Distributed Programming the Google Way

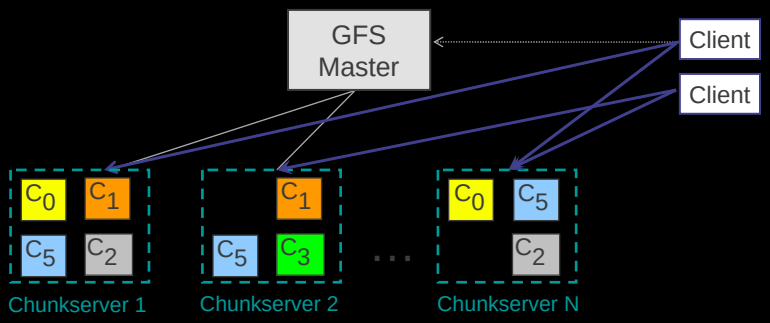
Gregor Hohpe
Software Engineer
www.EnterpriseIntegrationPatterns.com

Scalable & Distributed

- Fault tolerant distributed disk storage: *Google File System*
- Distributed shared memory: *Bigtable*
- Parallel programming abstraction: *MapReduce*
- Domain Specific Languages: *Sawzall*

GFS: Google File System

- Data replicated 3 times. Upon failure, software re-replicates.
- Master: Manages file metadata. Chunk size 64 MB.



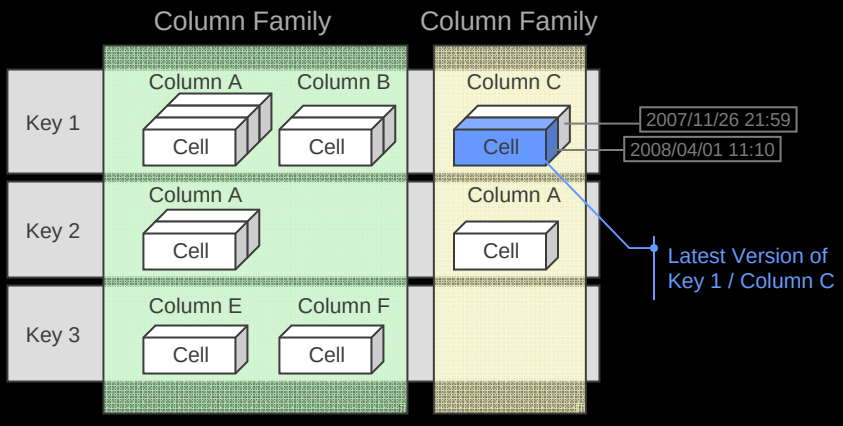
http://research.google.com/archive/gfs-sosp2003.pdf

Google

3

Bigtable: A sparse, distributed, persistent, multidimensional, sorted Map

(RowKey, ColumnFamily:Column, Timestamp) → Value



http://research.google.com/archive/bigtable-osdi06.pdf

Google

4

Map-Reduce

- Express computation as Map / Group / Reduce

```
map(in_key, data)
  → list(key, value)
(group output by key)
reduce(key, list(values))
  → list(out_data)
```

- Well suited for "embarrassingly parallel" problems
- Open source implementation: Hadoop

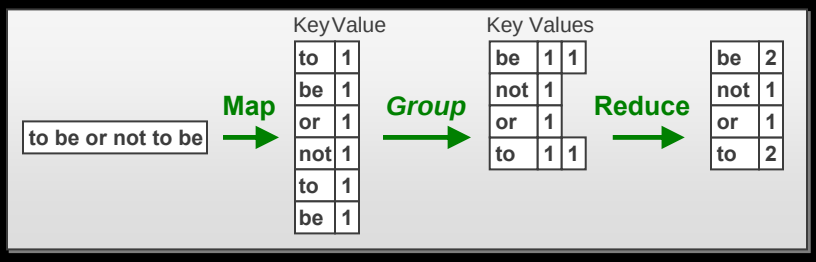
http://research.google.com/archive/mapreduce.html

Google

Simple Example: Word Count

- Chose the key so you get the most out of the framework

```
map(in_key, data):
  for word in data.split():
    output(word, 1)
reduce(key, list(values)):
  print key, len(values)
```



Google

Log Processing: Sawzall

- Domain-specific language
- Process one record at a time
- Aggregation externalized into "tables"

```
count: table sum of int;
total: table sum of float;
---
number: float = string(input);
emit count <- 1;
emit total <- number;
```

```
3.5
2
1.25
-------
count[] = 3
total[] = 6.75
```

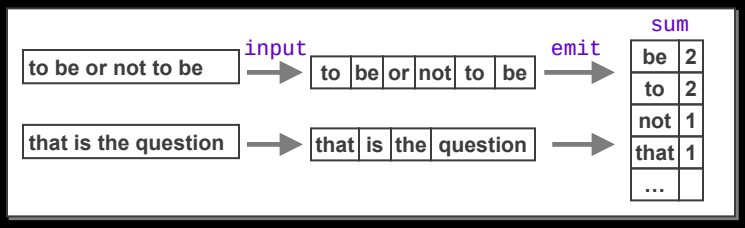
- "Programming in the first derivative"

http://labs.google.com/papers/sawzall.html

Google

Simple Example: Word Count

```
word_count: table sum[string] of int;
---
line = input;
words = split(line);
for each word in words:
  emit word_count[words] <- 1;
```



Google

Underlying Considerations

1. Sharding
2. Less is More
3. Autonomy
4. Expect Failure
5. Power to the Runtime
6. Favor Stateless
7. Separate Stateless from Stateful
8. Precision vs. Speed

Conscious trade-offs

Google

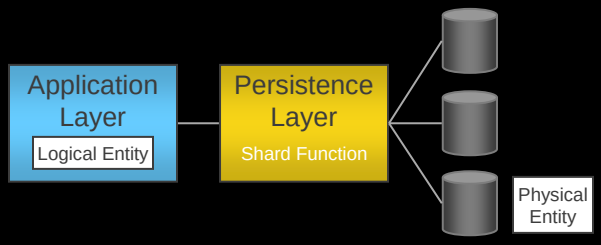
Sharding

Divide and Conquer

Google

Partition Data Across DB Instances

- Shard function, e.g. customer ID
- Hierarchies (one-way) work well
- Many-to-many relationships (two-way) difficult
- "Special Shard" / All shard queries



Google

Less is More

Google

Bigtable, Not Bigdatabase

- Less is More:
  - No transactions
  - No schema
  - No foreign keys
  - No join
  - No relational algebra, Cartesian product, etc
  - No SQL
- Single row updates are atomic. Everything else is not.
- Only basic data types: string, counter, protocol buffer

Google

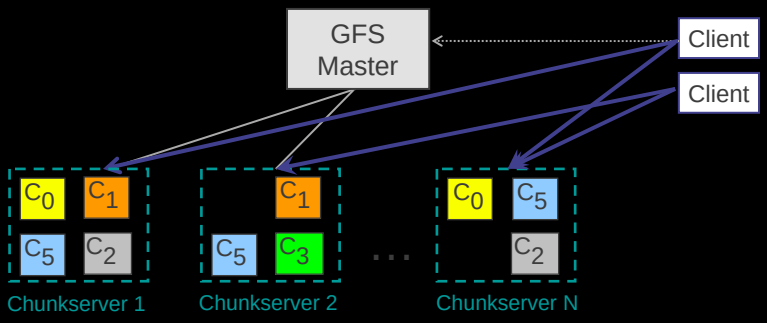
Autonomy

Keep Going without Supervision

Google

GFS

- Direct communication between client and Chunk server
- Large Chunk size (64MB)



Google

15

Expect Failure

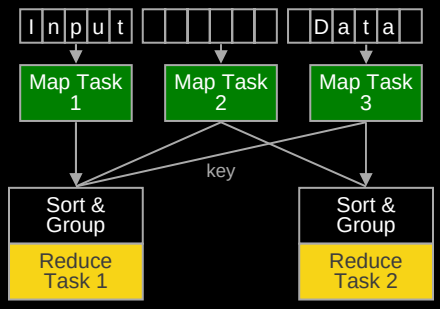
Not If, But When

Google

16

Expect Failure: GFS & MapReduce

- GFS:
  - Data replicated 3 times.
  - Upon failure, software re-replicates.
- MapReduce:
  - Restarts failed map / reduce workers
  - Detects key/value pairs that cause crashes, skips
  - Tougher to deal with: laggards



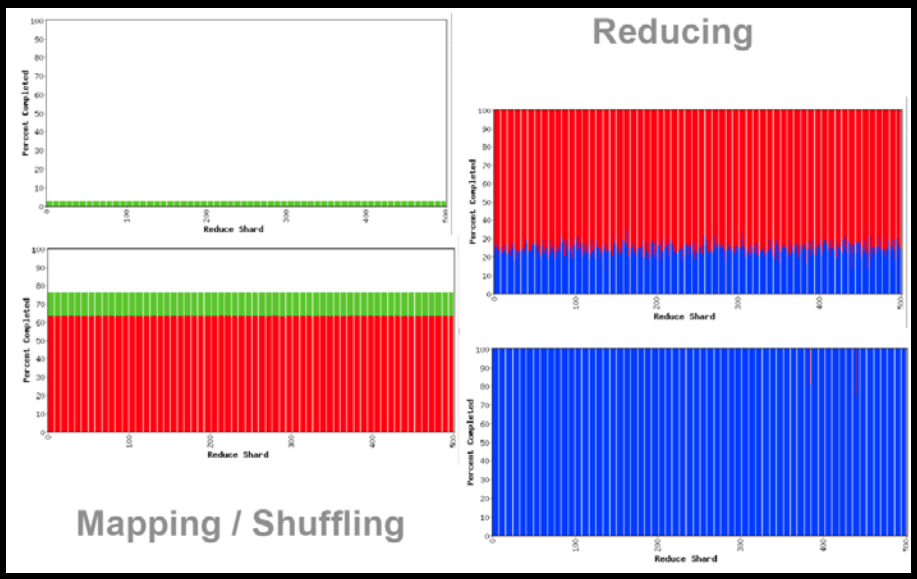
Google

Power to the Runtime

Free Flow Instead of Strict Rules

Google

MapReduce Execution



Sawzall Quantifiers

- Descriptive as opposed to Prescriptive

```
function(word: string): bool {
  when(i: some int; word[i] != word[$-1-i])
    return false;
  return true;
};
```

- some / all / each

Favor Stateless

Don't Remember a Thing

MapReduce / Sawzall /Bigtable

- Map and Reduce step are stateless

```
map(in_key, data)
 → list(key, value)
```

```
reduce(key, list(values))
→ list(out_data)
```

- Sawzall views input data as set, not list
- Bigtable has set() operation vs. insert / update

Separate Stateless From Stateful

The Real World is Rarely Stateless

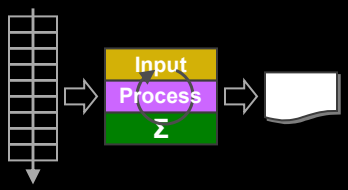
Google

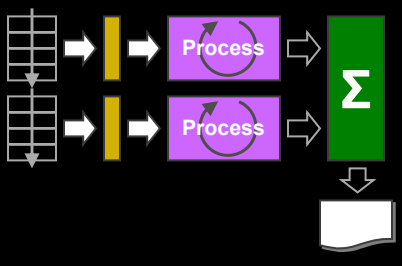
Sawzall

State!!

```
while !eof(file) {
  line = readline(file);
  words = split(line);
  for each word in words:
    map[word]++;
}
```

```
map: table sum[string] of int;
line = input;
words = split(line);
for each word in words:
  emit map[words] <- 1;
```





Google

Precision vs. Speed

Faster is Better (in Software)

Trade precision for speed: Sawzall

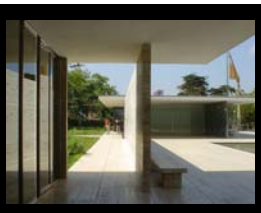
- Top() function:
  Statistical samplings that record the `top N' data items.

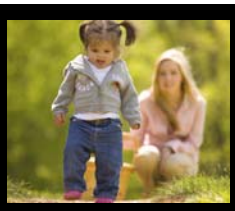
```
// This type is for estimating the most common
// entries based on the CountSketch algorithm from
// "Finding Frequent Items in Data Streams",
// Moses Charikar, Kevin Chen and Martin
// Farach-Colton.
```

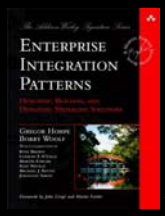
It's all about trade-offs!

- GFS
  - Large chunk size (64MB)
  - Optimized for sequential read, not random access
- Bigtable
  - Optimized for read-intensive applications
  - Distributed, but not transactional
- Sawzall
  - Cannot detect duplicate rows

Google







http://www.flickr.com/photos/hellolapomme/2552346073
http://www.flickr.com/photos/lorkano/3277969419
http://www.flickr.com/photos/mukluk/288925731

Thank You!