Google File System (GFS)

El sistema de archivos distribuido de Google !!

Mario A. del Riego

Sistemas Operativos - InCo
Facultad de Ingeniería de la
Universidad de la República

Agenda

Agenda

Agenda

Agenda

Solo unas definiciones

Definition

Un *Sistema Distribuido* es una colección de computadoras independientes que se muestran al usuario como un único sistema.

Definition

Un *Sistema de Archivos Distribudio (DFS)* es un servicio que permite al *usuario* acceder y procesar archivos guardados en un *servidor* como si fuese local.

Definition

Google File System (GFS) es el DFS que da soporte a *todas* las aplicaciones de Google Inc.

Solo unas definiciones

Definition

Un *Sistema Distribuido* es una colección de computadoras independientes que se muestran al usuario como un único sistema.

Definition

Un *Sistema de Archivos Distribudio (DFS)* es un servicio que permite al *usuario* acceder y procesar archivos guardados en un *servidor* como si fuese local.

Definition

Google File System (GFS) es el DFS que da soporte a *todas* las aplicaciones de Google Inc.

Solo unas definiciones

Definition

Un *Sistema Distribuido* es una colección de computadoras independientes que se muestran al usuario como un único sistema.

Definition

Un *Sistema de Archivos Distribudio (DFS)* es un servicio que permite al *usuario* acceder y procesar archivos guardados en un *servidor* como si fuese local.

Definition

Google File System (GFS) es el DFS que da soporte a *todas* las aplicaciones de Google Inc.

Objetivo de un DS: Transparencia

- Acceso: Ocultar representación de la info y cómo es accedido
- Ubicación: Ocultar donde se ubica el recurso
- Migración: Ocultar que el recurso puede moverse a otra ubicación
- ReUbicación: Ocultar que el recurso puede ser movido mientras esta en uso
- Replicación: Ocultar que el recuso es replicado
- Concurrencia: Ocultar que el recurso puede ser compartido por varios usuarios
- Falla: Ocultar las fallas y su recuperación
- Persistencia: Ocultar si un recurso esta en memoria o disco

Objetivo de un DS: Transparencia

- Acceso: Ocultar representación de la info y cómo es accedido
- Ubicación: Ocultar donde se ubica el recurso
- Migración: Ocultar que el recurso puede moverse a otra ubicación
- ReUbicación: Ocultar que el recurso puede ser movido mientras esta en uso
- Replicación: Ocultar que el recuso es replicado
- Concurrencia: Ocultar que el recurso puede ser compartido por varios usuarios
- Falla: Ocultar las fallas y su recuperación
- Persistencia: Ocultar si un recurso esta en memoria o disco

Objetivo de un DS: Transparencia

- Acceso: Ocultar representación de la info y cómo es accedido
- Ubicación: Ocultar donde se ubica el recurso
- Migración: Ocultar que el recurso puede moverse a otra ubicación
- ReUbicación: Ocultar que el recurso puede ser movido mientras esta en uso
- Replicación: Ocultar que el recuso es replicado
- Concurrencia: Ocultar que el recurso puede ser compartido por varios usuarios
- Falla: Ocultar las fallas y su recuperación
- Persistencia: Ocultar si un recurso esta en memoria o disco

Objetivo de un DS: Transparencia

- Acceso: Ocultar representación de la info y cómo es accedido
- Ubicación: Ocultar donde se ubica el recurso
- Migración: Ocultar que el recurso puede moverse a otra ubicación
- ReUbicación: Ocultar que el recurso puede ser movido mientras esta en uso
- Replicación: Ocultar que el recuso es replicado
- Concurrencia: Ocultar que el recurso puede ser compartido por varios usuarios
- Falla: Ocultar las fallas y su recuperación
- Persistencia: Ocultar si un recurso esta en memoria o disco

Objetivo de un DS: Transparencia

- Acceso: Ocultar representación de la info y cómo es accedido
- Ubicación: Ocultar donde se ubica el recurso
- Migración: Ocultar que el recurso puede moverse a otra ubicación
- ReUbicación: Ocultar que el recurso puede ser movido mientras esta en uso
- Replicación: Ocultar que el recuso es replicado
- Concurrencia: Ocultar que el recurso puede ser compartido por varios usuarios
- Falla: Ocultar las fallas y su recuperación
- Persistencia: Ocultar si un recurso esta en memoria o disco

Objetivo de un DS: Transparencia

- Acceso: Ocultar representación de la info y cómo es accedido
- Ubicación: Ocultar donde se ubica el recurso
- Migración: Ocultar que el recurso puede moverse a otra ubicación
- ReUbicación: Ocultar que el recurso puede ser movido mientras esta en uso
- Replicación: Ocultar que el recuso es replicado
- Concurrencia: Ocultar que el recurso puede ser compartido por varios usuarios
- Falla: Ocultar las fallas y su recuperación
- Persistencia: Ocultar si un recurso esta en memoria o disco

Objetivo de un DS: Transparencia

- Acceso: Ocultar representación de la info y cómo es accedido
- Ubicación: Ocultar donde se ubica el recurso
- Migración: Ocultar que el recurso puede moverse a otra ubicación
- ReUbicación: Ocultar que el recurso puede ser movido mientras esta en uso
- Replicación: Ocultar que el recuso es replicado
- Concurrencia: Ocultar que el recurso puede ser compartido por varios usuarios
- Falla: Ocultar las fallas y su recuperación
- Persistencia: Ocultar si un recurso esta en memoria o disco

Objetivo de un DS: Transparencia

- Acceso: Ocultar representación de la info y cómo es accedido
- Ubicación: Ocultar donde se ubica el recurso
- Migración: Ocultar que el recurso puede moverse a otra ubicación
- ReUbicación: Ocultar que el recurso puede ser movido mientras esta en uso
- Replicación: Ocultar que el recuso es replicado
- Concurrencia: Ocultar que el recurso puede ser compartido por varios usuarios
- Falla: Ocultar las fallas y su recuperación
- Persistencia: Ocultar si un recurso esta en memoria o disco

Las aplicaciones actuales de Google

- Google Mail (10GB)
- Google Search / History
- Google Video / Youtube
- Google Maps
- Google Street View
- Google Apps
- Google Docs
- Google Drive (**NUEVO!** 5GB por usuario)
- **Google ... ???**

Las aplicaciones actuales de Google

- Google Mail (10GB)
- Google Search / History
- Google Video / Youtube
- Google Maps
- Google Street View
- Google Apps
- Google Docs
- Google Drive (**NUEVO!** 5GB por usuario)
- **Google ... ???**

Las aplicaciones actuales de Google

- Google Mail (10GB)
- Google Search / History
- Google Video / Youtube
- Google Maps
- Google Street View
- Google Apps
- Google Docs
- Google Drive (**NUEVO!** 5GB por usuario)
- **Google ... ???**

Las aplicaciones actuales de Google

- Google Mail (10GB)
- Google Search / History
- Google Video / Youtube
- Google Maps
- Google Street View
- Google Apps
- Google Docs
- Google Drive (**NUEVO!** 5GB por usuario)
- **Google ... ???**

Las aplicaciones actuales de Google

- Google Mail (10GB)
- Google Search / History
- Google Video / Youtube
- Google Maps
- Google Street View
- Google Apps
- Google Docs
- Google Drive (**NUEVO!** 5GB por usuario)
- **Google ... ???**

Las aplicaciones actuales de Google

- Google Mail (10GB)
- Google Search / History
- Google Video / Youtube
- Google Maps
- Google Street View
- Google Apps
- Google Docs
- Google Drive (**NUEVO!** 5GB por usuario)
- **Google ... ???**

Las aplicaciones actuales de Google

- Google Mail (10GB)
- Google Search / History
- Google Video / Youtube
- Google Maps
- Google Street View
- Google Apps
- Google Docs
- Google Drive (**NUEVO!** 5GB por usuario)
- **Google ... ???**

Las aplicaciones actuales de Google

- Google Mail (10GB)
- Google Search / History
- Google Video / Youtube
- Google Maps
- Google Street View
- Google Apps
- Google Docs
- Google Drive (**NUEVO!** 5GB por usuario)
- Google ... ???

Las aplicaciones actuales de Google

- Google Mail (10GB)
- Google Search / History
- Google Video / Youtube
- Google Maps
- Google Street View
- Google Apps
- Google Docs
- Google Drive (**NUEVO!** 5GB por usuario)
- **Google ... ???**

. . . porqué no centralizado?

- La historia me avala!

. . . porqué no distribuido (NFS)?

- ...otra vez, la historia me avala!

. . . y porqué no empezamos por el principio?

- Análisis de requerimientos.. de las aplicaciones!
- Muy pocos estudios científicos al respecto

. . . y porqué no empezamos por el principio?

- Análisis de requerimientos.. de las aplicaciones!
- Muy pocos estudios científicos al respecto

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Basado en la experiencia...

- Los componentes *fallan*, es la norma y no la excepción
  - Errores en la aplicación
  - Errores en el SO
  - Errores humanos (errar es humano... dijo el caballo)
  - Errores en los HDD / RAM
  - Errores en la red
  - Cortes de energía
  - Inundaciones...
- Los archivos crecen muy rápido, llegando al orden de TB (!)
- La mayoría de los archivos son modificados *agregando* info. Modificaciones aleatorias dentro del archivo ni existen.

Hipótesis

- Es construído en base a componentes *baratos*
- Se debe manejar eficientemente archivos muy grandes (GB)
- Lecturas: Dos operaciones típicas.
  - *Lecturas largas y continuas* (Por ej?)
  - *Lecturas aleatorias y chicas* (Por ej?)
- Escrituras: *append* a los archivos (Por ej?)
- Debe soportar eficientemente la *concurrencia*
- Mucho ancho de banda

Hipótesis

- Es construído en base a componentes *baratos*
- Se debe manejar eficientemente archivos muy grandes (GB)
- Lecturas: Dos operaciones típicas.
  - *Lecturas largas y continuas* (Por ej?)
  - *Lecturas aleatorias y chicas* (Por ej?)
- Escrituras: *append* a los archivos (Por ej?)
- Debe soportar eficientemente la *concurrencia*
- Mucho ancho de banda

Hipótesis

- Es construído en base a componentes *baratos*
- Se debe manejar eficientemente archivos muy grandes (GB)
- Lecturas: Dos operaciones típicas.
  - *Lecturas largas y continuas* (Por ej?)
  - *Lecturas aleatorias y chicas* (Por ej?)
- Escrituras: *append* a los archivos (Por ej?)
- Debe soportar eficientemente la *concurrencia*
- Mucho ancho de banda

Hipótesis

- Es construído en base a componentes *baratos*
- Se debe manejar eficientemente archivos muy grandes (GB)
- Lecturas: Dos operaciones típicas.
  - *Lecturas largas y continuas* (Por ej?)
  - *Lecturas aleatorias y chicas* (Por ej?)
- Escrituras: *append* a los archivos (Por ej?)
- Debe soportar eficientemente la *concurrencia*
- Mucho ancho de banda

Hipótesis

- Es construído en base a componentes *baratos*
- Se debe manejar eficientemente archivos muy grandes (GB)
- Lecturas: Dos operaciones típicas.
  - *Lecturas largas y continuas* (Por ej?)
  - *Lecturas aleatorias y chicas* (Por ej?)
- Escrituras: *append* a los archivos (Por ej?)
- Debe soportar eficientemente la *concurrencia*
- Mucho ancho de banda

Hipótesis

- Es construído en base a componentes *baratos*
- Se debe manejar eficientemente archivos muy grandes (GB)
- Lecturas: Dos operaciones típicas.
  - *Lecturas largas y continuas* (Por ej?)
  - *Lecturas aleatorias y chicas* (Por ej?)
- Escrituras: *append* a los archivos (Por ej?)
- Debe soportar eficientemente la *concurrencia*
- Mucho ancho de banda

Hipótesis

- Es construído en base a componentes *baratos*
- Se debe manejar eficientemente archivos muy grandes (GB)
- Lecturas: Dos operaciones típicas.
  - *Lecturas largas y continuas* (Por ej?)
  - *Lecturas aleatorias y chicas* (Por ej?)
- Escrituras: *append* a los archivos (Por ej?)
- Debe soportar eficientemente la *concurrencia*
- Mucho ancho de banda

Hipótesis

- Es construído en base a componentes *baratos*
- Se debe manejar eficientemente archivos muy grandes (GB)
- Lecturas: Dos operaciones típicas.
  - *Lecturas largas y continuas* (Por ej?)
  - *Lecturas aleatorias y chicas* (Por ej?)
- Escrituras: *append* a los archivos (Por ej?)
- Debe soportar eficientemente la *concurrencia*
- Mucho ancho de banda

Interfase

- Agrega dos operaciones importantes: *snapshot* y *append*
  - *Snapshot*: Una "foto"del sistema en un momento determinado
- No soporta *POSIX* (..pero muy parecido)
- ..hay que **recompilar todo**!

Interfase

- Agrega dos operaciones importantes: *snapshot* y *append*
  - *Snapshot*: Una "foto"del sistema en un momento determinado
- No soporta *POSIX* (..pero muy parecido)
- ..hay que **recompilar todo**!

Interfase

- Agrega dos operaciones importantes: *snapshot* y *append*
  - *Snapshot*: Una "foto"del sistema en un momento determinado
- No soporta *POSIX* (..pero muy parecido)
- ..hay que **recompilar todo**!

Interfase

- Agrega dos operaciones importantes: *snapshot* y *append*
  - *Snapshot*: Una "foto"del sistema en un momento determinado
- No soporta *POSIX* (..pero muy parecido)
- ..hay que **recompilar todo**!

Arquitectura (cont.)

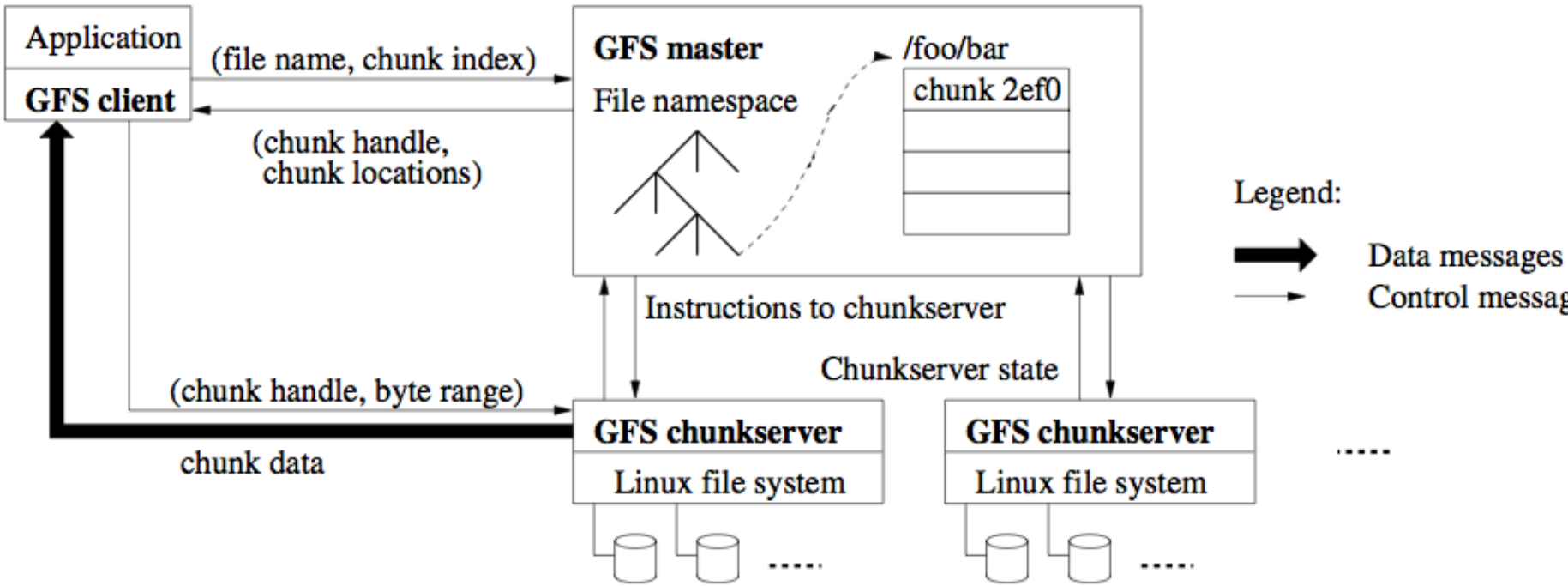

Figure 1: GFS Architecture

Arquitectura

- 1 master
- N chunkserver
- M clientes
- Archivos divididos en un tamaño **fijo**
  - Cada "pedazo" se llama *chunk*
  - Cada *chunk* es identificado por un número único de 64 bits, llamado *chunk handle*
  - 1 archivo se divide en $\frac{size(archivo)}{64MB}$ chunks
  - $1 chunk \rightarrow \{Location_1, \ldots, Location_N\}$

Arquitectura

- 1 master
- N chunkserver
- M clientes
- Archivos divididos en un tamaño **fijo**
  - Cada "pedazo" se llama *chunk*
  - Cada *chunk* es identificado por un número único de 64 bits, llamado *chunk handle*
  - 1 archivo se divide en $\frac{size(archivo)}{64MB}$ chunks
  - $1 chunk \rightarrow \{Location_1, \ldots, Location_N\}$

Arquitectura

- 1 master
- N chunkserver
- M clientes
- Archivos divididos en un tamaño **fijo**
  - Cada "pedazo" se llama *chunk*
  - Cada *chunk* es identificado por un número único de 64 bits, llamado *chunk handle*
  - 1 archivo se divide en $\frac{size(archivo)}{64MB}$ chunks
  - $1 chunk \rightarrow \{Location_1, \ldots, Location_N\}$

Arquitectura

- 1 master
- N chunkserver
- M clientes
- Archivos divididos en un tamaño **fijo**
  - Cada “pedazo” se llama *chunk*
  - Cada *chunk* es identificado por un número único de 64 bits, llamado *chunk handle*
  - 1 archivo se divide en $\frac{size(archivo)}{64MB}$ chunks
  - $1 chunk \rightarrow \{Location_1, \ldots, Location_N\}$

Arquitectura

- 1 master
- N chunkserver
- M clientes
- Archivos divididos en un tamaño **fijo**
  - Cada “pedazo” se llama *chunk*
  - Cada *chunk* es identificado por un número único de 64 bits, llamado *chunk handle*
  - 1 archivo se divide en $\frac{size(archivo)}{64MB}$ chunks
  - $1 chunk \rightarrow \{Location_1, \ldots, Location_N\}$

Arquitectura

- 1 master
- N chunkserver
- M clientes
- Archivos divididos en un tamaño **fijo**
  - Cada “pedazo” se llama *chunk*
  - Cada *chunk* es identificado por un número único de 64 bits, llamado *chunk handle*
  - 1 archivo se divide en $\frac{size(archivo)}{64MB}$ chunks
  - $1 chunk \rightarrow \{Location_1, \ldots, Location_N\}$

Arquitectura

- 1 master
- N chunkserver
- M clientes
- Archivos divididos en un tamaño **fijo**
  - Cada “pedazo” se llama *chunk*
  - Cada *chunk* es identificado por un número único de 64 bits, llamado *chunk handle*
  - 1 archivo se divide en $\frac{size(archivo)}{64MB}$ chunks
  - $1 chunk \rightarrow \{Location_1, \ldots, Location_N\}$

Arquitectura

- 1 master
- N chunkserver
- M clientes
- Archivos divididos en un tamaño **fijo**
  - Cada “pedazo” se llama *chunk*
  - Cada *chunk* es identificado por un número único de 64 bits, llamado *chunk handle*
  - 1 archivo se divide en $\frac{size(archivo)}{64MB}$ chunks
  - $1 chunk \rightarrow \{Location_1, \ldots, Location_N\}$

Master Server

- Mantiene la Metadata (control de acceso, mapeo de archivos a chunks, chunks a locations, etc)
- Gestiona los *chunk leases*, Garbage Collector, migraciones entre chunkservers, y monitorea el estado de cada chunkserver
- Un master simplifica el diseño y la implementación
- NO se envía datos a través de él
- Existe un *shadow server*. . .

Master Server

- Mantiene la Metadata (control de acceso, mapeo de archivos a chunks, chunks a locations, etc)
- Gestiona los *chunk leases*, Garbage Collector, migraciones entre chunkservers, y monitorea el estado de cada chunkserver
- Un master simplifica el diseño y la implementación
- NO se envía datos a través de él
- Existe un *shadow server*. . .

Master Server

- Mantiene la Metadata (control de acceso, mapeo de archivos a chunks, chunks a locations, etc)
- Gestiona los *chunk leases*, Garbage Collector, migraciones entre chunkservers, y monitorea el estado de cada chunkserver
- Un master simplifica el diseño y la implementación
- NO se envía datos a través de él
- Existe un *shadow server*. . .

Master Server

- Mantiene la Metadata (control de acceso, mapeo de archivos a chunks, chunks a locations, etc)
- Gestiona los *chunk leases*, Garbage Collector, migraciones entre chunkservers, y monitorea el estado de cada chunkserver
- Un master simplifica el diseño y la implementación
- NO se envía datos a través de él
- Existe un *shadow server*. . .

Master Server

- Mantiene la Metadata (control de acceso, mapeo de archivos a chunks, chunks a locations, etc)
- Gestiona los *chunk leases*, Garbage Collector, migraciones entre chunkservers, y monitorea el estado de cada chunkserver
- Un master simplifica el diseño y la implementación
- NO se envía datos a través de él
- Existe un *shadow server*. . .

Cliente y Chunk Server

- Cliente:
  - No utiliza cache para los datos. Solamente para los *chunk locations*
- Chunk Server:
  - Linux *standard*. En su tiempo kernel 2.2. . .
  - Todo implementado en *User space*
  - No implementa cache. Delegado al cache del kernel

Cliente y Chunk Server

- Cliente:
  - No utiliza cache para los datos. Solamente para los *chunk locations*
- Chunk Server:
  - Linux *standard*. En su tiempo kernel 2.2. . .
  - Todo implementado en *User space*
  - No implementa cache. Delegado al cache del kernel

Cliente y Chunk Server

- Cliente:
  - No utiliza cache para los datos. Solamente para los *chunk locations*
- Chunk Server:
  - Linux *standard*. En su tiempo kernel 2.2. . .
  - Todo implementado en *User space*
  - No implementa cache. Delegado al cache del kernel

Cliente y Chunk Server

- Cliente:
  - No utiliza cache para los datos. Solamente para los *chunk locations*
- Chunk Server:
  - Linux *standard*. En su tiempo kernel 2.2. . .
  - Todo implementado en *User space*
  - No implementa cache. Delegado al cache del kernel

Cliente y Chunk Server

- Cliente:
  - No utiliza cache para los datos. Solamente para los *chunk locations*
- Chunk Server:
  - Linux *standard*. En su tiempo kernel 2.2. . .
  - Todo implementado en *User space*
  - No implementa cache. Delegado al cache del kernel

Cliente y Chunk Server

- Cliente:
  - No utiliza cache para los datos. Solamente para los *chunk locations*
- Chunk Server:
  - Linux *standard*. En su tiempo kernel 2.2. . .
  - Todo implementado en *User space*
  - No implementa cache. Delegado al cache del kernel

Snapshot

- (← VER PIZARRÓN)

Ejemplo: Write

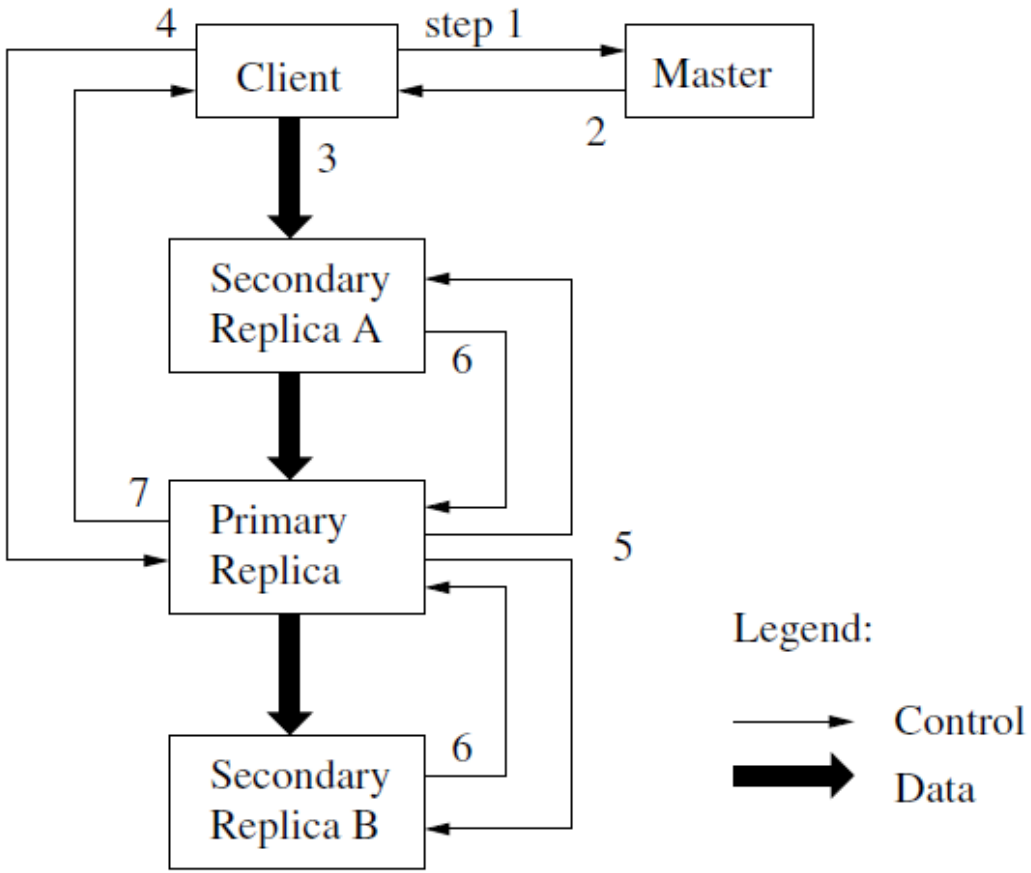

Figure 2: Write Control and Data Flow

Fin..g

- Preguntas?