# TP3 - Sistemas Distribuidos
# Map-Reduce

Daniel J. Foguelman -> Guido Chari -> Florencia Iglesias

DC - FCEyN - UBA

Sistemas Operativos
2c - 2014

## Quote

"Map-reduce is **a programming model** for expressing **distributed computations** on **massive amounts of data** and an execution framework for **large-scale data processing** on clusters of **commodity servers**"

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

¿Qué es?

- Es un modelo de programación
- Diseñado para el cómputo de datos de gran escala.
- Con una arquitectura de bajo costo (depende la implementación).
- Donde la performance es un un aspecto importante.
- Y un requerimiento fuerte: Escalabilidad lineal.

¡Datos, Datos, Datos !

# Google procesa 24 petabytes/día

Conceptos importantes

- Paralelización y distribución automática.
- Tolerancia a fallos.
- I/O scheduling.
- Load balancing.

## Map - Reduce

- **Map** es una función sencilla que opera sobre una porción del set de datos.

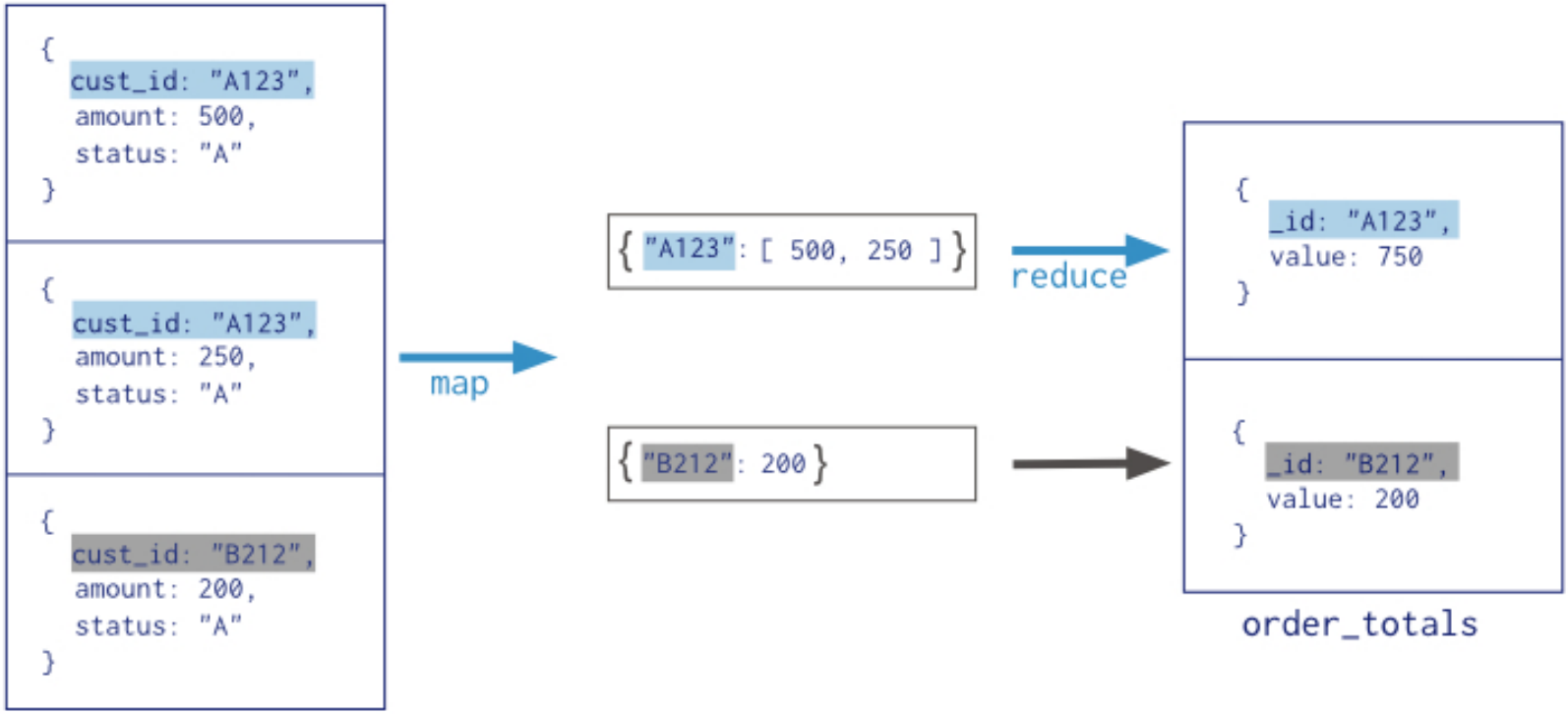
```
> map (in_key, in_value) ->
        list(out_key, intermediate_value)
```

- **Reduce** es una función que agrupa los resultados.

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

- Orientado a documentos, clave-valor.

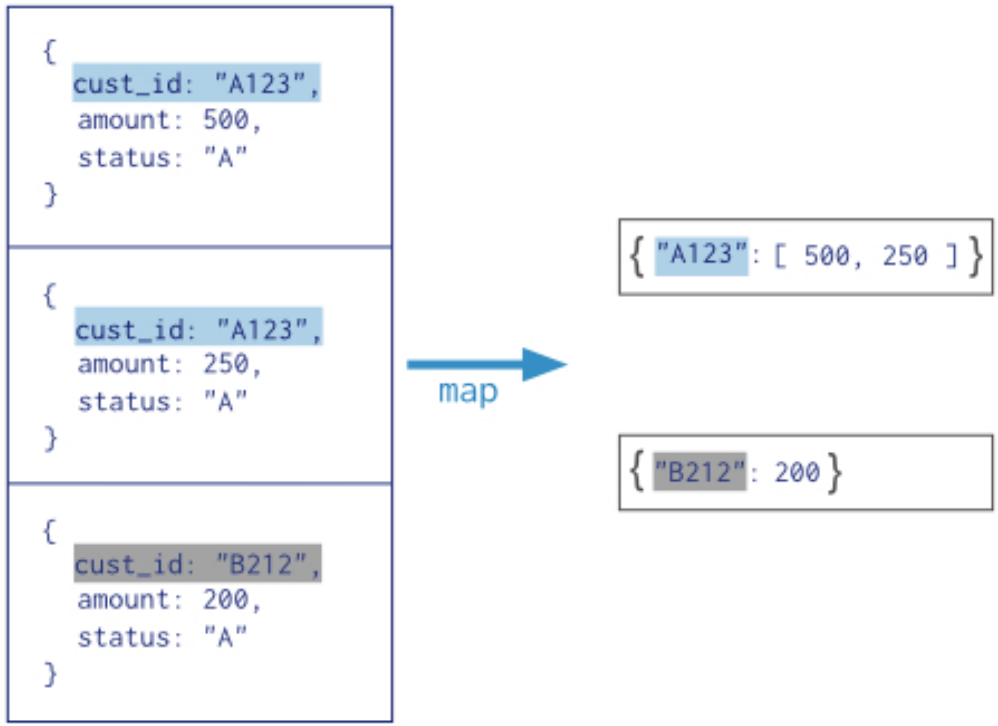
# Un ejemplo




http://docs.mongodb.org/manual/core/map-reduce

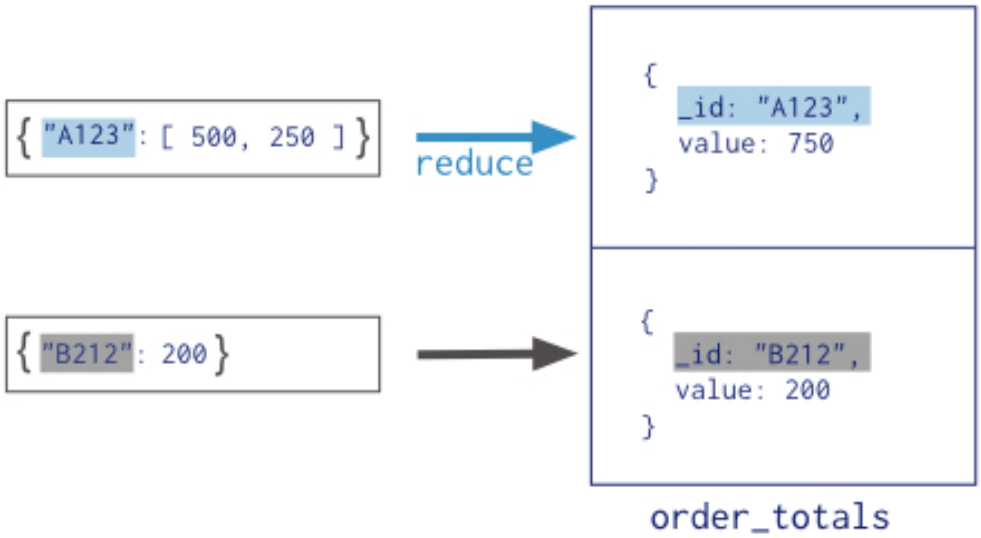
# Map

```
map = function() {
    emit(this.cust_id, this.amount);
}
```

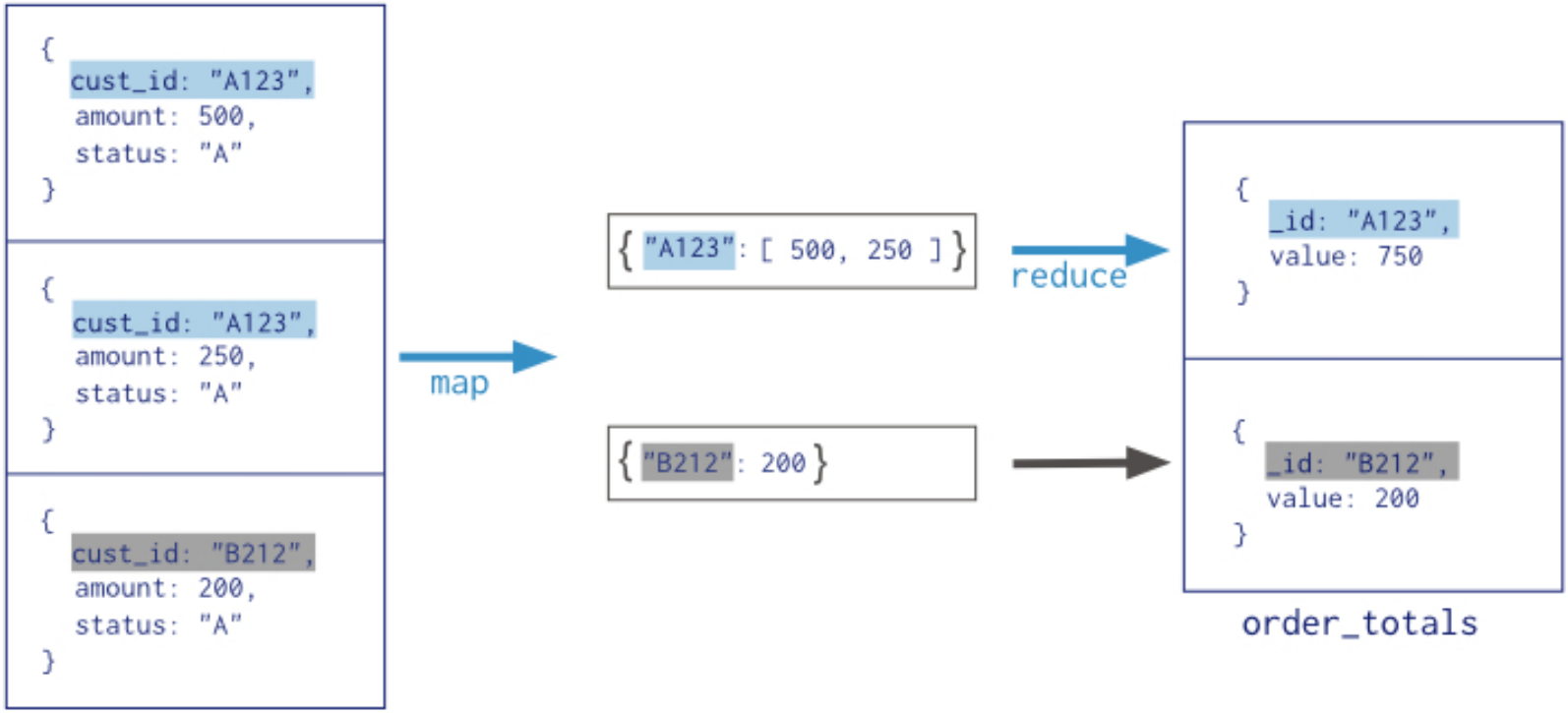


# Reduce

```
reduce = function(key, values) {
    return Array.sum(values);
}
```



```
> db.orders.mapReduce(map, reduce, order_totals);
```



La función **Reduce** puede ser invocada más de una vez para la misma clave. La salida anterior de la función para esa clave se utiliza como uno de los valores de entrada a la siguiente invocación de la función para esa misma clave.

```
(key, [A, B ]) -> reduce(key, [ A, B ]) = D
(key, [C])  -> reduce(key, [ C, D] )
```

Propiedades:

- Ser **asociativa**:

  ```
  reduce(key, [ C, reduce(key, [ A, B ]) ] ) == reduce( key, [ C, A, B ] )
  ```

- Ser **commutativa**:

  ```
  reduce( key, [ A, B ] ) == reduce( key, [ B, A ] )
  ```

- Ser **idempotente**:

  ```
  reduce( key, [ reduce(key, valuesArray) ] ) == reduce( key, valuesArray )
  ```

Ejemplo: El promedio no satisface las propiedades

- Salida del **map**:

```
(K1, [3, 2 ])
(K1, [5, 7])
```

- **reduce**

```
reduce = function(key, values) {
  return Array.sum(values)/values.length;
}
```

Ejemplo: El promedio no satisface las propiedades

- Salida del **map**:

```
(K1, [3, 2 ])
(K1, [5, 7])
```

- **reduce**

```
reduce = function(key, values) {
  return Array.sum(values)/values.length;
}
```

  - Salida:

```
(K1, [3, 2 ])  = (3 + 2) / 2 = 2.5
(K1, [5, 7]) -> (K1, [2.5, 5, 7]) = (2.5 + 5 + 7) / 3 = 4.82
```

Ejemplo: El promedio no satisface las propiedades

- Salida del **map**:

```
(K1, [3, 2 ])
(K1, [5, 7])
```

- **reduce**

```
reduce = function(key, values) {
  return Array.sum(values)/values.length;
}
```

  - Salida:

```
(K1, [3, 2 ])  = (3 + 2) / 2 = 2.5
(K1, [5, 7]) -> (K1, [2.5, 5, 7]) = (2.5 + 5 + 7) / 3 = 4.82
```

  - Es distinto de

```
(K1, [3, 2 , 5, 7])  = (3 + 2 + 5 + 7) / 4 = 4.25
```

¿Cómo resolvemos el problema?

¿Cómo resolvemos el problema?

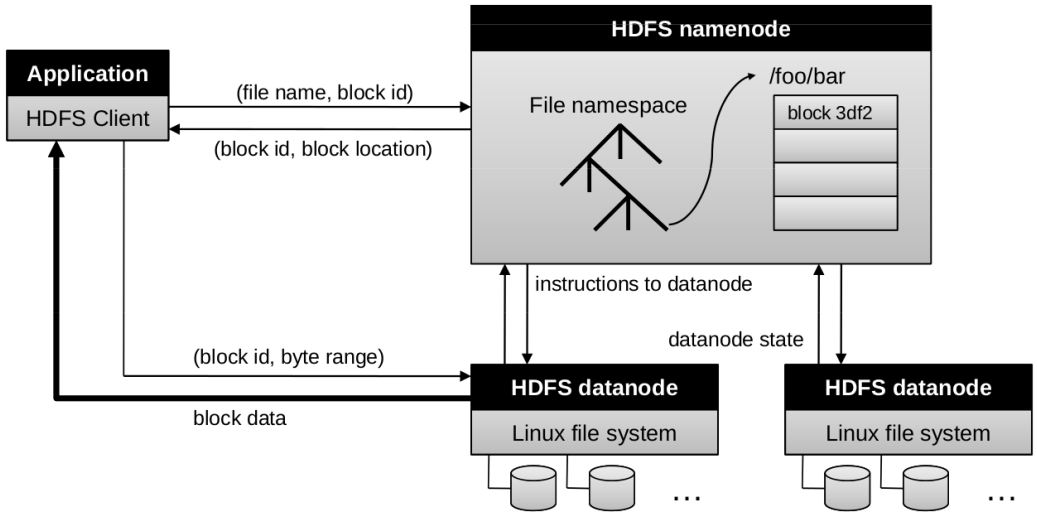
- Agregamos **finalize(Key, reducedValue)**

Idea

- **map** no se modifica
- **reduce** sólo calcula la suma de los elementos de values
- **finalize** realiza los cálculos restantes

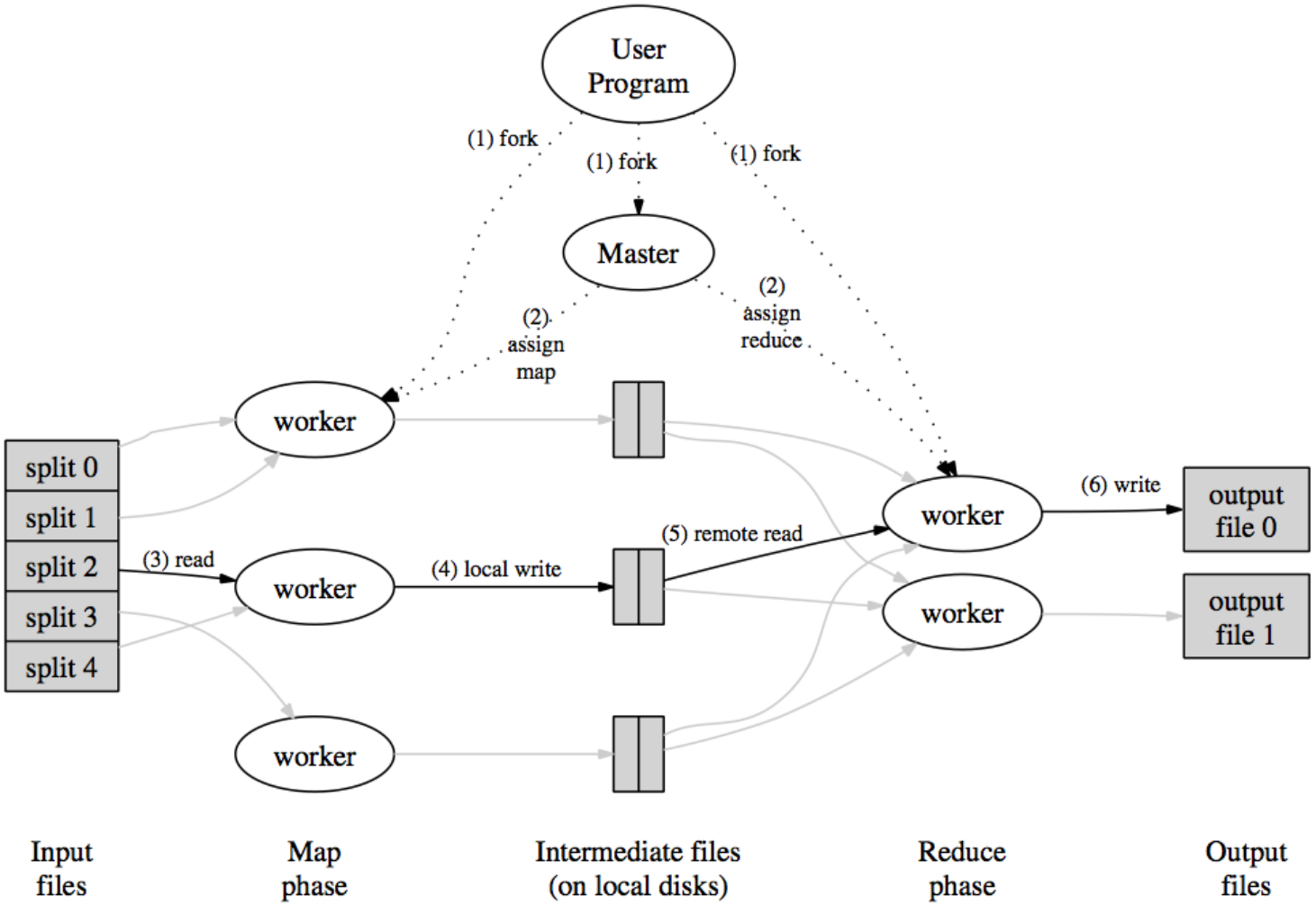
Hadoop Distributed File System (HDFS) es un sistema de archivos distribuido:

- Adecuado para aplicaciones que tienen grandes conjuntos de datos.
- Diseñado para ser implementado en el hardware de bajo costo.
- Brinda
  - Replicación de datos a través de múltiples datanodes
  - Alta tolerancia a fallos
  - Disponibilidad y alto rendimiento de acceso a los datos



- El *Network File System* es un protocolo que permite acceder a FS remotos como si fueran locales, utilizando RPC.
- La idea es que un FS remoto se monta en algún punto del sistema local y las aplicaciones acceden a archivos de ahí, sin saber que son remotos.
- Para poder soportar esto, los SO incorporan una capa llamada *Virtual File System*.
- Esta capa tiene *vnodes* por cada archivo abiertos. Se corresponden con inodos, si el archivo es local. Si es remoto, se almacena otra información.
- Así, los pedidos de E/S que llegan al VFS son despachados al FS real, o al *cliente de NFS*, que maneja el protocolo de red necesario.
- Si bien del lado del cliente es necesario un módulo de kernel, del lado del server alcanza con un programa común y corriente.

- Notar que, desde cierto punto de vista, NFS no es 100% distribuido, ya que todos los datos de un mismo "directorio" deben vivir físicamente en el mismo lugar.
- Otros FS distribuidos funcionan de manera similar (en cuanto a su integración con el kernel).
- Hay FS 100% distribuidos, como AFS o Coda.
- Han tenido un éxito relativo.


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
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

Una base de datos No-SQL que implementa:

- MapReduce.
- Replicación y alta disponibilidad.
- Autosharding, escalabilidad horizontal.

`http://docs.mongodb.org/manual/core/map-reduce`

## Sharding VS Vertical scaling

- Vertical scaling
  - Añade más recursos de la CPU y de almacenamiento para aumentar la capacidad.
  - Los sistemas de alto rendimiento son desproporcionadamente más caros que los sistemas más pequeños.
  - Hay una capacidad máxima práctica para el escalado vertical.
- Sharding o escalabilidad horizontal
  - Divide el conjunto de datos y distribuye los datos a través de múltiples servidores, o fragmentos.
  - Cada fragmento es una base de datos independiente, y colectivamente, los fragmentos forman una sola base de datos lógica.

- Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer University of Maryland, College Park
- Mining of Massive Datasets Anand Rajaraman Jure Leskovec Stanford Univ. Jeffrey D. Ullman
- Data-Intensive Text Processing with MapReduce Jimmy Lin and Chris Dyer University of Maryland, College Park
- The Little MongoDB Book Karl Seguin
- `http://docs.mongodb.org/manual/`
- `http://research.google.com/archive/mapreduce-osdi04-slides/index.html`
- `http://static.googleusercontent.com/media/research.google.com/es//archive/mapreduce-osdi04.pdf`
- `http://www.mongodb.com/dl/big-data`
- `http://info.mongodb.com/rs/mongodb/images/MongoDB-Performance-Considerations_2.4.pdf`
- `http://christophermeiklejohn.com/distributed/systems/2013/07/12/readings-in-distributed-systems.html`


LOONEY TUNES
Thats all Folks.