Distributed Data Management

Christoph Lofi
Institut für Informationssysteme
Technische Universität Braunschweig
http://www.ifis.cs.tu-bs.de
12 Yahoo and Map & Reduce

12.0 Data Storage at Yahoo
12.1 Sherpa/PNUTS
12.2 Map & Reduce
• Remember Lecture 10??
  – Have a look here:
    • http://aws.amazon.com/dynamodb/
    • http://www.youtube.com/watch?feature=player_embedded&v=oz-7wJJ9HZ0

• Free NoSQL Developers Refcard
  – Issue 105
  – http://refcardz.dzone.com/
Yahoo!

- One of the oldest internet companies around
  - Founded 1994 by Jerry Yang and David Filo
  - “Yet Another Hierarchical Officious Oracle”
  - Started out as a hierarchical web catalog
  - Added email, groupware, online gaming, IM, and website hosting (Geocities) in late 90ties
  - During dot.com era, Yahoo already high worth increased tenfold to $120 per stock (initially 1996: $13 p.s.)
  - After dot.com crash, Yahoo was one of the few surviving big companies (… $4 p.s.)
• In 2003, buys AltaVista and AlltheWeb to build an own search engine to counter Google
• In 2005, Yahoo Music was founded (online music store and streaming)
• Also, in 2005 acquired multiple uprising Web 2.0 and social services
  – Flickr (photo), del.icio.us (bookmarks), Webjay (playlists), Upcoming (calendar)
• Since 2005 cooperation with Microsoft
  – Compatible mail and IM platforms
• In 2009, longterm cooperation with Microsoft
  – Search technology is integrated into Bing
• Currently, largest internet display advertisement vendor
  – Specialized on personalized advertisements
    • 2500 records per user per month!
• … and many more services
In this lecture, we will mainly focus on Sherpa/PNUTS – Clouds storage solution initially designed for various Yahoo social services

- e.g. Today’s Yahoo Pulse
Yahoo Social Storage Requirements

- Small records – 100KB or less
- Structured records – lots of fields, evolving
- Extreme data scale - Tens of TB
- Extreme request scale - Tens of thousands of requests/sec
- Low latency globally - 20+ datacenters worldwide
- High Availability - outages cost $millions
- Variable usage patterns - as applications and users change
• **Design Decisions in PNUTS**
  
  – **Record-orientation**: Routing, data storage optimized for low-latency record access
  
  – **Scale out**: Add machines to scale throughput (while keeping latency low)
  
  – **Asynchrony**: Pub-sub replication to far-flung datacenters to mask propagation delay
  
  – **Consistency model**: Reduce complexity of asynchrony for the application programmer
    
    • Use timeline consistency
  
  – **Cloud deployment model**: Hosted, managed service to reduce app time-to-market and enable on demand scale and elasticity
12.0 Data Storage at Yahoo

- Horizontal Cloud Service

![Diagram showing Horizontal Cloud Services]

- Provisioning & Virtualization e.g., EC2
- Batch Storage & Processing e.g., Hadoop & Pig
- Operational Storage e.g., S3, MObStor, Sherpa
- Edge Content Services e.g., YCS, YCPI
- Other Services: Messaging, Workflow, virtual DBs & Webserving

Simple Web Service API’s

ID & Account Management

Metering, Billing, Accounting

Security

Monitoring & QoS

Shared Infrastructure

Common Approaches to QA, Production Engineering, Performance Engineering, Datacenter Management, and Optimization
• Yahoo Cloud Stack

12.0 Data Storage at Yahoo

Provisioning (Self-serve)

Monitoring/Metering/Security

EDGE

YCS

YCPI

Brooklyn

...

WEB

VM/OS

yApache

PHP

App Engine

APP

VM/OS

Serving Grid

...

STORAGE

Sherpa

MOBStor

...

BATCH

Hadoop

...

Data Highway
12.0 Data Storage at Yahoo

- Web Data Management
  - Scan oriented workloads
  - Focus on sequential disk I/O
  - $ per cpu cycle

- Large data analysis (Hadoop)

- Structured record storage (PNUTS/Sherpa)
  - CRUD
  - Point lookups and short scans
  - Index organized table and random I/Os
  - $ per latency

- Blob storage (SAN/NAS)
  - Object retrieval and streaming
  - Scalable file storage
  - $ per GB
We will cover Sherpa/PNUTS in the following:

- Sherpa is Yahoo’s cloud storage platform which addresses most of Yahoo’s needs.
- PNUTS is a component of Sherpa mainly responsible of planning and executing queries.
- Also includes replication.
12.1 PNUTS/Sherpa Design Decisions

• **Scalability**
  – Thousands of machines
  – Easy to add capacity
  – **Restrict query language** to avoid costly queries
    • But still allow for commonly needed complex queries, e.g. *joins or grouping*

• **Geographic replication**
  – Asynchronous replication around the globe
  – **Low-latency local access**
    • *Store data close to the user*

• **High availability and fault tolerance**
  – Automatically recover from failures
  – Serve reads and writes despite failures
12.1 PNUTS/Sherpa Design Decisions

- **Consistency**
  - *Per-record guarantees*
  - *Timeline model*
  - Option to relax if needed

- **Multiple access paths**
  - *Hash table, ordered table*

- **Hosted service**
  - Applications plug and play
  - Share operational cost
12.1 PNUTS/Sherpa Design Decisions

• Technologies

Distributed Data Management
– Christoph Lofi – IfIS – TU Braunschweig

PNUTS
• Query planning and execution
• Index maintenance

YDOT FS
• Ordered tables

YDHT FS
• Hash tables

Zookeeper
• Consistency service

PNUTS API

Tabular API

Distributed infrastructure for tabular data
• Data partitioning
• Update consistency
• Replication

Tribble
• Pub/sub messaging

Applications

YCA: Authorization

PNUTS/Sherpa Design Decisions

Applications
12.1 PNUTS/Sherpa Design

• Sherpa also partitions data horizontally into tablets
  – PNUTS system decides automatically on splitting / merging / moving tablets

• Sherpa offers two different underlying tablet storage models
  – Hash-based (YDHT - Yahoo Distributed Hash Table)
    • Optimized for single-record direct-key accesses
    • e.g. primary user data
  – Ordered (YDOT - Yahoo Distributed Ordered Table)
    • Optimized for ranged-scans
    • e.g. user comments, user picture galleries, etc.
## YDHT FS – Hashed Tablets

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grape</td>
<td>Grapes are good to eat</td>
<td>$12</td>
</tr>
<tr>
<td>Lime</td>
<td>Limes are green</td>
<td>$9</td>
</tr>
<tr>
<td>Apple</td>
<td>Apple is wisdom</td>
<td>$1</td>
</tr>
<tr>
<td>Strawberry</td>
<td>Strawberry shortcake</td>
<td>$900</td>
</tr>
<tr>
<td>Orange</td>
<td>Arrgh! Don’t get scurvy!</td>
<td>$2</td>
</tr>
<tr>
<td>Avocado</td>
<td>But at what price?</td>
<td>$3</td>
</tr>
<tr>
<td>Lemon</td>
<td>How much did you pay for this lemon?</td>
<td>$1</td>
</tr>
<tr>
<td>Tomato</td>
<td>Is this a vegetable?</td>
<td>$14</td>
</tr>
<tr>
<td>Banana</td>
<td>The perfect fruit</td>
<td>$2</td>
</tr>
<tr>
<td>Kiwi</td>
<td>New Zealand</td>
<td>$8</td>
</tr>
</tbody>
</table>

### PNUTS/Sherpa Design

- A table showing the distribution of fruits and their prices.
- Prices range from $1 to $900.

### 12.1 PNUTS/Sherpa Design

- A design for distributed data management systems.
## 12.1 PNUTS/Sherpa Design

**YDOT FS – Ordered Tablets**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>Apple is wisdom</td>
<td>$1</td>
</tr>
<tr>
<td>Avocado</td>
<td>But at what price?</td>
<td>$3</td>
</tr>
<tr>
<td>Banana</td>
<td>The perfect fruit</td>
<td>$2</td>
</tr>
<tr>
<td>Grape</td>
<td>Grapes are good to eat</td>
<td>$12</td>
</tr>
<tr>
<td>Kiwi</td>
<td>New Zealand</td>
<td>$8</td>
</tr>
<tr>
<td>Lemon</td>
<td>How much did you pay for this lemon?</td>
<td>$1</td>
</tr>
<tr>
<td>Lime</td>
<td>Limes are green</td>
<td>$9</td>
</tr>
<tr>
<td>Orange</td>
<td>Arrgh! Don’t get scurvy!</td>
<td>$2</td>
</tr>
<tr>
<td>Strawberry</td>
<td>Strawberry shortcake</td>
<td>$900</td>
</tr>
<tr>
<td>Tomato</td>
<td>Is this a vegetable?</td>
<td>$14</td>
</tr>
</tbody>
</table>
• Sherpa is hosted in multiple data centers which are geographically distributed
  – Called colos at Yahoo (colocation facility)
  – A colo for every major area is hosted
  – Main idea: Geographic Data Distribution
    • Users of a certain area primarily interact just with “their” local data center to prevent intercontinental high-latency connections
• Data Center Architecture

**Local region**
- Clients
- Tablet Controller
- Routers
- REST API

**Remote regions**
- Tribble
- Messaging Service
- Storage units

Distributed Data Management – Christoph Lofi – IfIS – TU Braunschweig
• Basic Tablet system roughly similar to Bigtable

• Storage Units
  – Store multiple tablets

• Tablet Controller
  – Responsible for load balancing and maintenance

• Routers
  – Responsible for routing queries to the correct storage unit
    • Central access points for clients
  – Aggregate results if query spans multiple storage units
  – Big difference to e.g. Bigtable
• **Query Processing: Accessing**

  – **Routers** route query to responsible storage unit
• **Query Processing: Bulk Read**
  – Router splits query to storage units
  – Gathers and aggregates results

1. \{k_1, k_2, \ldots, k_n\}

2. Scatter/gather server

SU

SU

SU
Clustered, ordered retrieval of records (YDOT)
12.1 PNUTS/Sherpa Queries

- Updates

1. Write key k
2. Write key k
3. Write key k
4. Message brokers
5. SUCCESS
6. Write key k
7. Sequence # for key k
8. Sequence # for key k

Remote Data Centers

Routers

SU

SU

SU
• Unlike Bigtable, PNUTS uses asynchronous updates of replicas
  – Bigtable is designed to run in a single data center, single updates are rather large
    • Synchronous tolerable expensive
  – Sherpa/PNUTS is designed to run on a global scale with tiny updates
    • e.g. update social network status message
      – Replicate from US East Coast to Singapore (latency commonly around 1 sec)
12.1 PNUTS Consistency

• Asynchronous Replication
• **Asynchronous** replica updates will increase risk for stale copies and **inconsistency**

• Goal: Make it easier for applications to reason about updates and cope with asynchrony
  – What happens to a record with primary key “Alice”?

As the record is updated, copies may get out of sync.
12.1 PNUTS Consistency

• Each record is assigned to one site
  – That site is the master of that record and holds the most actual version
    • All updates are preformed on the master and asynchronously propagated to clones
    • Master membership indicated as a value in a column
    • Master is located geographically close to the user who usually queries the record
• All users query records from their geographically closest site
  – That site is usually the master of that user’s data
  – If a user request data mastered by a remote site, the request is also served from the requester’s local site
    • e.g. checking profile of an oversea friend
    • Query result may be stale if updates are not fully propagated to local replica yet
    • Applications may also perform a priority query which will fetch the data from the master colo
• **Transferring mastership**
  
  – If a user temporarily changes his usage behavior, performance may be suboptimal
    • e.g. business trip
  
  – If the usage pattern changes for a longer period of time, mastership can be transferred to another colo
    • e.g. after moving to a new city
12.1 PNUTS Consistency

• Mastering
• Updating replicas
  – The replication protocol ensures that all sites are updated in the same *time order*
  – Each update is tagged with a timestamp / version number
  – Updates are propagated to replicas in the exact same order than they were received at the master
12.1 PNUTS Consistency

• Example: Social Alice

West

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Busy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Busy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>???</td>
</tr>
</tbody>
</table>

East

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Free</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>???</td>
</tr>
</tbody>
</table>

Record Timeline

- Busy
- Free
12.1 PNUTS Consistency

• Consistency Model

In general, reads are served using a local copy.
12.1 PNUTS Consistency

- Consistency Model

![Diagram](image_url)

- Read up-to-date
- Stale version
- Stale version
- Current version

Generation 1

v. 1  v. 2  v. 3  v. 4  v. 5  v. 6  v. 7  v. 8

But application can request and get current version
12.1 PNUTS Consistency

- Consistency Model

Or variations such as “read forward”—while copies may lag the master record, every copy goes through the same sequence of changes
12.1 PNUTS Consistency

• Consistency Model

Achieved via per-record primary copy protocol
(To maximize availability, record masterships automatically transferred if site fails)
Can be selectively weakened to eventual consistency
(local writes that are reconciled using version vectors)
12.1 PNUTS Consistency

- Consistency Model

Write if = v.7

ERROR

Stale version

Stale version

Current version

Test-and-set writes facilitate per-record transactions
Summary Consistency techniques

- **Per-record mastering**
  - Each record is assigned a “master region”
    - May differ between records in one tablet
  - Updates to the record forwarded to the master region
  - Ensures consistent ordering of updates

- **Tablet-level mastering**
  - Each tablet is assigned a “master region”
  - Inserts and deletes of records forwarded to the master region
  - Master region decides tablet splits

- These details are hidden from the application
  - Except for the latency impact!
12.1 PNUTS Consistency

- Bulk Inserts / Update / Replace

1. Client feeds records to bulk manager
2. Bulk loader transfers records to SU’s in batches
   - Bypass routers and message brokers
   - Efficient import into storage unit
• YDOT bulk inserts can cause performance hotspots

• Solution: preallocate tablets
12.1 PNUTS Complex Queries

• PNUTS offers support for complex queries like e.g. **joins** or **grouping** operations
  – Unlike most other cloud storage systems!
  – **Typical example:** Grouped Join
    • Return all gallery pictures of a user with all comments (and their authors) grouped by picture ID
– Naïve approach shows unacceptable performance
  • e.g. data used for joins may scattered across storage servers
  • Data and / or queries need to be shipped for complex multi-sever joins

• Solution
  – Use materialized views!
    • Most internet-scale systems use only very few complex queries known to the application programmer
    • Materialize those queries!

\[
\text{userPhotosAndComments}
\]

\[
\begin{array}{llllllll}
\text{u_id} & \text{name} & \text{p_id} & \text{owner} & \text{binary} & \text{author_id} & \text{author_name} & \text{text} \\
\end{array}
\]
12.1 PNUTS Complex Queries

– **Materialized views** can be treated as just another replica and data transformation
  
  • Use similar propagation techniques
  
  • Materialized views are also updated asynchronously when base tables are updated
    
    – Also, priority queries are possible which force reading the latest copy
• PNUTS focuses on following features
  
  – **Geographic replication and query optimization**
    • Each record is mastered by the data center closest to the owning user
    • All users usually only query the closest data center
    • Updates are always performed on the master copy
      – Asynchronous propagation of updates
  
  – **Timeline consistency**
    • Eventually, all replicas will be consistent
    • In contrast to system like e.g. Dynamo, all updates are performed in the original query order
  
  – **Support for complex queries** (i.e. joins and groups)
    • Realized by materialized views
      – Queries must be known upfront
    • Updated asynchronously
• **Query expressiveness**

![Diagram showing the query expressiveness of different data management systems. The systems are ordered from simple (left) to feature rich (right):

- **Simple**
  - Dynamo
  - PNUTS, BigTable
  - Oracle

- **Object retrieval**
  - Retrieval from single table of objects/records

- **Feature rich**
  - SQL

Systems include Dynamo, PNUTS, BigTable, and Oracle.]
• Consistency model

Best effort → Dynamo: Eventual consistency → PNUTS: Timeline consistency → BigTable: Synchronous Replication → Oracle: ACID → Strong guarantees

Object-centric consistency → Program centric consistency
• Data model

Dynamo

PNUTS

BigTable

Oracle

Flexibility, Schema evolution

No schema  Flexible schema

Optimized for Fixed schemas
• Elasticity

- Oracle: Limited (via data distribution)
- PNUTS: VLSD (Very Large Scale Distribution / Replication)

Inelastic  |  Elastic
--------|--------
Oracle   |  PNUTS
Limited  |  Dynamo
(Via data distribution) | (Very Large Scale Distribution / Replication)
### Comparing PNUTS

<table>
<thead>
<tr>
<th>PNUTS</th>
<th>Versus PNUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-partitioned SQL stores</td>
<td></td>
</tr>
<tr>
<td>- Microsoft Azure SDS</td>
<td></td>
</tr>
<tr>
<td>- Amazon SimpleDB</td>
<td></td>
</tr>
<tr>
<td>Multi-tenant application databases</td>
<td></td>
</tr>
<tr>
<td>- Salesforce.com</td>
<td></td>
</tr>
<tr>
<td>- Oracle on Demand</td>
<td></td>
</tr>
<tr>
<td>Mutable object stores</td>
<td></td>
</tr>
<tr>
<td>- Amazon S3 or Dynamo</td>
<td></td>
</tr>
<tr>
<td>More expressive queries</td>
<td></td>
</tr>
<tr>
<td>Users must control partitioning</td>
<td></td>
</tr>
<tr>
<td>Limited elasticity</td>
<td></td>
</tr>
<tr>
<td>Highly optimized for complex workloads</td>
<td></td>
</tr>
<tr>
<td>Limited flexibility to evolving applications</td>
<td></td>
</tr>
<tr>
<td>Inherit limitations of underlying data management system</td>
<td></td>
</tr>
<tr>
<td>Object storage versus record management</td>
<td></td>
</tr>
</tbody>
</table>
12.1 PNUTS

- Query for things
  - Sherpa
  - MySQL
  - Oracle
- Scan everything
  - BigTable
  - Everest
- Records
  - Hadoop
- Files
  - MObStor
  - YMDB
  - Filer
## 12.1 PNUTS

<table>
<thead>
<tr>
<th></th>
<th>Elastic</th>
<th>Operability</th>
<th>Availability</th>
<th>Global low latency</th>
<th>Structured access</th>
<th>Updates</th>
<th>Consistency model</th>
<th>SQL / ACID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sherpa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y! UDB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MySQL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oracle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BigTable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cassandra</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Elastic: Green
- Operability: Blue
- Availability: Blue
- Global low latency: Blue
- Structured access: Blue
- Updates: Blue
- Consistency model: Red
- SQL / ACID: Red
12.1 PNUTS - Literature

- “Efficient Bulk Insertion into a Distributed Ordered Table” (SIGMOD 2008) Adam Silberstein, Brian Cooper, Utkarsh Srivastava, Erik Vee, Ramana Yerneni, Raghu Ramakrishnan


- Parag Agrawal, Adam Silberstein, Brian F. Cooper, Utkarsh Srivastava and Raghu Ramakrishnan, “Asynchronous View Maintenance for VLSD Databases”, SIGMOD 2009

• Just storing massive amounts of data is often not enough!
  – Often, we also need to process and transform that data

• **Large-Scale Data Processing**
  – Use thousands of worker nodes within a *computation cluster* to process large data batches
    • But don’t want hassle of managing things

• **Map & Reduce** provides
  – Automatic parallelization & distribution
  – Fault tolerance
  – I/O scheduling
  – Monitoring & status updates
12.2 Map & Reduce

• Initially, implemented by Google for building the Google search index
  – i.e. crawling the web, building inverted word index, computing page rank, etc.
  • General framework for parallel high volume data processing

– Also available as Open Source implementation as part of Apache Hadoop
  • http://hadoop.apache.org/mapreduce/
12.2 Map & Reduce

• Base idea
  – There is a large number of input data, identified by a key
    • i.e. input given as key-value pairs
    • e.g. all web pages of the internet identified by their URL
  – A map operation is a simple function which accepts one input key-value pair
    • A map operation runs as a autonomous thread on one single node of a cluster
      – Many map jobs can run in parallel on different input keys
    • Returns for a single input key-value pair a set of intermediate key-value pairs
      – \( \text{map}(\text{key}, \text{value}) \rightarrow \text{Set of intermediate (key, value)} \)
    • After map job is finished, the node is free to perform another map job for the next input key-value pair
      – A central controller distributes map jobs to free nodes
– After input data is mapped, reduce jobs can start
– `reduce(key, values)` is run for each `unique` key emitted by `map()`
  • Each reduce job is also run autonomously on one single node
    – Many reduce jobs can run in parallel on different intermediate key groups
  • Reduce emits final output of the map-reduce operation
  • Each reduce job takes all map tuples with a given key as input
  • Generate usually one, but possible more output tuples
12.2 Map & Reduce

• Each reduce is executed on a set of intermediate map results which have the same key
  – To efficiently select that set, the intermediate key-value pairs are usually shuffled
    • i.e. just sorted and grouped by their respective key
  – After shuffling, reduce input data can be selected by a simple range scan
12.2 Map & Reduce

• Responsibility of the map and reduce master
  • Often, also called scheduler
    – Assign Map and Reduce tasks to workers on nodes
      • Usually, map tasks are assigned to worker nodes as a batch and not one by one
        – Often called a split, i.e. subset of the whole input data
        – Split often implemented by a simple hash function with as many buckets as worker nodes
        – Full split data is assigned to worker node which starts a map task for each input key-value pair
    – Check for node failure
    – Check for task completion
    – Route map results to reduce tasks
12.2 Map & Reduce

• Map and Reduce overview
• Master is responsible for **worker node fault tolerance**
  – Handled via re-execution
    • Detect failure via periodic heartbeats
    • Re-execute completed + in-progress map tasks
    • Re-execute in progress reduce tasks
    • Task completion committed through master
  – Robust: lost 1600/1800 machines once → finished ok
• Master failures are not handled
  – Unlikely due to redundant hardware…
12.2 Map & Reduce

- Example: **Counting words in documents**

```python
map(key, value):
  // key: doc name;
  // value: text of doc
  for each word w in value:
    emit(w, 1);

reduce(key, values):
  // key: a word;
  // values: list of counts
  result = 0;
  for each v in values)
    result += v;
  emit(key, result);
```
12.2 Map and Reduce

- Example: **Counting words in documents**

- **doc1**: “distributed db and p2p”

- **doc2**: “map and reduce is a distributed processing technique for db”

```
map (key, value)
    distributed
    db
    and
    p2p
    map
    and
    reduce
    is
    a
    distributed

reduce (key, value)
    distributed
    db
    and
    p2p
    map
    and
    reduce
    is
    a
    distributed
```
12.2 Map and Reduce

• Improvement: **Combiners**
  – Combiners are **mini-reducers** that run in-memory after the map phase
  – Used to group rare map keys into larger groups
    • e.g. word counts: group multiple extremely rare words under one key (and mark that they are grouped…)
  – Used to **reduce** network and worker scheduling overhead
12.2 Map and Reduce

- Showcase: machine usage during **web indexing**
  - Fine granularity tasks: map tasks $\mapsto$ machines
    - Minimizes time for fault recovery
    - Can pipeline shuffling with map execution
    - Better dynamic load balancing
  - Showcase uses 200,000 map & 5,000 reduce tasks
  - Running on 2,000 machines

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
Process       & Time \multicolumn{1}{c}{\textbf{MapReduce}()} \textbf{… wait …} \\
\hline
User Program & \\
\hline
Master       & \\
Worker 1     & \\
Worker 2     & \\
Worker 3     & \\
Worker 4     & \\
\hline
\end{tabular}
\end{table}

- Assign tasks to worker machines...
  - Map 1, Map 3
  - Map 2
  - Read 1.1, Read 1.3
  - Read 1.2
  - Read 2.1
  - Read 2.2, Read 2.3
  - Reduce 1, Reduce 2
### 12.2 MR - Performance

**MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03**

**Started:** Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>0</td>
<td>323</td>
<td>878934.6</td>
<td>1314.4</td>
<td>717.0</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>323</td>
<td>717.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Counters**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>72.5</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>docs-index-hits</td>
<td>145825686</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>506631</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>508192</td>
</tr>
<tr>
<td>mr-operator-output</td>
<td>506631</td>
</tr>
</tbody>
</table>
### MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>1857</td>
<td>1707</td>
<td>878934.6</td>
<td>191995.8</td>
<td>113936.6</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>113936.6</td>
<td>57113.7</td>
<td>57113.7</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>57113.7</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

#### Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>699.1</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>349.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>500441194</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17290135</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17331371</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17290135</td>
</tr>
</tbody>
</table>
## MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

**Started:** Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>5354</td>
<td>1707</td>
<td>878934.6</td>
<td>406020.1</td>
<td>241058.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>241058.2</td>
<td>196362.5</td>
<td>196362.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>196362.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>704.4</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>371.9</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>5000364228</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17300709</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17342493</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17300709</td>
</tr>
</tbody>
</table>
### MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

**Started:** Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec  
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>8841</td>
<td>1707</td>
<td>878934.6</td>
<td>621608.5</td>
<td>369459.8</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>369459.8</td>
<td>326986.8</td>
<td>326986.8</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>326986.8</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

#### Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>706.5</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>419.2</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>4982870667</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17229926</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operators-calls</td>
<td>17272056</td>
</tr>
<tr>
<td>mr-operators-outputs</td>
<td>17229926</td>
</tr>
</tbody>
</table>
12.2 MR - Performance

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>195</td>
<td>305</td>
<td>523499.2</td>
<td>523389.6</td>
<td>523389.6</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>195</td>
<td>523389.6</td>
<td>2685.2</td>
<td>2742.6</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.3</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>45.7</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>2313178</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>7936</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>1954105</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>1954105</td>
</tr>
</tbody>
</table>
12.2 MR - Performance

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>133837.8</td>
<td>136929.6</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.1</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1238.8</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51738599</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>51738599</td>
</tr>
</tbody>
</table>
12.2 MR - Performance

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>263283.3</td>
<td>269351.2</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1225.1</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51842100</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>51842100</td>
</tr>
</tbody>
</table>
# 12.2 MR - Performance

### MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

- **Started:** Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec
- **Workers:** 1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>520468.6</td>
<td>520468.6</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>406</td>
<td>94</td>
<td>520468.6</td>
<td>512265.2</td>
<td>514373.3</td>
</tr>
</tbody>
</table>

### Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>849.5</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>35083350</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>35083350</td>
</tr>
</tbody>
</table>
12.2 MR - Performance

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>519774.3</td>
<td>519774.3</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>499</td>
<td>1</td>
<td>519774.3</td>
<td>519735.2</td>
<td>519764.0</td>
</tr>
</tbody>
</table>

**Counters**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1.9</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>105466</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index</td>
<td>0</td>
</tr>
<tr>
<td>merge-merge-calls</td>
<td>73442</td>
</tr>
<tr>
<td>merge-merge-outs</td>
<td>73442</td>
</tr>
</tbody>
</table>
• **PageRank** is one of the major algorithm behind Google Search
  – See our wonderful **IR lecture** (No 12)!!
  – **Key Question:** How important is a given website?
    • Importance independent of query
  – Idea: other pages “vote” for a site by linking to it
    • also called “giving credit to”
    • Pages with many votes are probably important
  – If an important site “votes” for another site, that vote has a higher weight as when an unimportant site votes
Given page $x$ with in-bound links $t_1, \ldots, t_n$, where

- $C(t)$ is the out-degree of $t$
- $\alpha$ is probability of random jump
- $N$ is the total number of nodes in the graph

$$PR(x) = \alpha \left( \frac{1}{N} \right) + (1 - \alpha) \sum_{i=1}^{n} \left( \frac{PR(t_i)}{C(t_i)} \right)$$
• Properties of PageRank
  – Can be computed iteratively
  – Effects at each iteration is local

• Sketch of algorithm:
  – Start with seed $PR_i$ values
  – Each page distributes $PR_i$ “credit” to all pages it links to
  – Each target page adds up “credit” from multiple inbound links to compute $PR_{i+1}$
  – Iterate until values converge
Map Step: Distribute PageRank “Credits” to link targets

Reduce Step: Gather up PageRank “credit” from multiple sources to compute new PageRank value
Turbo-Charging Map and Reduce

Naïve approach for implementing Map and Reduce

- Move data to workers
- Have a cluster of computation nodes
  - A master, multiple workers
- Master has access to all data
- Master splits the data and assigns map tasks
  - Master transfers input data to workers
- Map results are somehow transferred to reduce workers
  - Directly? Pipelined? Via master?
- In short: a lot of data shipping is necessary
• **Location aware file system approach**
  – Rely on a distributed file system like GFS or HFS
    • Or even on a higher layers like Bigtable or HBase
    • All those systems are especially designed for increased Map and Reduce performance

• **Idea:** Each processing node runs a GFS chunk server and a Map & Reduce Worker
  – Input data is stored in large chunks in GFS
  – Start a worker task which uses a local chunk as batch map input
    • **Read sequentially** through the local chunk
    • GFS as well as BigTable are optimized for sequential scans
Map workers sequentially **append**s intermediate key-value pairs to another chunk (local or remote)

- GFS as well as BigTable are optimized for append operations...

Reduce workers also scan through local chunks as input and append results to a local or remote chunk

**File system responsible for distributing data**

- Very easy scheduling for master
  - Just assign local data to workers
- Fault tolerant (data loss improbable)
• The Cloud beyond Storage
  – Business Concepts
    • CaaS
    • SaaS
    • PaaS
  – Computing as a commodity
  – Platform Overview