- TaskTracker spouští každou zadanou úlohu v samostatné JVM.
  (umožňuje mu absolutně kontrolovat její běh a být na něm nezávislý)

Architektura Hadoop a zadávání úloh



(diagram převzat z „How does hadoop MapReduce works, Big Data Foundation.“)

Spuštění MapReduce aplikace nad Hadoop



(diagram převzat z „How MapReduce Works with Hadoop“)

Hadoop API

Abstraktní z MapReduce paradigma:

- $Map(k_1, v_1) \rightarrow list(k_2, v_2)$
- $Reduce(k_2, list(v_2)) \rightarrow list(k_3, v_3)$

Konkrétní z Java balíku „org.apache.hadoop.mapreduce“:

- Interface Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

```
protected void map(KEYIN key,
        VALUEIN value,
        org.apache.hadoop.mapreduce.Mapper.Context context)
        throws IOException, InterruptedException
```

- Interface Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

```
protected void reduce(KEYIN key,
        Iterable<VALUEIN> values,
        org.apache.hadoop.mapreduce.Reducer.Context context)
        throws IOException, InterruptedException
```

- Výstup dvojic (klíč, hodnota) je voláním metody kontextu

```
Context.write(KEYOUT key, VALUEOUT value)
```

Příklad použití Hadoop API – WordCount I

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {
```

Příklad použití Hadoop API – WordCount II

```
public static class Map
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);

    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, one);
    }

  }

}
```

Příklad použití Hadoop API – WordCount III

```
public static class Reduce
  extends Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterable<IntWritable> values, Context context)
   throws IOException, InterruptedException {

    int sum = 0;

    for (IntWritable val : values) {
      sum += val.get();
    }

    context.write(key, new IntWritable(sum));

  }

}
```

Příklad použití Hadoop API – WordCount IV

```
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);

  }

}
```

Řízení Hadoop z příkazového řádku

- Příkazy skriptem „hadoop“ se syntaxí

  `hadoop [-config dir] [COMMAND] [GENERIC_OPTIONS] [COMMAND_OPTIONS]`

- Nejpoužívanější jsou z příkazů (parametr „COMMAND“):
  - „fs“ nebo „dfs“ – práce se soubory na HDFS
    (`hadoop fs [GENERIC_OPTIONS] [COMMAND_OPTIONS]`)
  - „jar“ – spouštění MapReduce aplikací distrib. v *.jar archivech
    (`hadoop jar <jar> [mainClass] args...`)
  - „job“ – práce se běžícími aplikacemi na JobTracker
    (`hadoop job [GENERIC_OPTIONS] [-status <job-id>] | ...`)
  - „dfsadmin“ – administrace HDFS souborového systému
    (`hadoop dfsadmin [GENERIC_OPTIONS] [-report] | ...`)

Pro další příkazy vizte manuál.

Manipulace se soubory na HDFS

V případě `hadoop fs <args...>` se nejčastěji používá

- „-ls“ pro výpis obsahu HDFS adresářů
  (`hadoop fs -ls hdfs://myhost/mypath`)
- „-cat“ pro výpis obsahu HDFS souborů
  (`hadoop fs -cat hdfs://myhost/mypath/myfile`)
- „-mkdir“ pro tvorbu HDFS adresářů
  (`hadoop fs -mkdir hdfs://myhost/mypath/mydir`)
- „-rm“ nebo „-rmr“ pro mazání HDFS souborů/adresářů
  (`hadoop fs -rmr hdfs://myhost/mypath/mydir`)
- „-put“ pro kopii souborů z lokálního systému na HDFS
  (`hadoop fs -put mylocalpath/myfile hdfs://myhost/myfile`)
- „-get“ pro kopii HDFS souborů na lokální systém
  (`hadoop fs -get hdfs://myhost/myfile mylocalpath/myfile`)
- „-getmerge“ pro spojení HDFS souborů do jednoho lokálního
  (`hadoop fs -getmerge hdfs://myhost/myf1 hdfs://myhost/myf2 myfile`)

Pro další příkazy vizte manuál.

Spuštění MapReduce aplikací na Hadoop

- Nejjednodušší je zabalit aplikaci do JAR archivu a spustit.
  1. `javac -cp <hadoop-*-core.jar> -d <class-files-dir> <java-files>`
  2. `jar -cvf <myapp.jar> -C <class-files-dir> .`
  3. `hadoop jar <myapp.jar> [mainClass] args...`
- Před spuštění je nutno do HDFS nakopírovat potřebná data.

Příklad:

```
hadoop fs -put file01 hdfs:/localhost/usr/joe/input/
hadoop fs -put file02 hdfs:/localhost/usr/joe/input/
hadoop jar wordcount.jar org.myorg.WordCount /usr/joe/input /usr/joe/output
hadoop fs -cat /usr/joe/output/part-00000
```

Shrnutí a závěr

- MapReduce paradigma pro paralelní, distribuované, dávkové výpočty.
  (funkce Map a Reduce; doplněk k OLTP/OLAP, nikoliv náhrada)
- Hadoop je rámec pro běh MapReduce aplikací.
  (NameNode a DataNode pro HDFS, JobTracker a TaskTracker pro MapReduce)
- Programátorovi stačí implementovat Map a Reduce fce.
  (případně navíc org.apache.hadoop.mapred.Partitioner)