MapReduce

张坤龙
zhangkl@tju.edu.cn

2007-12-05

内容

- 问题 - MapReduce要解决什么问题？
- 理论 - MapReduce的理论基础
- 模型 – MapReduce的编程模型
- 实现 - MapReduce的实现和评测
- 未来 - MapReduce的未来发展趋势

处理海量数据

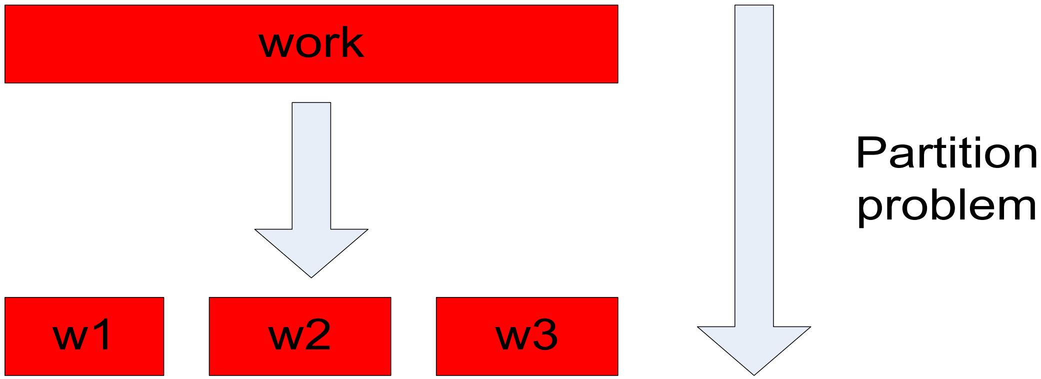
如何统计Google收集的网页中各个单词出现的次数？

Goolge收集的网页占用存储空间超过400TB，假设一台计算机以30MB/sec的速度从磁盘读取数据，那么所需时间将超过4个月！

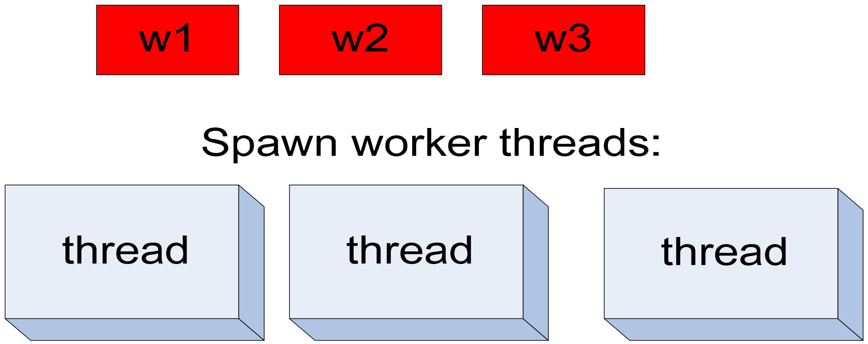
Google Cluster

- 采用并行计算技术，可以将时间缩短到3个小时以下。

并行化（1）

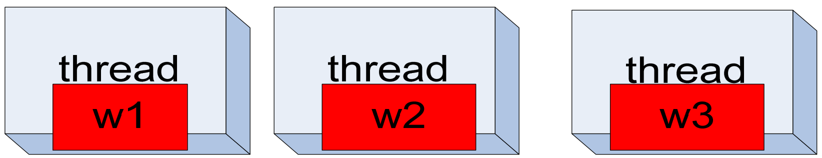


并行化（2）

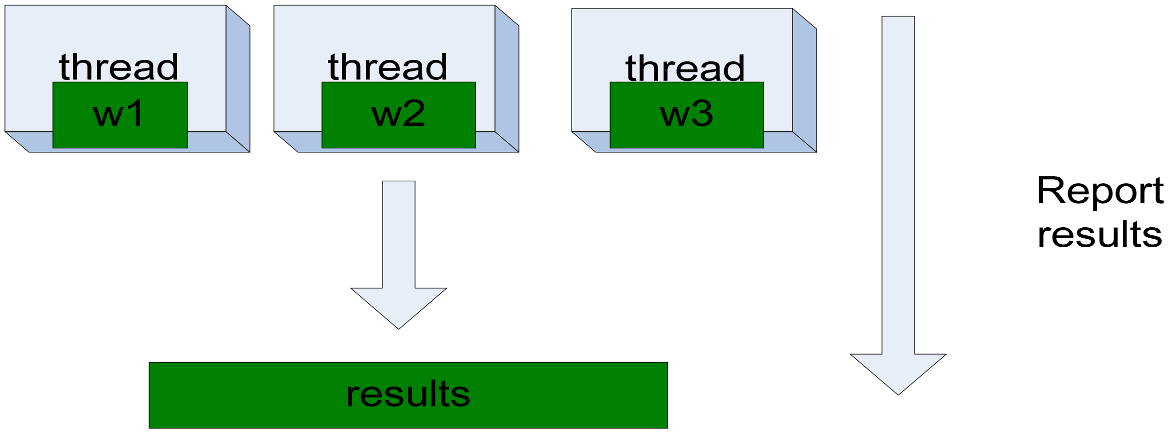


并行化（3）

Workers process data:



并行化（4）



并行化时要考虑的问题

- 如何划分工作？
- 工作之间需要同步吗？
- 各线程的工作量均衡吗？
- 如何将工作指派给线程？
- 如何处理故障？
- 如何知道所有的工作都已经完成？
- 最后阶段如何汇总结果？
- ……

简单的任务
复杂的实现

小结

- 简单的计算任务
  - 单词计数、Grep、倒排索引、排序、……
- 海量的输入数据
  - 整个互联网，网页数目至少是百亿级
- 集群计算环境
  - 超过一万个结点
- 如何充分利用硬件，简化程序设计？

函数式程序设计的特点(1)

- 不修改数据

```
int x = 5;
x = x + 1;
```

函数式程序设计的特点(2)

- 运算次序无关紧要

```
fun foo(lst: int list) =
  sum(lst) + mul(lst) + length(lst)
```

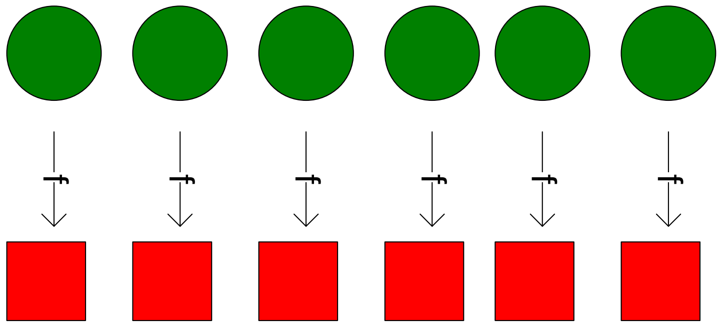
函数式程序设计的特点(3)

- 函数可以做参数

```
fun DoDouble(f, x) = f (f x)
```

Map

```
fun map f []        = []
  | map f (x::xs) = (f x) :: (map f xs)
```

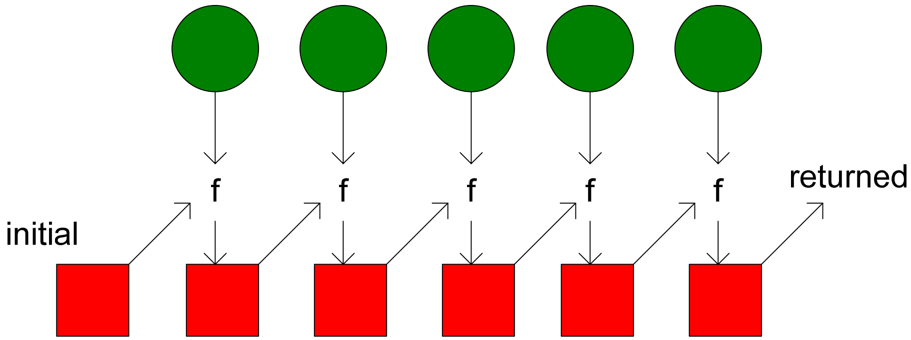


map sqrt [1,4,9,16]

Fold

```
fun foldl f a []       = a
  | foldl f a (x::xs) = foldl f (f(x, a)) xs

fun foldr f a []       = a
  | foldr f a (x::xs) = f(x, (foldr f a xs))
```



foldl (-) 1 [4,8,5], foldr (-) 1 [4,8,5]

举例

```
fun foo(lst: int list) =
  sum(lst) + mul(lst) + length(lst)

fun sum(lst) = foldl (fn (x,a)=>x+a) 0 lst
fun mul(lst) = foldl (fn (x,a)=>x*a) 1 lst
fun length(lst) = foldl (fn (x,a)=>1+a) 0 lst
```

Map的并行化

```
map f [] = []
map f (x:xs) = f x : map xs
```

在什么条件下可以并行化map?

– 计算是独立的，各个元素上的计算互不影响

– 计算次序不需要从左到右，结果输出顺序任意

Fold的并行化

```
foldl f z [] = z
foldl f z (x:xs) = foldl f (f z x) xs
```

在什么条件下可以并行化fold?

– 不可以并行化fold

MapReduce

```
mapreduce f_m f_r lst =
    map (reducePerKey f_r) (group (map f_m lst))

reducePerKey f_r (k,v_list) =
    (k, (foldl (f_r k) [] v_list))
```

MapReduce maps a fold over the result of a map!

小结

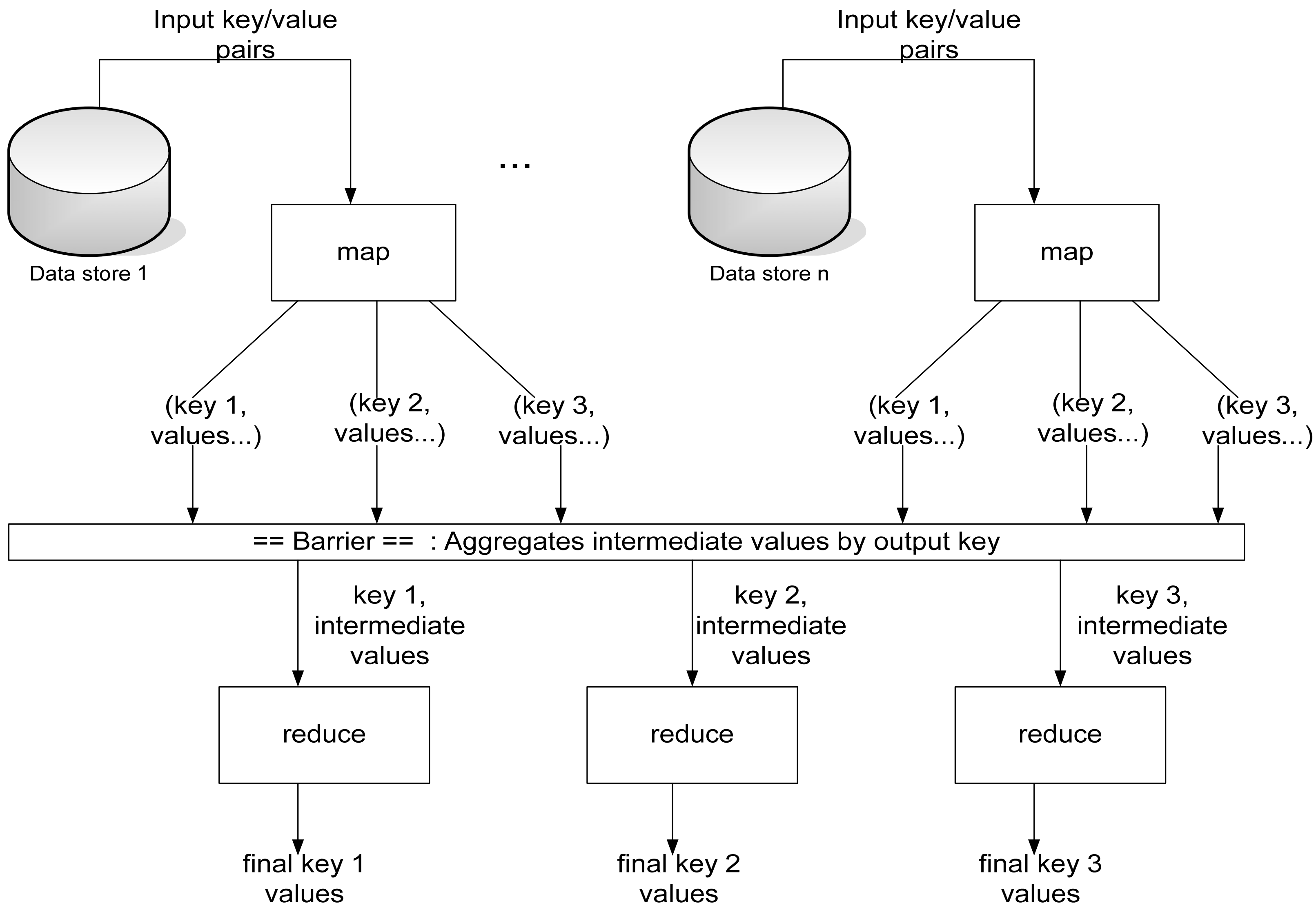
- MapReduce借鉴了函数式程序设计语言的设计思想
  - MapReduce is inspired by the map and reduce primitives present in Lisp and many other functional languages.
- Lämmel对MapReduce的理论基础作了更深入地探讨
  - R. Lämmel. Google's MapReduce Programming Model – Revisited. http://www.cs.vu.nl/~ralf/MapReduce/.

程序设计模型

- 用户定义两个函数

```
□ map  (in_key, in_value) ->
   (out_key, intermediate_value) list

□ reduce (out_key, intermediate_value list) ->
   out_value list
```



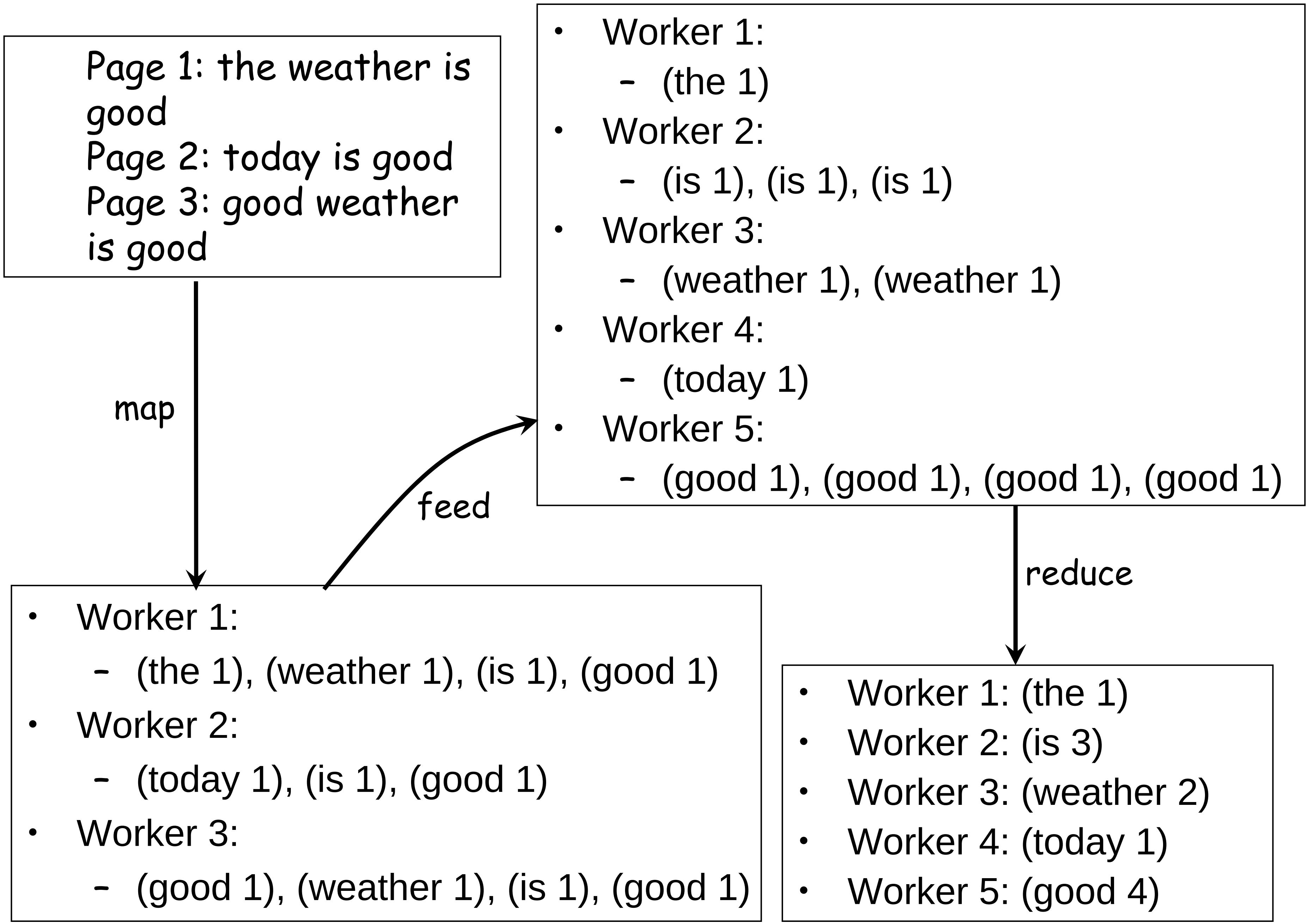
Input key/value pairs
Input key/value pairs
Data store 1
map
...
Data store n
map
(key 1, values...)
(key 2, values...)
(key 3, values...)
(key 1, values...)
(key 2, values...)
(key 3, values...)
== Barrier == : Aggregates intermediate values by output key
key 1, intermediate values
key 2, intermediate values
key 3, intermediate values
reduce
reduce
reduce
final key 1 values
final key 2 values
final key 3 values

举例：单词计数

- 统计多个文档中每个单词出现的次数
  - Page 1: the weather is good
  - Page 2: today is good
  - Page 3: good weather is good

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

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


Page 1: the weather is good
Page 2: today is good
Page 3: good weather is good
map
Worker 1:
(the 1), (weather 1), (is 1), (good 1)
Worker 2:
(today 1), (is 1), (good 1)
Worker 3:
(good 1), (weather 1), (is 1), (good 1)
feed
Worker 1:
(the 1)
Worker 2:
(is 1), (is 1), (is 1)
Worker 3:
(weather 1), (weather 1)
Worker 4:
(today 1)
Worker 5:
(good 1), (good 1), (good 1), (good 1)
reduce
Worker 1: (the 1)
Worker 2: (is 3)
Worker 3: (weather 2)
Worker 4: (today 1)
Worker 5: (good 4)

实际的代码

```
#include "mapreduce/mapreduce.h“

// User's map function
class WordCounter : public Mapper {
 public:
  virtual void Map(const MapInput& input) {
    const string& text = input.value();
    const int n = text.size();
    for (int i = 0; i < n; ) {
        // Skip past leading whitespace
        while ((i < n) && isspace(text[i])) i++;
        // Find word end
        int start = i;
        while ((i < n) && !isspace(text[i])) i++;
        if (start < i) Emit(text.substr(start,i-start),"1");
    }
  }
};
REGISTER_MAPPER(WordCounter);
```

```
// User's reduce function
class Adder : public Reducer {
   virtual void Reduce(ReduceInput* input) {
      // Iterate over all entries with the
      // same key and add the values
      int64 value = 0;
      while (!input->done()) {
        value += StringToInt(input->value());
        input->NextValue();
      }
      // Emit sum for input->key()
      Emit(IntToString(value));
   }
};
REGISTER_REDUCER(Adder);
```

```
int main(int argc, char** argv) {
    ParseCommandLineFlags(argc, argv);
    MapReduceSpecification spec;
    // Store list of input files into "spec"
    for (int i = 1; i < argc; i++) {
        MapReduceInput* input = spec.add_input();
        input->set_format("text");
        input->set_filepattern(argv[i]);
        input->set_mapper_class("WordCounter");
    }
    // Specify the output files:
    MapReduceOutput* out = spec.output();
    out->set_filebase("/gfs/test/freq");
    out->set_num_tasks(100);
    out->set_format("text");
    out->set_reducer_class("Adder");
    // Optional: do partial sums within map tasks
    out->set_combiner_class("Adder");
    // Tuning parameters
    spec.set_machines(2000);
    spec.set_map_megabytes(100);
    spec.set_reduce_megabytes(100);
    // Now run it
    MapReduceResult result;
    if (!MapReduce(spec, &result)) abort();
    // Done: 'result' structure contains info about counters, time
    // taken, number of machines used, etc.
    return 0;
}
```

小结

- 简单的程序设计模型
- 并行化、容错、数据分布、负载均衡等工作均由系统来实现
- 一个3800行的C++程序重写后只需要700行。

运行环境

- 硬件
  - 2-CPU x86 machines，2-4 GB of memory
  - 100 mbps or 1 gbps Ethernet
  - Storage is on local IDE disks
  - Clusters consists of thousands of machines

- 软件
  - GFS: distributed file system manages data
  - Job scheduling system: jobs made up of tasks, scheduler assigns tasks to machines

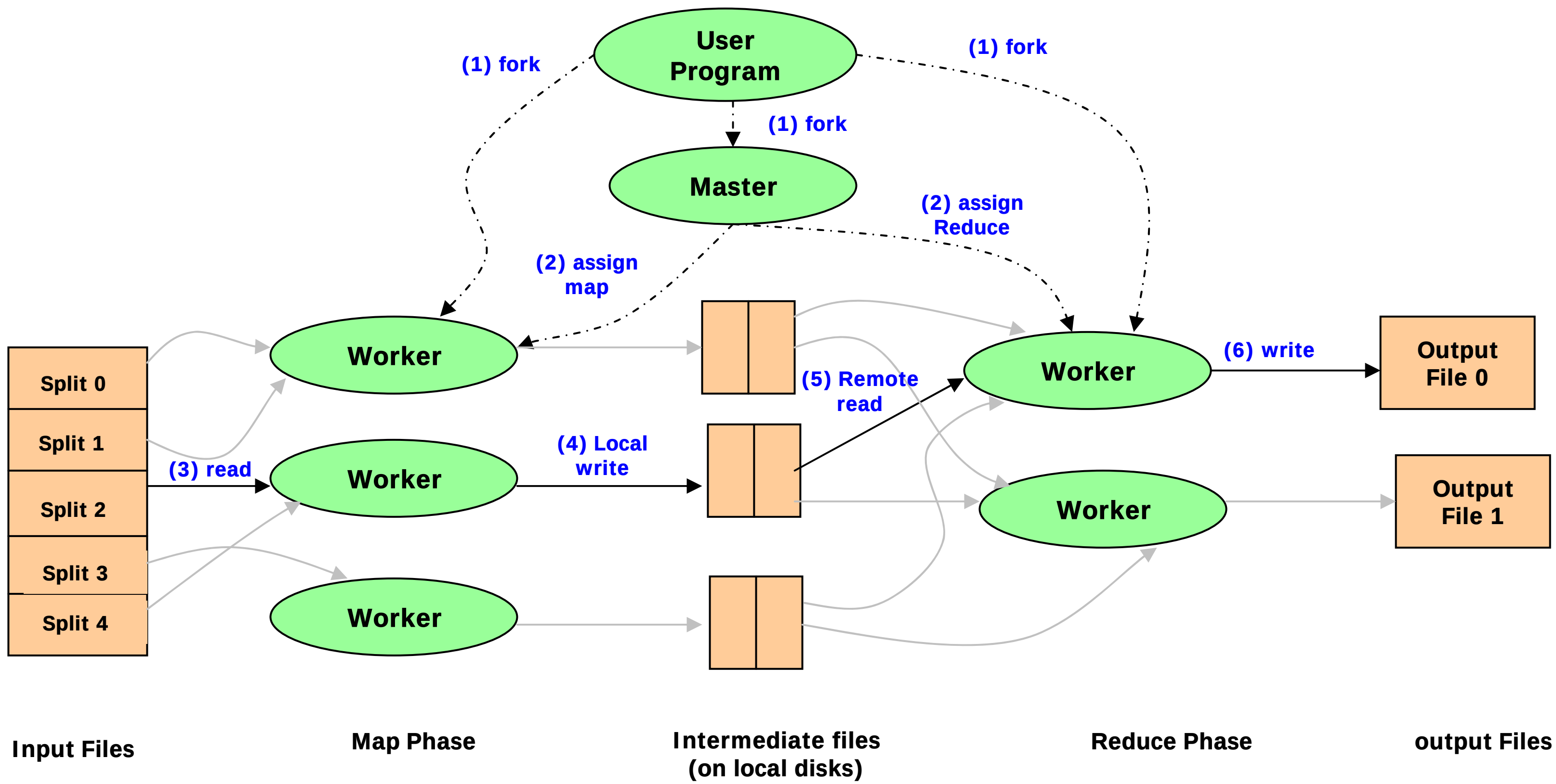
并行化

- Map
  - 将输入划分为M个块，
  - 每块的大小为16-64MB

- Reduce
  - 将中间键值划分为R块
  - **`hash(intermediate_key) mod R`**

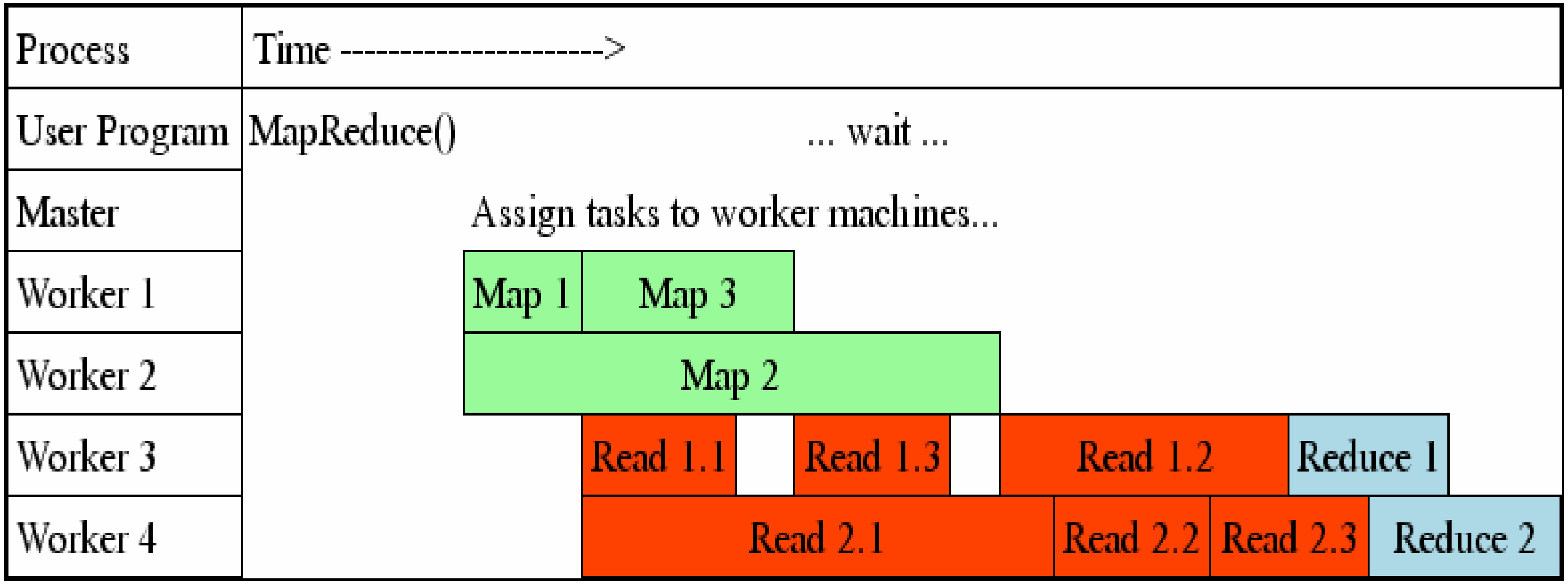
- 典型配置
  - 2000个机器，M=200000，R=5000

运行过程



User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign Reduce
(2) assign map
Worker
Worker
Worker
Worker
Worker
Split 0
Split 1
Split 2
Split 3
Split 4
(3) read
(4) Local write
(5) Remote read
(6) write
Output File 0
Output File 1
Input Files
Map Phase
Intermediate files (on local disks)
Reduce Phase
output Files

执行次序


Process
Time ---------------------->
User Program
MapReduce()
... wait ...
Master
Assign tasks to worker machines...
Worker 1
Map 1
Map 3
Worker 2
Map 2
Worker 3
Read 1.1
Read 1.3
Read 1.2
Reduce 1
Worker 4
Read 2.1
Read 2.2
Read 2.3
Reduce 2

实现细节

- 容错
  - Master定期探测worker。遇到故障时，对于map，无论是否完成都要重新执行，对于reduce，则只在未完成时重新执行。
- 本地化
  - 尽量将map就近其输入数据所在地执行。
- 任务粒度
  - 偏好细粒度，M和R要大于worker的数目以利于负载均衡和故障恢复。
- 后备任务
  - MapReduce将要完成时，再一次执行尚未完成的任务，先完成者获胜。

改进

- 划分函数
  - Map的输出被分块，划分函数可以定制
- 有序
  - Map的输出被分块，块内按中间键值排序
- Combiner函数
  - 与map在同一个机器上执行，做map输出数据的reduce
- 其他改进……

性能测试

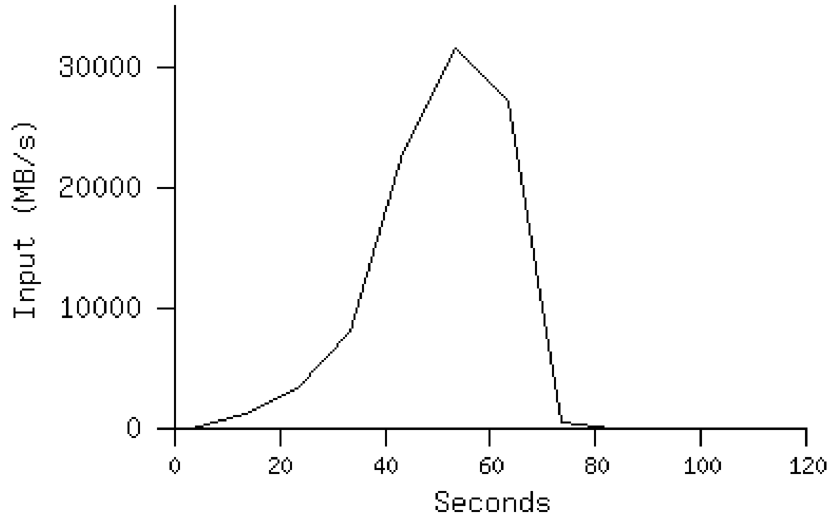
- 1800个机器的集群
  - 4 GB内存
  - Dual-processor 2 GHz Xeons with Hyperthreading
  - Dual 160 GB IDE disks
  - Gigabit Ethernet per machine

- 两个基准测试程序

MR_GrepScan　10^{10}个100字节的记录，从中找出符合特定模式的记录(92K个)

MR_SortSort　10^{10}个100字节的记录按照 TeraSort基准测试程序的方式排序

MR_Grep



在150秒之内处理1TB数据

本地化的作用

- 1800个机器读1TB数据的峰速为大约31GB/s
- 如果不做本地化，只能达到网络的10GB/s限制

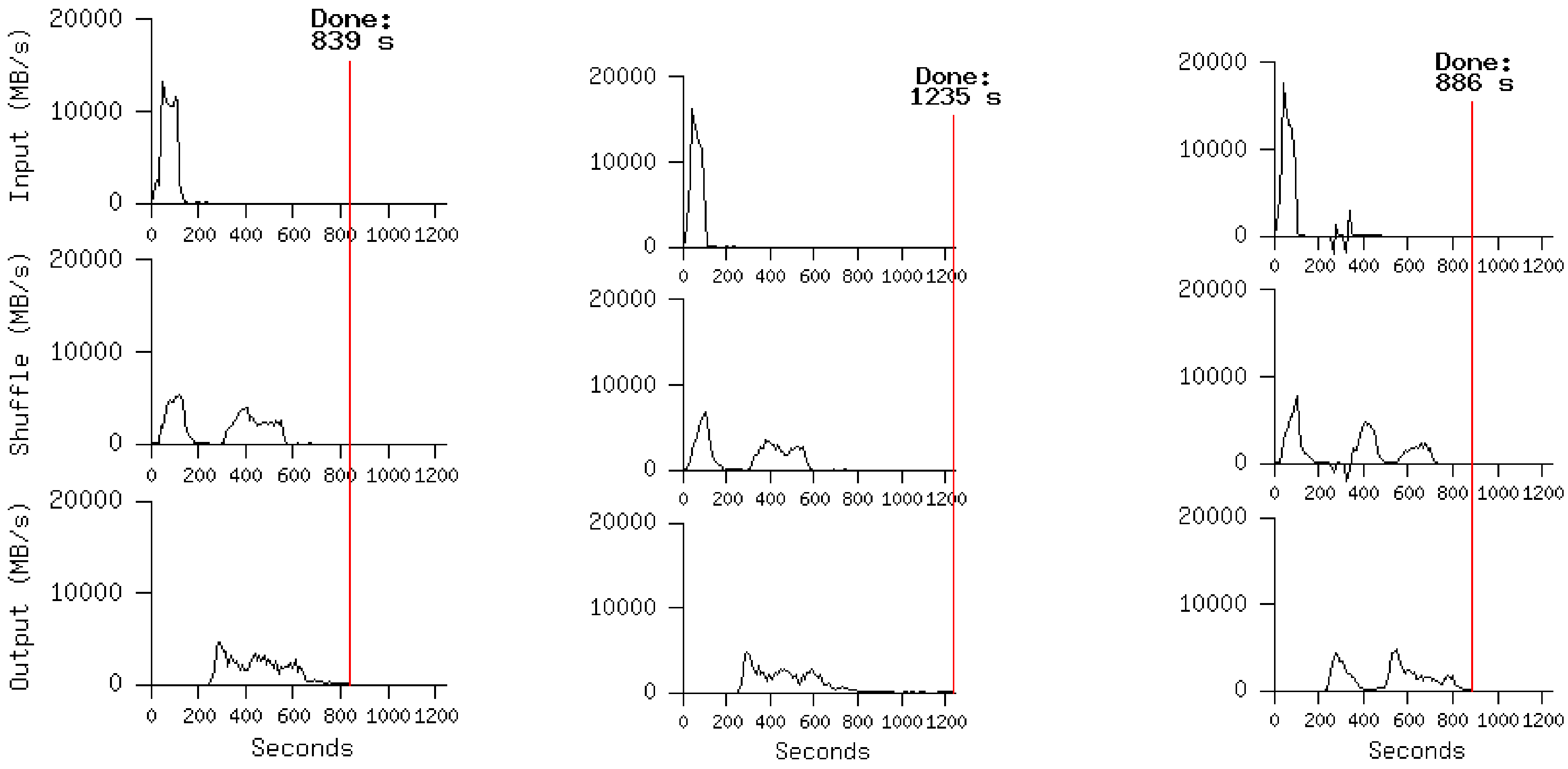
启动的额外开销很大

- 计算总共花了150秒，其中有1分钟的启动时间

MR_Sort

在14分钟内完成了对1TB记录的排序

Normal **No backup tasks** **200 processes killed**



后备任务减少了执行时间　　系统的容错性好

MapReduce的使用情况

Usage Statistics Over Time

	Aug, '04	Mar, '05	Mar, '06
Number of jobs	29,423	72,229	171,834
Average completion time (secs)	634	934	874
Machine years used	217	981	2,002
Input data read (TB)	3,288	12,571	52,254
Intermediate data (TB)	758	2,756	6,743
Output data written (TB)	193	941	2,970
Average worker machines	157	232	268
Average worker deaths per job	1.2	1.9	5.0
Average map tasks per job	3,351	3,097	3,836
Average reduce tasks per job	55	144	147
Unique map/reduce combinations	426	411	2345

小结

- 高效的实现使得MapReduce已经被用于多项任务
  - distributed grep，distributed sort，term-vector per host，document clustering，machine learning，web access log stats，web link-graph reversal，inverted index construction，statistical machine translation
- 编程模型和实现是可以分离的。
  - Hadoop：http://lucene.apache.org/hadoop/

The Landscape of Parallel Computing Research: A View from Berkeley

Krste Asanovic
Ras Bodik
Bryan Christopher Catanzaro
Joseph James Gebis
Parry Husbands
Kurt Keutzer
David A. Patterson
William Lester Plishker
John Shalf
Samuel Webb Williams
Katherine A. Yelick

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-183
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

December 18, 2006

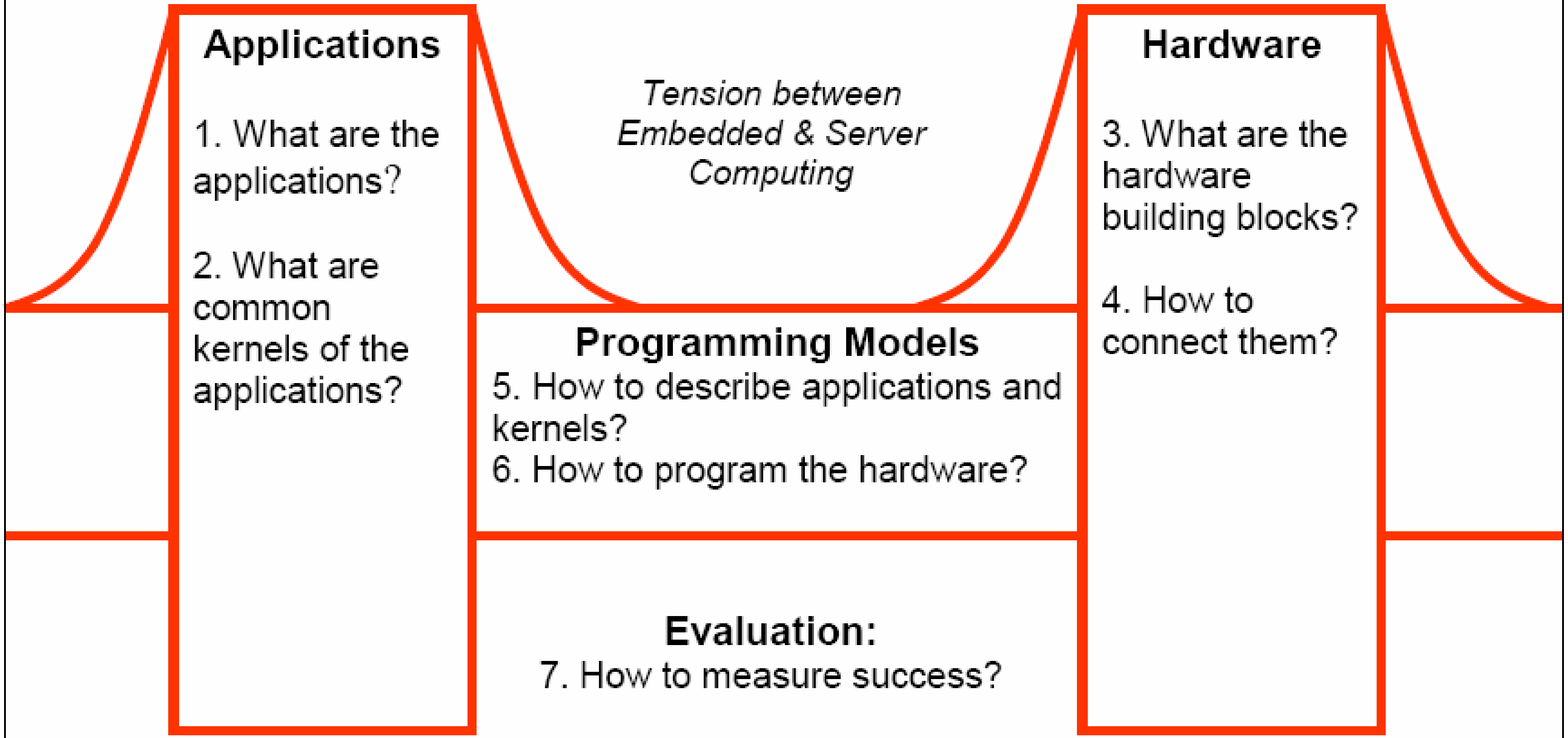
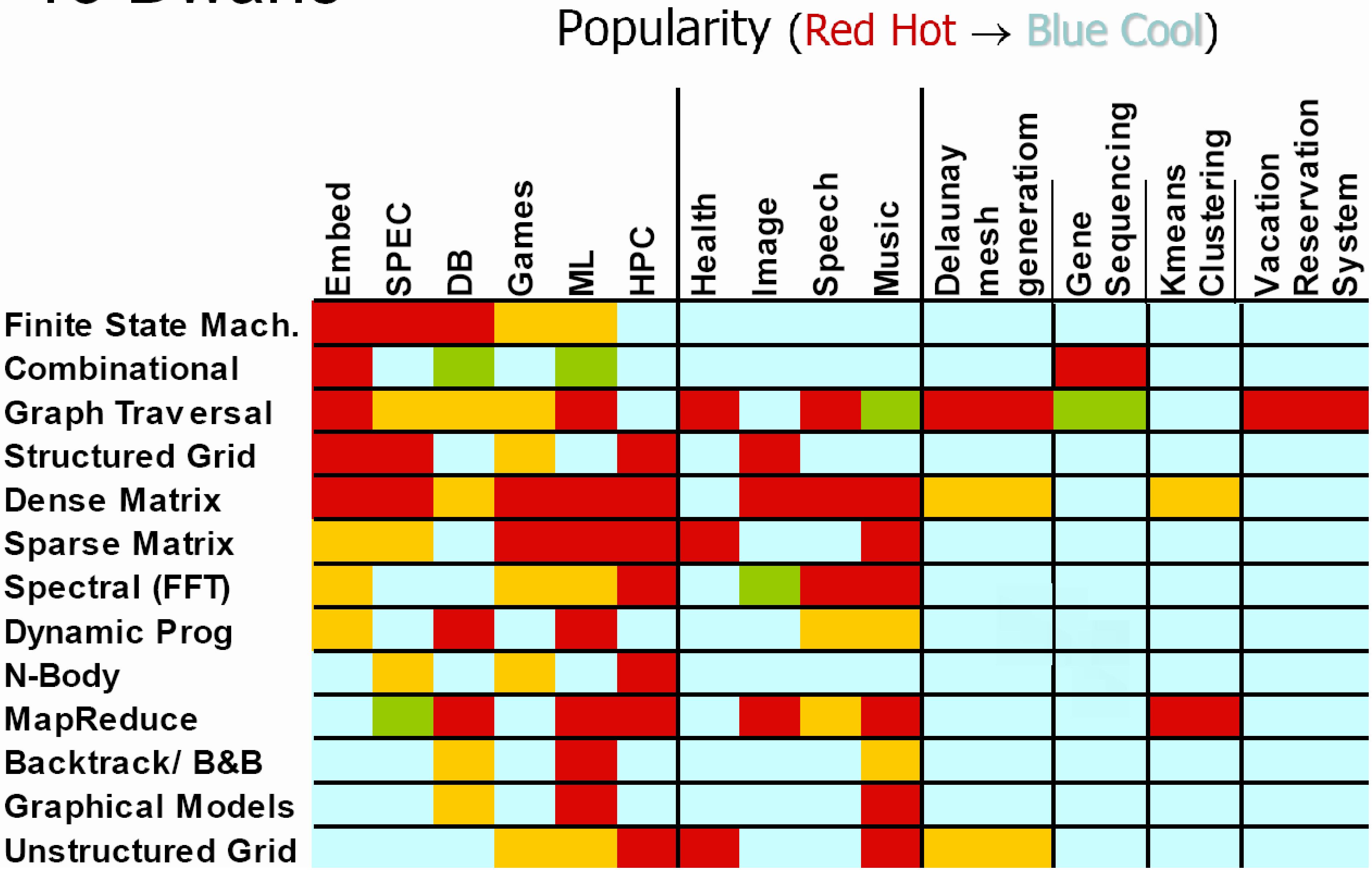


Figure 1. A view from Berkeley: seven critical questions for 21st Century parallel computing. (This figure is inspired by a view of the Golden Gate Bridge from Berkeley.)

13 Dwarfs


Popularity (Red Hot → Blue Cool)
Embed
SPEC
DB
Games
ML
HPC
Health
Image
Speech
Music
Delaunay mesh generatiom
Gene Sequencing
Kmeans Clustering
Vacation Reservation System
Finite State Mach.
Combinational
Graph Traversal
Structured Grid
Dense Matrix
Sparse Matrix
Spectral (FFT)
Dynamic Prog
N-Body
MapReduce
Backtrack/ B&B
Graphical Models
Unstructured Grid

小结

- 新的程序设计模型
  - 学习函数式程序设计语言获取灵感
- 新的MapReduce实现方式
  - 不同的执行环境，如多核
- 更多的MapReduce应用
  - 例如数据库中的OLAP、图像处理

总结

MapReduce是一个

易于使用的

处理海量数据的

并行程序设计模型

参考资料

- J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, OSDI 2004.
- K. Asanovic et. al., “The Landscape of Parallel Computing Research: A View from Berkeley”, Technical Report No. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.
- Ranger et. al., "Evaluating MapReduce for Multi-core and Multiprocessor Systems", HPCA 2007.
- M. Kruijf and K. Sankaralingam, “MapReduce for the Cell B.E. Architecture”, Technical Report No. TR1625, Computer Science Department, University of Wisconsin, Madison, 2007.

注意

本讲义中，有很多内容来自于其他人做的相关讲义，这些内容并未一一注明出处！