



MapReduce

Ralf Schenkel

Web-Scale Computation

Many problems cannot be easily scaled to the Web
(about 20TB per Google crawl)

- Document inversion
- PageRank etc. computation
- Web log mining
- Host statistics
 - Term distribution per host
 - Accesses per host

Motivation

Precomputation on 20TB of data?

Easy, we have ~~paris: 64 cores, 192 GB RAM~~

titan: 16 cores, 256 GB RAM

25,000€ \Rightarrow 1625.50 €/core, 97.66 €/GB

Multimedia PC
ALDI MEDION akoya E4320 D

Intel® Core™2 Duo Prozessor E7300
2,66 GHz, 3 MB L2 Cache, 1066 MHz FSB

Speicherkapazität erweiterbar!
Der MEDION® Datenhafen 2.

inkl. Software-Paket und Zubehör!
Umfangreiche Anschlussmöglichkeiten!

3 Jahre Garantie

Mit Service-Hotline,
365 Tage im Jahr.

je **499,-***

Intel, das Intel Logo, Intel Core und Intel Core Inside sind Marken der Intel Corporation oder ihrer Tochtergesellschaften in den USA oder anderen Ländern.

- gigantische 640 GB Festplatte
- 3 GB Arbeitsspeicher 3072 MB DDR2 SDRAM
- Multiformat DVD/CD Brenner
- kristallklare DirectX®10 Grafik NVIDIA® GEFORCE® 9500 GS Grafikkarte
- Netzwerk Controller Ethernet LAN 10/100 Mbit/s
- eSATA-Anschluss
- 8 Kanal High Definition Audio

ALDI Süd, 29.09.2008:
2 cores, 3GB RAM, 499€
 \Rightarrow 299.50 €/core,
166.33 €/GB

Large Clusters of Commodity Hardware

- Thousands of off-the-shelf networked PCs
- Hardware failures (of single machines) common
- Harddrive failures common

- Distributed Programs to exploit full power (RPC, CORBA, MPI, WebServices, REST, ...?)

MapReduce Features

- Complete solution for distributed computing
- **Simple**, but powerful **interface**
- Implementation within **hours**, not weeks
- **Detects** machine failures and **redistributes** work
- **Avoids** data loss due to harddisk failures (together with distributed file system)

**Widely used at Google for daily business
(2 mio MapReduce jobs in Sep 07 on 15TB each, 400s each)**

MapReduce by Example

Problem: Compute document frequencies

- Input: data with keys (docs with docids/urls)
- Output: aggregated data (terms with counts)

Solved by two functions (provided by user):

- **MAP**: partition input data by output key (term)
- **REDUCE**: aggregate data for each output key

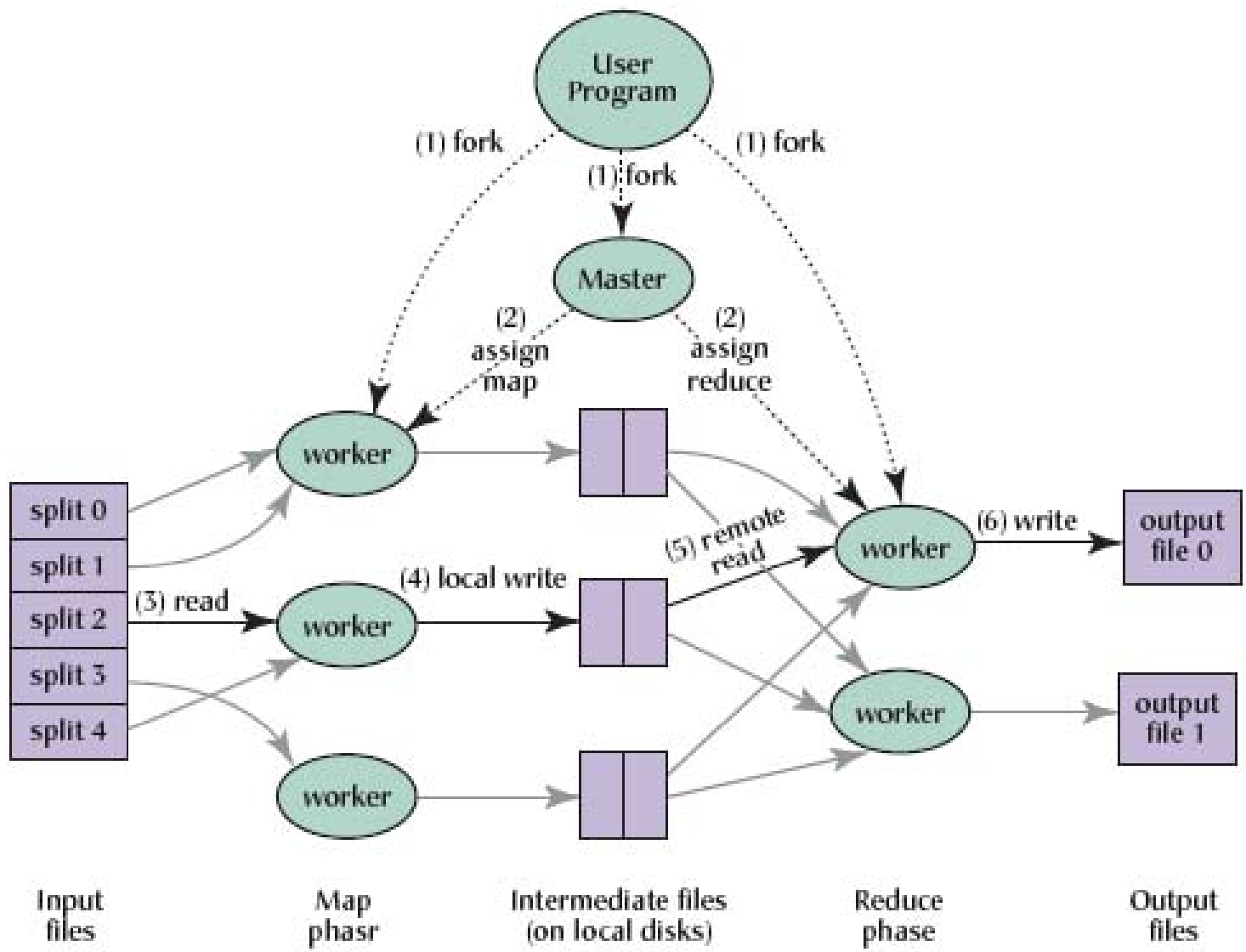
Automatically executed in a distributed fashion

MapReduce by Example: Compute DF

```
map(String key, String value)
// key: document name
// value: document content
for each term in value:
    EmitIntermediate(term,1);
```

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

Architecture



taken from [Dean et al., CACM 51(1), 2008]

Architecture

- Dedicated **master process** identifies worker processes/machines for map and reduce
- Master **partitions** input file into M partitions
- Partitions **assigned** to map workers
- Map workers **output to R files** on local hard disks (by hash code), master notified
- Each reduce worker **reads** one output file from the map workers for each input partition (by RPC) & **sorts** them (many output keys per file!)
- Each reduce worker **aggregates** data per key

Failure Handling

- Master monitors workers
- On worker failure:
 - All MAP tasks marked failed and submitted to other workers (including finished ones – data on local hard disk!)
 - All active REDUCE tasks resubmitted to other workers
 - Requires idempotence of operations (workers could just be slow, not failed)

Application Example: PageRank

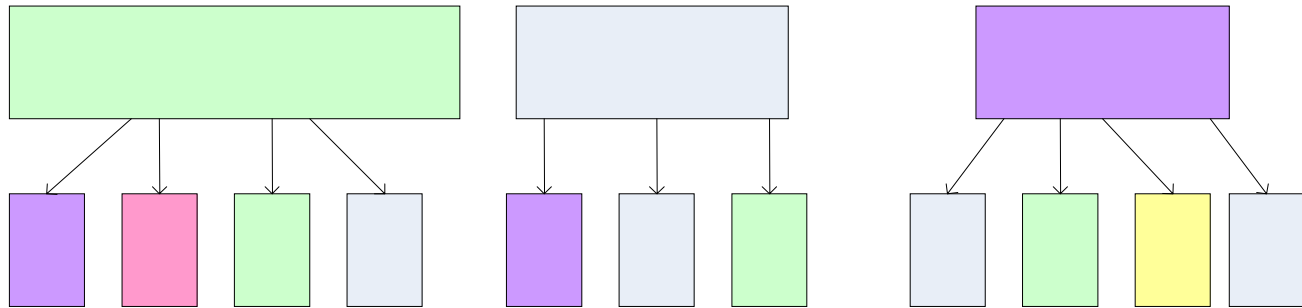
- Definition of PageRank

$$PR(v) = \varepsilon \sum_{(u,v) \in E} \frac{PR(u)}{\text{outdeg}(u)} + (1 - \varepsilon)$$

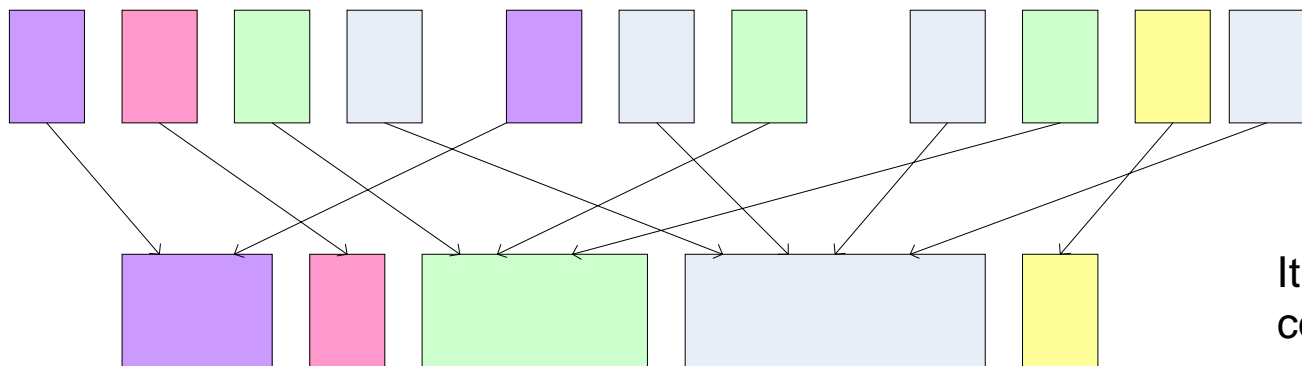
- Computed through power iteration:
values in step i computed from values in step $i-1$
and graph structure
- Highly local computation: requires only old pageranks from incident nodes

PageRank in MapReduce

Map: distribute PageRank “credit” to link targets



Reduce: gather up PageRank “credit” from multiple sources to compute new PageRank value



Iterate until convergence

[Picture probably courtesy of Jimmy Lin or Christophe Bisciglia et al.]

Initial Step

MAP:

(url, content)



(url, (initial pagerank, list(linked urls)))

REDUCE:

Passes input tuples to output without change

Iteration Steps

MAP:

(url, (PR, list(n linked urls)))



(linked url 1, PR/n), ..., (linked url n, PR/n),
(url, list(n linked urls))

REDUCE:

(url, PR1), ..., (url, PRx) , (url, list(linked urls))



(url, (PR', list(linked urls)))

Termination

Terminate when values are stable
(determined by central component)

Implementations freely available

- PIG (Yahoo)

<http://research.yahoo.com/node/90>

- Hadoop (Apache)

<http://hadoop.apache.org/>

- DryadLinq (Microsoft)

<http://research.microsoft.com/research/sv/DryadLINQ/>

Ongoing Work

- Extensions to different settings (multicore CPUs, graphic hardware)
- Extensions to more complex, DB-style applications
- Extensions with a more powerful language (“pig latin”)

Pig Latin vs. SQL

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 106
```

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)>106;
output = FOREACH big_groups GENERATE
category, AVG(good_urls.pagerank);
```

Summary

- MapReduce is a powerful framework for distributed computing
- Exploits potential of commodity hardware
- Hadoopify your applications!
- But: Does not solve everything

References

- [Dean04] J. Dean et al. MapReduce: Simplified Data Processing on Large Clusters. *OSDI* 2004.
- [DeWitt08] D. DeWitt et al. MapReduce: A major step backwards. *The Database Column*, January 17, 2008
<http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html>
- [He08] B. He et al. Mars: A MapReduce Framework on Graphics Processors. *PACT* 2008.
- [Olston08] C. Olston et al. Pig Latin: A Not-So-Foreign Language for Data Processing. *SIGMOD* 2008.
- [Ranger07] C. Ranger. et al. Evaluating MapReduce for Multi-core and Multiprocessor Systems. *HPCA* 2007.
- [Witten99] I. Witten et al. *Managing Gigabytes*. Morgan Kaufman, 1999.
- [Yang07] H. Yang et a. Map-reduce-merge: simplified relational data processing on large clusters. *SIGMOD* 2007.