Procesos distribuidos
Diseño de sistemas distribuidos
Caso de estudio: Google

Profesor: Diego Villalba

Escuela de Ciencias de la Computación e Informática
Universidad de Costa Rica

2014-01-15

Outline

Definición de *sistemas distribuidos*

Un sistema distribuido es un conjunto de componentes (hardware y software), localizados en computadoras independientes, que se presenta a sus usuarios como un solo sistema coherente.

Definición de *sistemas distribuidos*

Un sistema distribuido es un conjunto de componentes (hardware y software), localizados en computadoras independientes, que se presenta a sus usuarios como un solo sistema coherente.

Definición de *sistemas distribuidos*

Un sistema distribuido es un conjunto de componentes (hardware y software), localizados en computadoras independientes, que se presenta a sus usuarios como un solo sistema coherente.

Definición de *sistemas distribuidos*

Un sistema distribuido es un conjunto de componentes (hardware y software), localizados en computadoras independientes, que se presenta a sus usuarios como un solo sistema coherente.

Definición de *sistemas distribuidos*

Un sistema distribuido es un conjunto de componentes (hardware y software), localizados en computadoras independientes, que se presenta a sus usuarios como un solo sistema coherente.

Definición de *sistemas distribuidos*

Un sistema distribuido es un conjunto de componentes (hardware y software), localizados en computadoras independientes, que se presenta a sus usuarios como un solo sistema coherente.

Definición de *sistemas distribuidos*

Un sistema distribuido es un conjunto de componentes (hardware y software), localizados en computadoras independientes, que se presenta a sus usuarios como un solo sistema coherente.

Definición de *sistemas distribuidos*

Dos aspectos fundamentales sobresalen de esta definición:

Definición de *sistemas distribuidos*

Dos aspectos fundamentales sobresalen de esta definición:

- Un sistema distribuido consiste de componentes (i.e., computadoras o aplicaciones) que son autónomos.

Definición de *sistemas distribuidos*

Dos aspectos fundamentales sobresalen de esta definición:

- Un sistema distribuido consiste de componentes (i.e., computadoras o aplicaciones) que son autónomos.
- Sus usuarios (sean éstos personas o programas) piensan que están interactuando con un único sistema, es decir, el aspecto distribuido del sistema es transparente para el usuario.

Definición de *sistemas distribuidos*

- De lo anterior se deduce que, de una forma u otra, los componentes autónomos necesitan comunicarse y colaborar entre ellos a través del intercambio de mensajes.

Definición de *sistemas distribuidos*

- De lo anterior se deduce que, de una forma u otra, los componentes autónomos necesitan comunicarse y colaborar entre ellos a través del intercambio de mensajes.
- Cómo establecer esta colaboración es uno de los aspectos básicos del desarrollo de sistemas distribuidos.

Definición de *sistemas distribuidos*

- De lo anterior se deduce que, de una forma u otra, los componentes autónomos necesitan comunicarse y colaborar entre ellos a través del intercambio de mensajes.
- Cómo establecer esta colaboración es uno de los aspectos básicos del desarrollo de sistemas distribuidos.
- Nótese que no se hicieron suposiciones sobre el tipo de computadoras que lo componen; podría tratarse de una colección de supercomputadoras de alto rendimiento, de un grupo de servidores e innumerables máquinas clientes o dispositivos móviles, o de una red de sensores.

Definición de *sistemas distribuidos*

- De lo anterior se deduce que, de una forma u otra, los componentes autónomos necesitan comunicarse y colaborar entre ellos a través del intercambio de mensajes.
- Cómo establecer esta colaboración es uno de los aspectos básicos del desarrollo de sistemas distribuidos.
- Nótese que no se hicieron suposiciones sobre el tipo de computadoras que lo componen; podría tratarse de una colección de supercomputadoras de alto rendimiento, de un grupo de servidores e innumerables máquinas clientes o dispositivos móviles, o de una red de sensores.
- Tampoco se hicieron suposiciones sobre la forma en que estos componentes están interconectados.

Diseño de sistemas distribuidos

- La capacidad de crear diseños efectivos es una habilidad muy importante en sistemas distribuidos.

Diseño de sistemas distribuidos

- La capacidad de crear diseños efectivos es una habilidad muy importante en sistemas distribuidos.
- Se requiere conocimiento de diferentes alternativas tecnológicas y de los requerimientos relevantes del dominio de aplicación.

Diseño de sistemas distribuidos

- La capacidad de crear diseños efectivos es una habilidad muy importante en sistemas distribuidos.
- Se requiere conocimiento de diferentes alternativas tecnológicas y de los requerimientos relevantes del dominio de aplicación.
- El objetivo es crear una arquitectura consistente que incorpore un conjunto completo de recursos que contemplen los requerimientos globales del sistema.

Conceptos claves de diseño

- Heterogeneidad

Conceptos claves de diseño

- Heterogeneidad
- Apertura

Conceptos claves de diseño

- Heterogeneidad
- Apertura
- Seguridad

Conceptos claves de diseño

- Heterogeneidad
- Apertura
- Seguridad
- Escalabilidad

Conceptos claves de diseño

- Heterogeneidad
- Apertura
- Seguridad
- Escalabilidad
- Manejo de fallos

Conceptos claves de diseño

- Heterogeneidad
- Apertura
- Seguridad
- Escalabilidad
- Manejo de fallos
- Concurrencia

Conceptos claves de diseño

- Heterogeneidad
- Apertura
- Seguridad
- Escalabilidad
- Manejo de fallos
- Concurrencia
- Transparencia

Conceptos claves de diseño

- Heterogeneidad
- Apertura
- Seguridad
- Escalabilidad
- Manejo de fallos
- Concurrencia
- Transparencia
- Calidad de servicio

Partes que constituyen un sistema distribuido

- Paradigmas de comunicación como invocación remota y alternativas indirectas.

Partes que constituyen un sistema distribuido

- Paradigmas de comunicación como invocación remota y alternativas indirectas.
- Abstracciones de programación ofrecidas por objetos, componentes o servicios web.

Partes que constituyen un sistema distribuido

- Paradigmas de comunicación como invocación remota y alternativas indirectas.
- Abstracciones de programación ofrecidas por objetos, componentes o servicios web.
- Servicios específicos de seguridad, resolución de nombres o soporte de sistemas de archivos.

Partes que constituyen un sistema distribuido

- Paradigmas de comunicación como invocación remota y alternativas indirectas.
- Abstracciones de programación ofrecidas por objetos, componentes o servicios web.
- Servicios específicos de seguridad, resolución de nombres o soporte de sistemas de archivos.
- Soluciones algorítmicas para la coordinación y el acuerdo.

Diseño global

Sin embargo...

Diseño global

Sin embargo...

- También es importante considerar cómo ensamblar las partes

Diseño global

Sin embargo...

- También es importante considerar cómo ensamblar las partes para obtener una arquitectura global completa

Diseño global

Sin embargo...

- También es importante considerar cómo ensamblar las partes para obtener una arquitectura global completa
  que cumpla con los requerimientos en un ambiente o dominio a gran escala.

Google

- Sus oficinas centrales están localizadas en Mountain View, California (el Googleplex).

Google

- Sus oficinas centrales están localizadas en Mountain View, California (el Googleplex).
- Ofrece servicios de búsqueda en la web, entre otros servicios.

Google

- Sus oficinas centrales están localizadas en Mountain View, California (el Googleplex).
- Ofrece servicios de búsqueda en la web, entre otros servicios.
- Su principal fuente de ingreso son anuncios asociados a su servicio de búsqueda.

Google

- Sus oficinas centrales están localizadas en Mountain View, California (el Googleplex).
- Ofrece servicios de búsqueda en la web, entre otros servicios.
- Su principal fuente de ingreso son anuncios asociados a su servicio de búsqueda.
- Su nombre es una variación de la palabra *googol* (10^{100}).

Google

- Sus oficinas centrales están localizadas en Mountain View, California (el Googleplex).
- Ofrece servicios de búsqueda en la web, entre otros servicios.
- Su principal fuente de ingreso son anuncios asociados a su servicio de búsqueda.
- Su nombre es una variación de la palabra *googol* (10^{100}).
- Su misión es "organizar la información del mundo y hacerla útil y accesible universalmente".

Google

- Sus oficinas centrales están localizadas en Mountain View, California (el Googleplex).
- Ofrece servicios de búsqueda en la web, entre otros servicios.
- Su principal fuente de ingreso son anuncios asociados a su servicio de búsqueda.
- Su nombre es una variación de la palabra *googol* (10^{100}).
- Su misión es "organizar la información del mundo y hacerla útil y accesible universalmente".
- Se originó a partir de un proyecto de investigación de Stanford y nació como empresa en 1998.

Google

- Sus oficinas centrales están localizadas en Mountain View, California (el Googleplex).
- Ofrece servicios de búsqueda en la web, entre otros servicios.
- Su principal fuente de ingreso son anuncios asociados a su servicio de búsqueda.
- Su nombre es una variación de la palabra *googol* (10^{100}).
- Su misión es "organizar la información del mundo y hacerla útil y accesible universalmente".
- Se originó a partir de un proyecto de investigación de Stanford y nació como empresa en 1998.
- Desde entonces, ha dominado el mercado de los motores de búsqueda gracias, principalmente, a su algoritmo de ranqueo de resultados.

Googleplex

El desafío de Google

Desde la perspectiva de los sitemas distribuidos, Google provee un estudio de caso bastante completo, con requerimientos extremadamente exigentes, particularmente en términos de:

El desafío de Google

Desde la perspectiva de los sitemas distribuidos, Google provee un estudio de caso bastante completo, con requerimientos extremadamente exigentes, particularmente en términos de:

- Escalabilidad

El desafío de Google

Desde la perspectiva de los sitemas distribuidos, Google provee un estudio de caso bastante completo, con requerimientos extremadamente exigentes, particularmente en términos de:

- Escalabilidad
- Confiabilidad

El desafío de Google

Desde la perspectiva de los sitemas distribuidos, Google provee un estudio de caso bastante completo, con requerimientos extremadamente exigentes, particularmente en términos de:

- Escalabilidad
- Confiabilidad
- Rendimiento

El desafío de Google

Desde la perspectiva de los sitemas distribuidos, Google provee un estudio de caso bastante completo, con requerimientos extremadamente exigentes, particularmente en términos de:

- Escalabilidad
- Confiabilidad
- Rendimiento
- Apertura

El desafío de Google

Desde la perspectiva de los sitemas distribuidos, Google provee un estudio de caso bastante completo, con requerimientos extremadamente exigentes, particularmente en términos de:

- Escalabilidad
- Confiabilidad
- Rendimiento
- Apertura

Vamos a examinar las estrategias y decisiones de diseño más relevantes, tanto para su motor de búsqueda como para otros servicios que brinda, como un ejemplo de un sistema distribuido contemporáneo complejo.

Google Search Engine

- Crawling

Google Search Engine

- Crawling
- Indexing

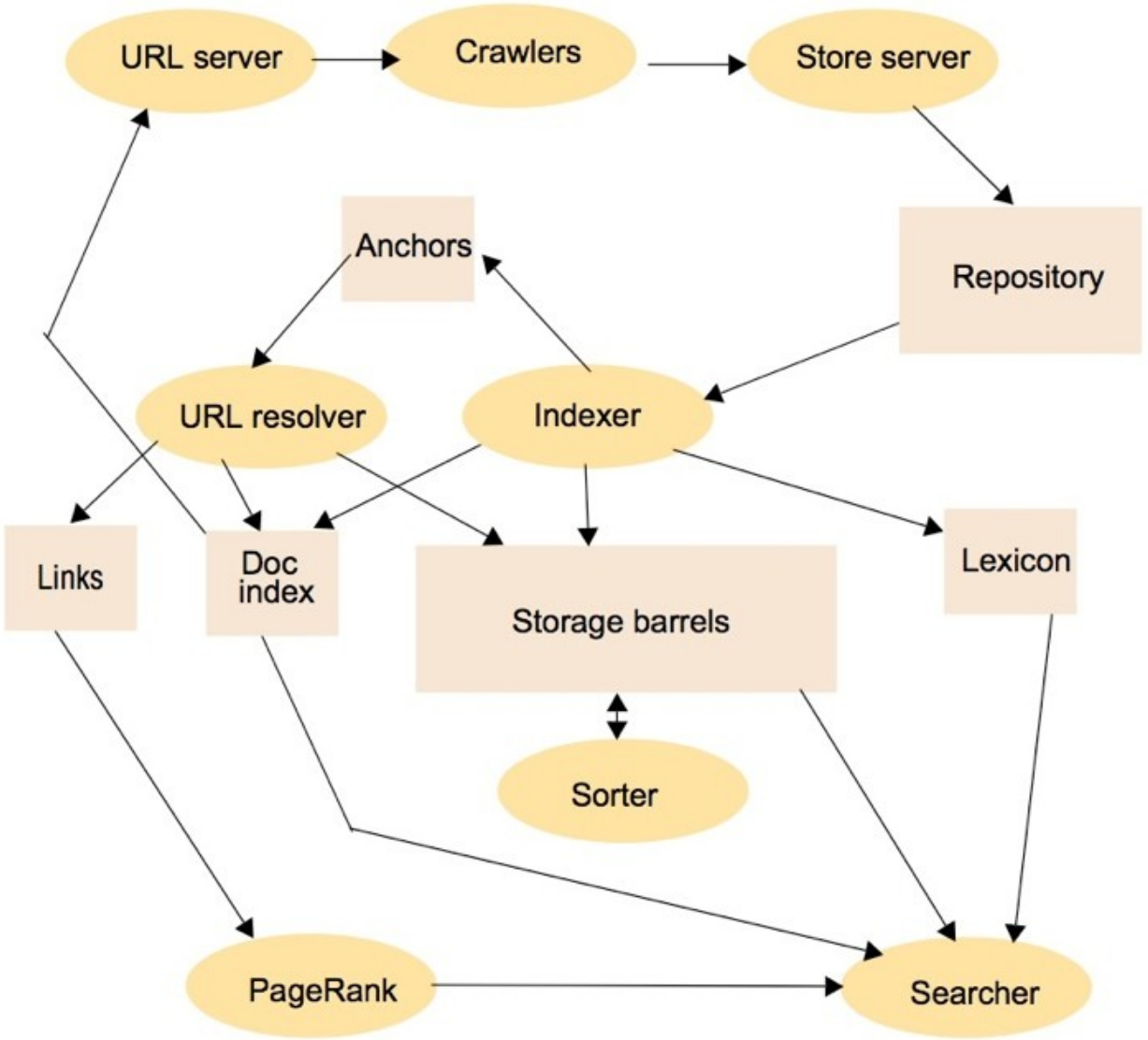
Google Search Engine

- Crawling
- Indexing
- Ranking

Google Search Engine

- Crawling
- Indexing
- Ranking
- GUI

Arquitectura del sistema original



Google como proveedor de servicios en la nube

- Software as a service

Google como proveedor de servicios en la nube

- Software as a service
- Platform as a service

Ejemplos de aplicaciones de Google

Application	*Description*
Gmail	Mail system with messages hosted by Google but desktop-like message management.
Google Docs	Web-based office suite supporting shared editing of documents held on Google servers.
Google Sites	Wiki-like web sites with shared editing facilities.
Google Talk	Supports instant text messaging and Voice over IP.
Google Calendar	Web-based calendar with all data hosted on Google servers.
Google Wave	Collaboration tool integrating email, instant messaging, wikis and social networks.
Google News	Fully automated news aggregator site.
Google Maps	Scalable web-based world map including high-resolution imagery and unlimited user-generated overlays.
Google Earth	Scalable near-3D view of the globe with unlimited user-generated overlays.
Google App Engine	Google distributed infrastructure made available to outside parties as a service (platform as a service).

Outline

Filosofía de diseño

- Utilizar grandes cantidades de servidores de bajo costo para producir un ambiente costo-efectivo para la computación y el almacenamiento distribuido.

Filosofía de diseño

- Utilizar grandes cantidades de servidores de bajo costo para producir un ambiente costo-efectivo para la computación y el almacenamiento distribuido.
- Datos para 2006: unidades de 2 TB de disco duro, 16 GB de RAM y con una versión personalizada de Linux (aproximadamente 1.000.- dólares cada una).

Filosofía de diseño

- Utilizar grandes cantidades de servidores de bajo costo para producir un ambiente costo-efectivo para la computación y el almacenamiento distribuido.
- Datos para 2006: unidades de 2 TB de disco duro, 16 GB de RAM y con una versión personalizada de Linux (aproximadamente 1.000.- dólares cada una).
- Se reconoce que parte de la infraestructura va a fallar, por lo que se crean estrategias para tolerar esas fallas.

Filosofía de diseño

- Utilizar grandes cantidades de servidores de bajo costo para producir un ambiente costo-efectivo para la computación y el almacenamiento distribuido.
- Datos para 2006: unidades de 2 TB de disco duro, 16 GB de RAM y con una versión personalizada de Linux (aproximadamente 1.000.- dólares cada una).
- Se reconoce que parte de la infraestructura va a fallar, por lo que se crean estrategias para tolerar esas fallas.
- La fallas más comunes son debidas al software, con 20 máquinas que necesitan reiniciarse por día.

Filosofía de diseño

- Utilizar grandes cantidades de servidores de bajo costo para producir un ambiente costo-efectivo para la computación y el almacenamiento distribuido.
- Datos para 2006: unidades de 2 TB de disco duro, 16 GB de RAM y con una versión personalizada de Linux (aproximadamente 1.000.- dólares cada una).
- Se reconoce que parte de la infraestructura va a fallar, por lo que se crean estrategias para tolerar esas fallas.
- La fallas más comunes son debidas al software, con 20 máquinas que necesitan reiniciarse por día.
- Las fallas de hardware se dan a razón de una por cada 10 de las debidas al software.

Filosofía de diseño

- Este es el razonamiento para utilizar hardware de bajo costo: si la mayoría de las fallas son de software, no vale la pena invertir en harware más confiable.

Filosofía de diseño

- Este es el razonamiento para utilizar hardware de bajo costo: si la mayoría de las fallas son de software, no vale la pena invertir en harware más confiable.
- Los servidores de bajo costo están organizadas en *racks* de 40 a 80 unidades.

Filosofía de diseño

- Este es el razonamiento para utilizar hardware de bajo costo: si la mayoría de las fallas son de software, no vale la pena invertir en harware más confiable.
- Los servidores de bajo costo están organizadas en *racks* de 40 a 80 unidades.
- Cada rack cuenta con un *switch* de Ethernet que provee conectividad dentro del rack y hacia afuera.

Filosofía de diseño

- Este es el razonamiento para utilizar hardware de bajo costo: si la mayoría de las fallas son de software, no vale la pena invertir en harware más confiable.
- Los servidores de bajo costo están organizadas en *racks* de 40 a 80 unidades.
- Cada rack cuenta con un *switch* de Ethernet que provee conectividad dentro del rack y hacia afuera.
- Las conexiones internas utilizan puertos de 100 Mbps y las conexiones externas son de 1 Gbps.

Filosofía de diseño

- Los racks se organizan en *clusters*, que son las unidades básicas de administración.

Filosofía de diseño

- Los racks se organizan en *clusters*, que son las unidades básicas de administración.
- Los clusters determinan, por ejemplo, la ubicación y la replicación de los servicios.

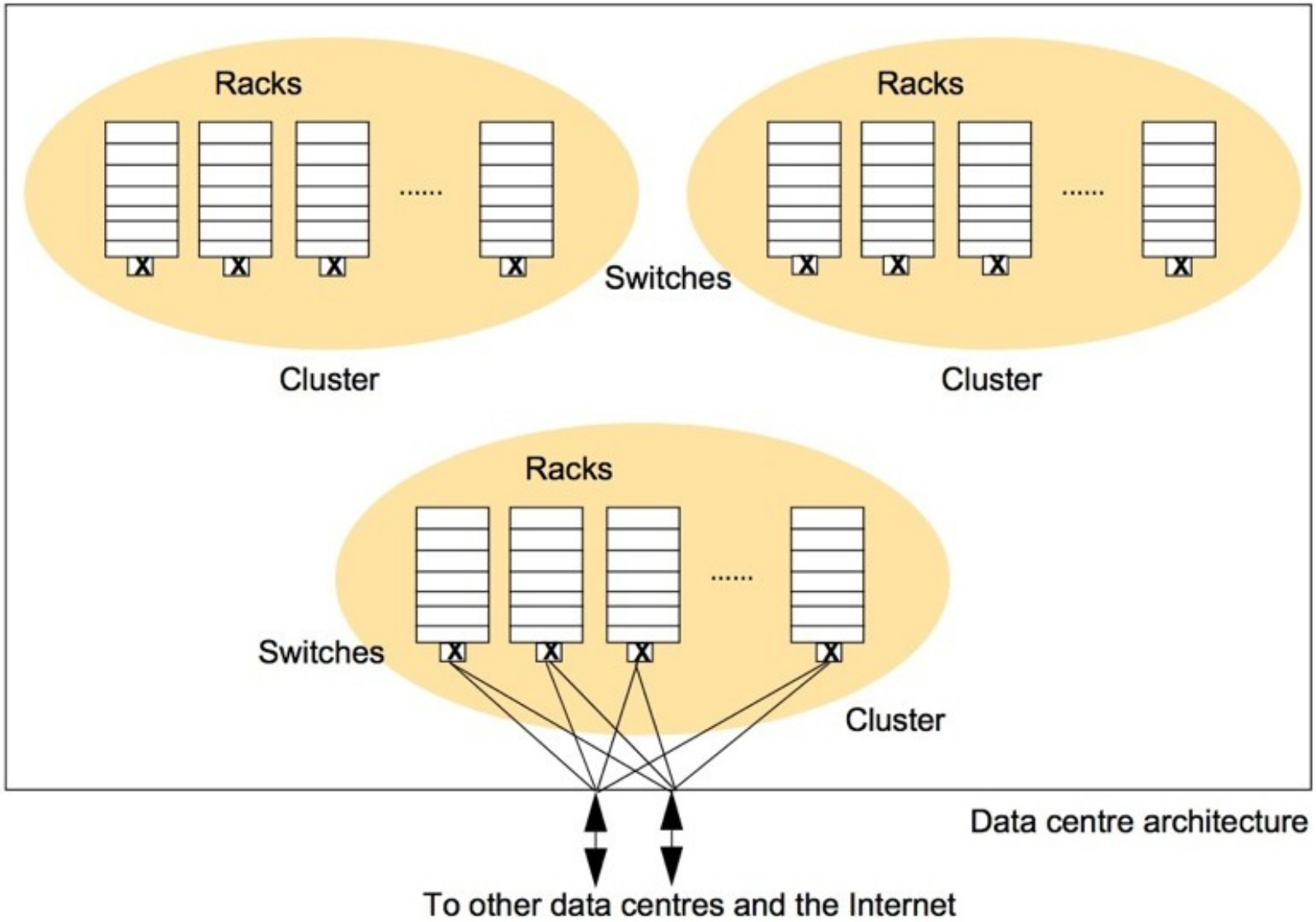
Filosofía de diseño

- Los racks se organizan en *clusters*, que son las unidades básicas de administración.
- Los clusters determinan, por ejemplo, la ubicación y la replicación de los servicios.
- Un cluster consiste típicamente de 30 o más racks y de dos switches de banda ancha que lo conectan al mundo exterior (la Internet y el resto de los centros de datos de Google).

Filosofía de diseño

- Los racks se organizan en *clusters*, que son las unidades básicas de administración.
- Los clusters determinan, por ejemplo, la ubicación y la replicación de los servicios.
- Un cluster consiste típicamente de 30 o más racks y de dos switches de banda ancha que lo conectan al mundo exterior (la Internet y el resto de los centros de datos de Google).
- Cada rack se conecta a los dos switches por redundancia (además, cada switch tiene conexiones redundantes con el mundo exterior).

Organización de la infraestructura física



Filosofía de diseño

- Los clusters se localizan en los centros de datos de Google repartidos alrededor del mundo.

Filosofía de diseño

- Los clusters se localizan en los centros de datos de Google repartidos alrededor del mundo.
- En el 2000, Google tenía dos centros de datos en Silicon Valley y uno en Virginia.

Filosofía de diseño

- Los clusters se localizan en los centros de datos de Google repartidos alrededor del mundo.
- En el 2000, Google tenía dos centros de datos en Silicon Valley y uno en Virginia.
- En 2008, tenía otros centros de datos en EE.UU. y también en Irlanda, Bélgica, Suiza, Japón y China.

Filosofía de diseño

- Los clusters se localizan en los centros de datos de Google repartidos alrededor del mundo.
- En el 2000, Google tenía dos centros de datos en Silicon Valley y uno en Virginia.
- En 2008, tenía otros centros de datos en EE.UU. y también en Irlanda, Bélgica, Suiza, Japón y China.
- Información más actualizada muchas veces se obtiene de comunicados de prensa de la compañía y no siempre es completa.

Capacidad de almacenamiento

- Cada servidor ofrece capacidad para 2 TB (año 2006).

Capacidad de almacenamiento

- Cada servidor ofrece capacidad para 2 TB (año 2006).
- Un rack de 80 servidores provee 160 TB.

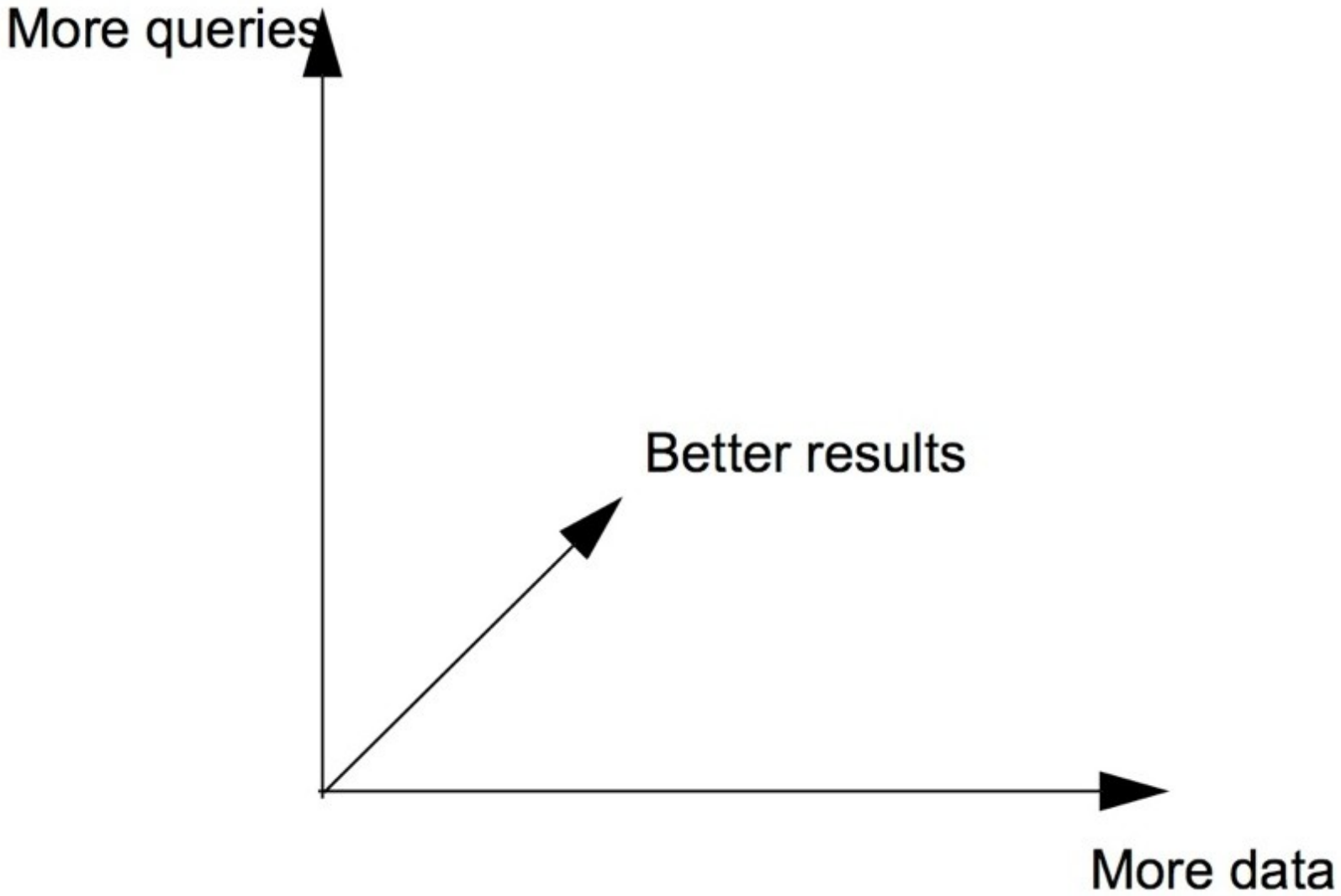
Capacidad de almacenamiento

- Cada servidor ofrece capacidad para 2 TB (año 2006).
- Un rack de 80 servidores provee 160 TB.
- Un cluster de 30 racks ofrece 4.8 PB.

Capacidad de almacenamiento

- Cada servidor ofrece capacidad para 2 TB (año 2006).
- Un rack de 80 servidores provee 160 TB.
- Un cluster de 30 racks ofrece 4.8 PB.
- La cantidad total de clusters es un secreto.

El problema de la escalabilidad



El problema de la escalabilidad

- Demanda estrategias sofisticadas de sistemas distribuidos.

El problema de la escalabilidad

- Demanda estrategias sofisticadas de sistemas distribuidos.
- Si asumiéramos que la web consiste de 20 mil millones de páginas de 20 KB cada una.

El problema de la escalabilidad

- Demanda estrategias sofisticadas de sistemas distribuidos.
- Si asumiéramos que la web consiste de 20 mil millones de páginas de 20 KB cada una.
- El tamaño total sería de 400 TB.

El problema de la escalabilidad

- Demanda estrategias sofisticadas de sistemas distribuidos.
- Si asumiéramos que la web consiste de 20 mil millones de páginas de 20 KB cada una.
- El tamaño total sería de 400 TB.
- Si una computadora puede leer 30 MB por segundo, un crawler duraría 4 meses en recorrerla.

El problema de la escalabilidad

- Demanda estrategias sofisticadas de sistemas distribuidos.
- Si asumiéramos que la web consiste de 20 mil millones de páginas de 20 KB cada una.
- El tamaño total sería de 400 TB.
- Si una computadora puede leer 30 MB por segundo, un crawler duraría 4 meses en recorrerla.
- En contraste, 1000 máquinas podrían leer esta cantidad de datos en menos de 3 horas.

El problema de la escalabilidad

- Demanda estrategias sofisticadas de sistemas distribuidos.
- Si asumiéramos que la web consiste de 20 mil millones de páginas de 20 KB cada una.
- El tamaño total sería de 400 TB.
- Si una computadora puede leer 30 MB por segundo, un crawler duraría 4 meses en recorrerla.
- En contraste, 1000 máquinas podrían leer esta cantidad de datos en menos de 3 horas.
- Del mismo modo, las otras funciones (indexado, ranking y búsqueda) también requieren soluciones altamente distribuidas para escalar.

El problema de la confiabilidad

- Se debe garantizar disponibilidad del servicio 24/7.

El problema de la confiabilidad

- Se debe garantizar disponibilidad del servicio 24/7.
- El 1 de setiembre de 2009 Gmail estuvo caído por 100 minutos.

El problema de la confiabilidad

- Se debe garantizar disponibilidad del servicio 24/7.
- El 1 de setiembre de 2009 Gmail estuvo caído por 100 minutos.
- Se debe ser capaz de anticipar las fallas de un modo u otro.

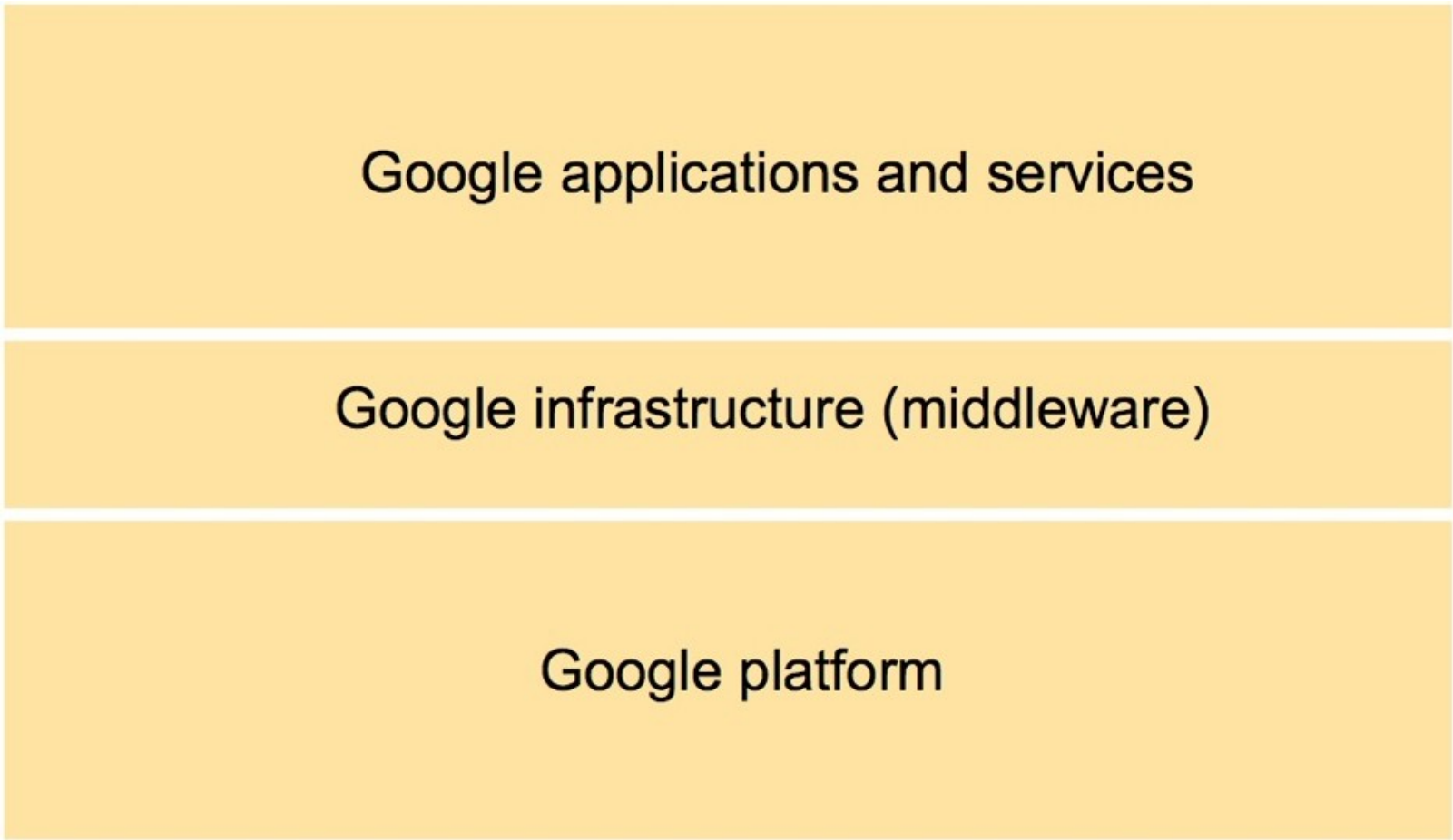
Otros problemas a resolver

- Rendimiento: Baja latencia entre interacciones.

Otros problemas a resolver

- Rendimiento: Baja latencia entre interacciones.
- Apertura: Estimular innovación, arquitectura extensible y estrategias y principios de diseño comunes para proveer coherencia a programadores.

Arquitectura global



Infraestructura

El sistema está construido como un conjunto de servicios que ofrecen la funcionalidad básica a los desarrolladores. Este conjunto puede ser particionado en los siguientes subconjuntos:

- Los paradigmas de comunicación, incluyendo servicios para invocación remota y comunicación indirecta.

Infraestructura

El sistema está construido como un conjunto de servicios que ofrecen la funcionalidad básica a los desarrolladores. Este conjunto puede ser particionado en los siguientes subconjuntos:

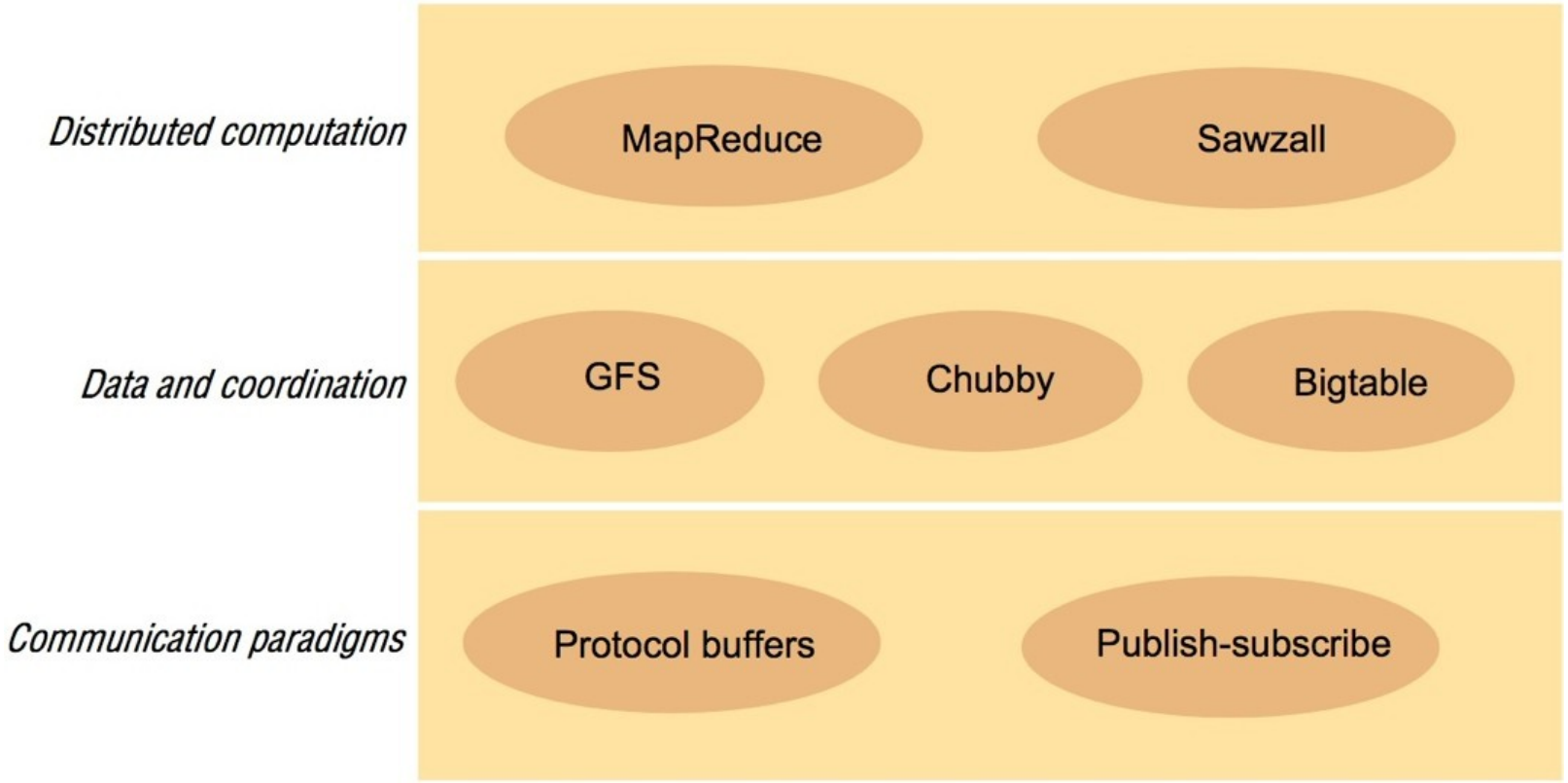
- Los paradigmas de comunicación, incluyendo servicios para invocación remota y comunicación indirecta.
- Los servicios de datos y de coordinación, que proveen abstracciones estructuradas y semiestructuradas para el almacenamiento de datos, junto con los servicios para soportar el acceso coordinado a los datos.

Infraestructura

El sistema está construido como un conjunto de servicios que ofrecen la funcionalidad básica a los desarrolladores. Este conjunto puede ser particionado en los siguientes subconjuntos:

- Los paradigmas de comunicación, incluyendo servicios para invocación remota y comunicación indirecta.
- Los servicios de datos y de coordinación, que proveen abstracciones estructuradas y semiestructuradas para el almacenamiento de datos, junto con los servicios para soportar el acceso coordinado a los datos.
- Los servicios de computación distribuida, que proveen los medios para llevar a cabo la computación paralela y distribuida sobre la infraestructura física.

Infraestructura



Principios de diseño asociados

- Simplicidad (el principio más importante). Por ejemplo, una API debe ser tan pequeña como sea posible, pero no más pequeña (Occam's Razor).

Principios de diseño asociados

- Simplicidad (el principio más importante). Por ejemplo, una API debe ser tan pequeña como sea posible, pero no más pequeña (Occam's Razor).
- Rendimiento (*cada milisegundo cuenta*). Un diseñador debería ser capaz de poder estimar el rendimiento de un sistema basado en su conocimiento de los costos de rendimiento de las operaciones primitivas necesarias (como acceso a memoria o lecturas de disco).

Principios de diseño asociados

- Simplicidad (el principio más importante). Por ejemplo, una API debe ser tan pequeña como sea posible, pero no más pequeña (Occam's Razor).
- Rendimiento (*cada milisegundo cuenta*). Un diseñador debería ser capaz de poder estimar el rendimiento de un sistema basado en su conocimiento de los costos de rendimiento de las operaciones primitivas necesarias (como acceso a memoria o lecturas de disco).
- Pruebas meticulosas. Además, las pruebas deben ser complementadas con un énfasis en *logging* y *tracing*.

Outline

Paradigmas de comunicación

Esta elección es crucial para el éxito del diseño global del sistema. Las opciones incluyen:

- Utilizar un servicio de IPC directamente, como los ofrecidos con sockets.

Paradigmas de comunicación

Esta elección es crucial para el éxito del diseño global del sistema. Las opciones incluyen:

- Utilizar un servicio de IPC directamente, como los ofrecidos con sockets.
- Utilizar un servicio de invocación remota (como RPC o RMI).

Paradigmas de comunicación

Esta elección es crucial para el éxito del diseño global del sistema. Las opciones incluyen:

- Utilizar un servicio de IPC directamente, como los ofrecidos con sockets.
- Utilizar un servicio de invocación remota (como RPC o RMI).
- Utilizar un paradigma de comunicación indirecta, como comunicación grupal, aproximaciones distribuidas basadas en eventos o memoria compartida distribuida.

Esquema escogido

- De acuerdo con sus principios de diseño, Google adopta un servicio de invocación remota simple, mínimo y eficiente (variante de RPC).

Esquema escogido

- De acuerdo con sus principios de diseño, Google adopta un servicio de invocación remota simple, mínimo y eficiente (variante de RPC).
- RPC requiere de un componente de serialización para convertir los datos del procedimiento de invocación (nombre del procedimiento y parámetros, posiblemente estructurados).

Esquema escogido

- De acuerdo con sus principios de diseño, Google adopta un servicio de invocación remota simple, mínimo y eficiente (variante de RPC).
- RPC requiere de un componente de serialización para convertir los datos del procedimiento de invocación (nombre del procedimiento y parámetros, posiblemente estructurados).
- Estos datos son convertidos de su representación binaria interna en una representación con formato plano o serializado, independiente de procesador, listo para ser transmitido al socio remoto.

Esquema escogido

- De acuerdo con sus principios de diseño, Google adopta un servicio de invocación remota simple, mínimo y eficiente (variante de RPC).
- RPC requiere de un componente de serialización para convertir los datos del procedimiento de invocación (nombre del procedimiento y parámetros, posiblemente estructurados).
- Estos datos son convertidos de su representación binaria interna en una representación con formato plano o serializado, independiente de procesador, listo para ser transmitido al socio remoto.
- En Java RPC, XML es el formato universal para serialización. Sin embargo, tiene un alto *overhead*.

Protocol buffers

- Por esto, Google desarrolló un componente de serialización simplificado y de alto rendimiento, conocido como *protocol buffers*.

Protocol buffers

- Por esto, Google desarrolló un componente de serialización simplificado y de alto rendimiento, conocido como *protocol buffers*.
- Éste se utiliza en la mayoría de las interacciones dentro de la infraestructura.

Protocol buffers

- Por esto, Google desarrolló un componente de serialización simplificado y de alto rendimiento, conocido como *protocol buffers*.
- Éste se utiliza en la mayoría de las interacciones dentro de la infraestructura.
- Puede ser usado en cualquier mecanismo de comunicación y provee capacidad para RPC.

Protocol buffers

- Por esto, Google desarrolló un componente de serialización simplificado y de alto rendimiento, conocido como *protocol buffers*.
- Éste se utiliza en la mayoría de las interacciones dentro de la infraestructura.
- Puede ser usado en cualquier mecanismo de comunicación y provee capacidad para RPC.
- Existe una versión de código abierto disponible.

Servicio *publish-subscribe*

- Se utiliza también un servicio aparte tipo *publish-subscribe*.

Servicio *publish-subscribe*

- Se utiliza también un servicio aparte tipo *publish-subscribe*.
- Se reconoce el papel clave que un servicio de este tipo provee al diseño de un sistema distribuido.

Servicio *publish-subscribe*

- Se utiliza también un servicio aparte tipo *publish-subscribe*.
- Se reconoce el papel clave que un servicio de este tipo provee al diseño de un sistema distribuido.
- Por ejemplo, para la diseminación de eventos de múltiples participantes.

Servicio *publish-subscribe*

- Se utiliza también un servicio aparte tipo *publish-subscribe*.
- Se reconoce el papel clave que un servicio de este tipo provee al diseño de un sistema distribuido.
- Por ejemplo, para la diseminación de eventos de múltiples participantes.
- Al igual que muchas otras soluciones en sistemas distribuidos, se ofrece un paradigma de comunicación híbrido, dejando que los desarrolladores escojan el esquema que mejor se adapte a sus requerimientos.

Servicio *publish-subscribe*

- Se utiliza también un servicio aparte tipo *publish-subscribe*.
- Se reconoce el papel clave que un servicio de este tipo provee al diseño de un sistema distribuido.
- Por ejemplo, para la diseminación de eventos de múltiples participantes.
- Al igual que muchas otras soluciones en sistemas distribuidos, se ofrece un paradigma de comunicación híbrido, dejando que los desarrolladores escojan el esquema que mejor se adapte a sus requerimientos.
- *Publish-subscribe* no es una alternativa a *protocol buffers*, más bien un servicio de valor agregado para donde es más apropiado.

Invocación remota

- *Protocol buffers* pone especial énfasis en la descripción y subsecuente serialización de los datos.

Invocación remota

- *Protocol buffers* pone especial énfasis en la descripción y subsecuente serialización de los datos.
- Es posible compararlo con alternativas directas, como XML.

Invocación remota

- *Protocol buffers* pone especial énfasis en la descripción y subsecuente serialización de los datos.
- Es posible compararlo con alternativas directas, como XML.
- El objetivo es proveer un modo de especificar y serializar datos (independiente del lenguaje y plataforma), de una forma simple, altamente eficiente y extensible.

Invocación remota

- *Protocol buffers* pone especial énfasis en la descripción y subsecuente serialización de los datos.
- Es posible compararlo con alternativas directas, como XML.
- El objetivo es proveer un modo de especificar y serializar datos (independiente del lenguaje y plataforma), de una forma simple, altamente eficiente y extensible.
- Los datos serializados pueden ser utilizados para su almacenamiento subsecuente o para ser transmitidos por medio de los protocolos de comunicación subyacentes. O para cualquier otro propósito que requiera la serialización de los datos.

Invocación remota

- En *protocol buffers* se provee un lenguaje para la especificación de mensajes.

Invocación remota

- En *protocol buffers* se provee un lenguaje para la especificación de mensajes.
- Vamos a ver sus características claves por medio de un ejemplo.

Ejemplo de *protocol buffers*

```
message Book {
    required string title = 1;
    repeated string author = 2;
    enum Status {
        IN_PRESS = 0;
        PUBLISHED = 1;
        OUT_OF_PRINT = 2;
    }
    message BookStats {
        required int32 sales = 1;
        optional int32 citations = 2;
        optional Status bookstatus = 3 [default = PUBLISHED];
    }
    optional BookStats statistics = 3;
    repeated string keyword = 4;
}
```

Ejemplo de *protocol buffers*

Los tipos de datos pueden ser:

- Primitive data type (incluyendo integer, floating-point, boolean, string o raw bytes).

Ejemplo de *protocol buffers*

Los tipos de datos pueden ser:

- Primitive data type (incluyendo integer, floating-point, boolean, string o raw bytes).
- Enumerated type.

Ejemplo de *protocol buffers*

Los tipos de datos pueden ser:

- Primitive data type (incluyendo integer, floating-point, boolean, string o raw bytes).
- Enumerated type.
- Nested message (permite estructuras de datos jerárquicas).

Ejemplo de *protocol buffers*

Además, los campos son anotados con una de tres etiquetas:

- required
- optional
- repeated

Ejemplo de *protocol buffers*

- Esta especificación se escribe en un archivo tipo *.proto* y se compila con una herramienta *protoc*.

Ejemplo de *protocol buffers*

- Esta especificación se escribe en un archivo tipo *.proto* y se compila con una herramienta *protoc*.
- La salida de esta herramiente es código generado que permite a los programadores manipular un tipo particular de mensaje, especialmente asignando/extrayendo valores hacia/desde mensajes.

Ejemplo de *protocol buffers*

- Esta especificación se escribe en un archivo tipo *.proto* y se compila con una herramienta *protoc*.
- La salida de esta herramiente es código generado que permite a los programadores manipular un tipo particular de mensaje, especialmente asignando/extrayendo valores hacia/desde mensajes.
- *protoc* genera una clase *builder* que provee métodos *getter* y *setter* para cada campo y otros métodos adicionales.

Ejemplo de *protocol buffers*

Los siguientes métodos son generados para el campo *title*:

- public boolean hasTitle();
- public java.lang.String getTitle();
- public Builder setTitle(String value);
- public Builder clearTitle();

La importancia de la clase builder se da porque mientras que los mensajes son inmutables en protocol buffers, los builders sí pueden cambiarse y son utilizados para construir y manipular mensajes.

Ejemplo de *protocol buffers*

Para campos con etiqueta *repeated* el código generado es algo más complejo, con métodos para regresar un contador con el número de elementos en la lista asociada, para obtener o asignar campos específicos en la lista, para agregar un nuevo elemento a la lista y para agregar un conjunto de elementos a la lista.

Ejemplo de *protocol buffers*

Los siguientes son los asociados al campo *keyword*:

- public List<string>getKeywordList();
- public int getKeywordCount();
- public string getKeyword(int index);
- public Builder setKeyword(int index, string value);
- public Builder addKeyword(string value);
- public Builder addAllKeyword(Iterable<string>value);
- public Builder clearKeyword();

El código generado también provee otros métodos para manipular mensajes, como *toString*, o para *parsear* mensajes entrantes.

Ejemplo de *protocol buffers*

Este formato es bastante simple comparado con XML y según sus desarrolladores:

Ejemplo de *protocol buffers*

Este formato es bastante simple comparado con XML y según sus desarrolladores:

- De 3 a 10 veces más pequeño que XML.

Ejemplo de *protocol buffers*

Este formato es bastante simple comparado con XML y según sus desarrolladores:

- De 3 a 10 veces más pequeño que XML.
- De 10 a 100 veces más rápido que XML.

Ejemplo de *protocol buffers*

Este formato es bastante simple comparado con XML y según sus desarrolladores:

- De 3 a 10 veces más pequeño que XML.
- De 10 a 100 veces más rápido que XML.
- Su API es también más simple.

Ejemplo de *protocol buffers*

Sin embargo, esta comparación no es del todo justa, por dos razones:

Ejemplo de *protocol buffers*

Sin embargo, esta comparación no es del todo justa, por dos razones:

- La infraestructura de Google es un sistema relativamente cerrado y, a diferencia de XML, no debe lidiar con problemas de interoperabilidad entre sistemas abiertos.

Ejemplo de *protocol buffers*

Sin embargo, esta comparación no es del todo justa, por dos razones:

- La infraestructura de Google es un sistema relativamente cerrado y, a diferencia de XML, no debe lidiar con problemas de interoperabilidad entre sistemas abiertos.
- XML es significativamente más rico en el sentido de que genera mensajes auto-descriptivos que contienen los datos y los metadatos asociados que describen la estructura de los mensajes. Protocol buffers no provee esta facilidad directamente (aunque es posible conseguir este efecto solicitando a *protoc* que genere un *FileDescriptorSet*).

Soporte de RPC

- Protocol buffers es un mecanismo general que puede ser utilizado para almacenamiento o para comunicación.

Soporte de RPC

- Protocol buffers es un mecanismo general que puede ser utilizado para almacenamiento o para comunicación.
- El uso más común es para especificar intercambios de RPC a través de la red, acomodada con una sintaxis extra en el lenguaje.

Soporte de RPC

```
service SearchService {
        rpc Search (RequestType) returns (ResponseType);
}
```

Soporte de RPC

```
service SearchService {
        rpc Search (RequestType) returns (ResponseType);
}
```

- Este fragmento de código especifica una interfaz de servicio llamada *SearchService*.

Soporte de RPC

```
service SearchService {
        rpc Search (RequestType) returns (ResponseType);
}
```

- Este fragmento de código especifica una interfaz de servicio llamada *SearchService*.
- Contiene una operación remota: *Search*.

Soporte de RPC

```
service SearchService {
        rpc Search (RequestType) returns (ResponseType);
}
```

- Este fragmento de código especifica una interfaz de servicio llamada *SearchService*.
- Contiene una operación remota: *Search*.
- *Search* toma un parámetro de tipo *RequestType* y regresa un parámetro de tipo *ResponseType*.

Publish-subscribe

- Principal aplicación: sistema de Google Ads.

Publish-subscribe

- Principal aplicación: sistema de Google Ads.
- Google adopta un sistema basado en tópicos.

Resumen de elecciones de diseño

Element	*Design choice*	*Rationale*	*Trade-offs*
Protocol buffers	The use of a language for specifying data formats	Flexible in that the same language can be used for serializing data for storage or communication	-
	Simplicity of the language	Efficient implementation	Lack of expressiveness when compared, for example, with XML
	Support for a style of RPC (taking a single message as a parameter and returning a single message as result)	More efficient, extensible and supports service evolution	Lack of expressiveness when compared with other RPC or RMI packages
	Protocol-agnostic design	Different RPC implementations can be used	No common semantics for RPC exchanges

Resumen de elecciones de diseño

Publish-subscribe	Topic-based approach	Supports efficient implementation	Less expressive than content-based approaches (mitigated by the additional filtering capabilities)
	Real-time and reliability guarantees	Supports maintenance of consistent views in a timely manner	Additional algorithmic support required with associated overhead

Outline

Almacenamiento de datos y servicios de coordinación

Veremos tres componentes que juntos proveen servicios de almacenamiento de datos y coordinación a las aplicaciones de alto nivel. Se trata de servicios complementarios dentro de la infraestructura de Google.

Almacenamiento de datos y servicios de coordinación

- *Google File System*. Es un sistema de archivos distribuido que ofrece servicios de forma similar a NFS y AFS. Permite el acceso a datos no estructurados (en forma de archivos), pero optimizado para el estilo de datos y acceso que requiere Google (por ejemplo, archivos muy grandes).

Almacenamiento de datos y servicios de coordinación

- *Google File System*. Es un sistema de archivos distribuido que ofrece servicios de forma similar a NFS y AFS. Permite el acceso a datos no estructurados (en forma de archivos), pero optimizado para el estilo de datos y acceso que requiere Google (por ejemplo, archivos muy grandes).
- *Chubby*. Es un servicio que soporta cerraduras distribuidas para coordinación y para el almacenamiento de pequeñas cantidades de datos.

Almacenamiento de datos y servicios de coordinación

- *Google File System*. Es un sistema de archivos distribuido que ofrece servicios de forma similar a NFS y AFS. Permite el acceso a datos no estructurados (en forma de archivos), pero optimizado para el estilo de datos y acceso que requiere Google (por ejemplo, archivos muy grandes).
- *Chubby*. Es un servicio que soporta cerraduras distribuidas para coordinación y para el almacenamiento de pequeñas cantidades de datos.
- *Bigtable*. Ofrece acceso a datos más estructurados en forma de tablas que pueden ser indexadas de diferentes formas, incluyendo por filas o por columnas. Es un tipo de base de datos distribuida, pero a diferencia de muchas bases de datos no soporta operadores relacionales completos (éstos son vistos por Google como innecesariamente complejos y no escalables).

Almacenamiento de datos y servicios de coordinación

Los tres servicios son interdependientes. Por ejemplo, Bigtable usa GFS para el almacenamiento y Chubby para la coordinación.

Google File System

- Es un sistema de archivos distribuido especializado en los requerimientos de Google en cuanto al almacenamiento y acceso a cantidades muy grandes de datos.

Google File System

- Es un sistema de archivos distribuido especializado en los requerimientos de Google en cuanto al almacenamiento y acceso a cantidades muy grandes de datos.
- Estos requerimientos incluyen el correr de forma confiable dentro de la infraestructura de Google, compuesta de una gran cantidad de servidores de bajo costo.

Google File System

- Es un sistema de archivos distribuido especializado en los requerimientos de Google en cuanto al almacenamiento y acceso a cantidades muy grandes de datos.
- Estos requerimientos incluyen el correr de forma confiable dentro de la infraestructura de Google, compuesta de una gran cantidad de servidores de bajo costo.
- El diseño contempla que el hardware o el software pueden fallar en cualquier momento por lo que el sistema debe ser tolerante con dichas fallas y permitir que las aplicaciones de alto nivel continúen corriendo.

Google File System

- GFS debe escalar, tanto en cuanto al volumen de datos como a la cantidad de usuarios.

Google File System

- GFS debe escalar, tanto en cuanto al volumen de datos como a la cantidad de usuarios.
- Debe soportar el desarrollo de nuevas aplicaciones web.

Google File System

- GFS debe escalar, tanto en cuanto al volumen de datos como a la cantidad de usuarios.
- Debe soportar el desarrollo de nuevas aplicaciones web.
- Se prioriza el rendimiento en la lectura de datos.

Google File System

- GFS debe escalar, tanto en cuanto al volumen de datos como a la cantidad de usuarios.
- Debe soportar el desarrollo de nuevas aplicaciones web.
- Se prioriza el rendimiento en la lectura de datos.
- GFS está optimizado para los patrones de uso y acceso a datos de Google. El número de archivos tiende a no ser muy grande pero sí su tamaño (en 2003, del orden de un millón de archivos de 100 MB en promedio, pero algunos de más de 1 GB).

Google File System

Aunque no provee compatibilidad completa con POSIX, muchas de sus operaciones son familiares para los usuarios de sistemas de este tipo:

- create y delete

Google File System

Aunque no provee compatibilidad completa con POSIX, muchas de sus operaciones son familiares para los usuarios de sistemas de este tipo:

- create y delete
- open y close

Google File System

Aunque no provee compatibilidad completa con POSIX, muchas de sus operaciones son familiares para los usuarios de sistemas de este tipo:

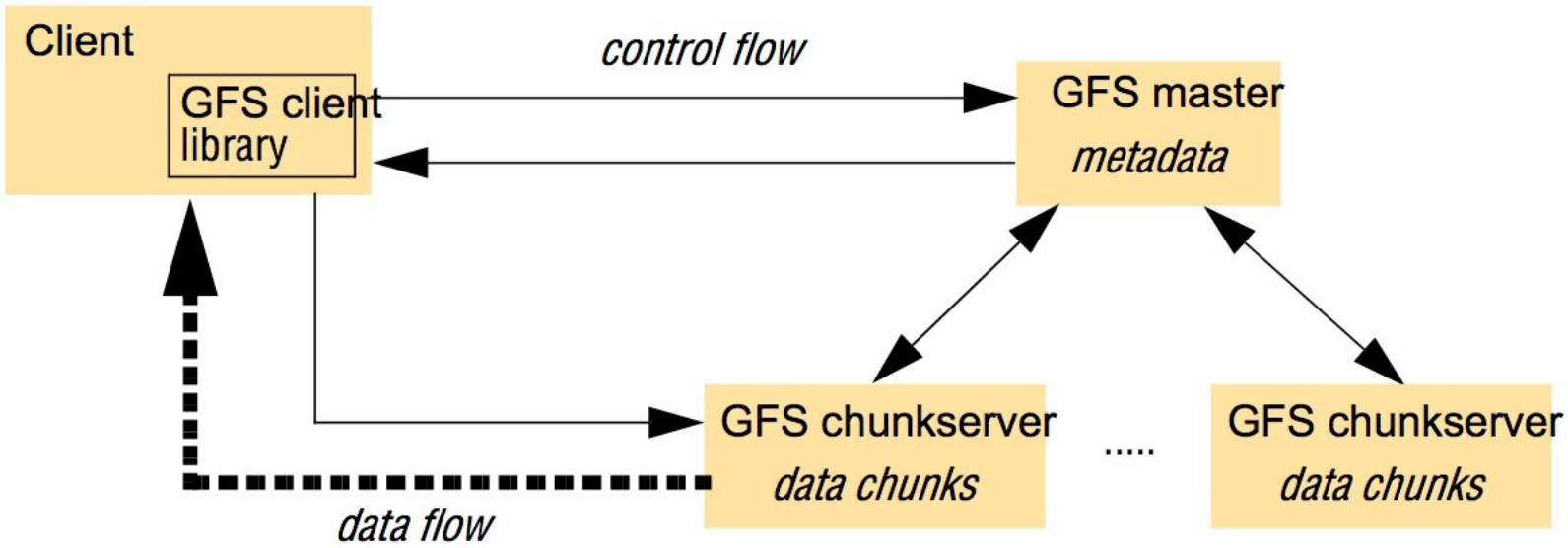
- create y delete
- open y close
- read y write

Google File System

Aunque no provee compatibilidad completa con POSIX, muchas de sus operaciones son familiares para los usuarios de sistemas de este tipo:

- create y delete
- open y close
- read y write
- snapshot y record append

Arquitectura



Problemas del esquema centralizado

- El máster es un cuello de botella.

Problemas del esquema centralizado

- El máster es un cuello de botella.
- A pesar de la reducción de metadatos, la cantidad de éstos ha crecido tanto que ya es difícil mantenerlos en memoria principal.

Problemas del esquema centralizado

- El máster es un cuello de botella.
- A pesar de la reducción de metadatos, la cantidad de éstos ha crecido tanto que ya es difícil mantenerlos en memoria principal.

Por esta razón en este momento Google está trabajando en una solución de máster distribuido.

Caching

- GFS no hace mucho uso de caching.

Caching

- GFS no hace mucho uso de caching.
- Información sobre la ubicación de los chunks está en caché en los clientes luego del primer uso para minimizar las consultas al máster.

Caching

- GFS no hace mucho uso de caching.
- Información sobre la ubicación de los chunks está en caché en los clientes luego del primer uso para minimizar las consultas al máster.
- No se guarda en caché el contenido de los archivos.

Caching

- GFS no hace mucho uso de caching.
- Información sobre la ubicación de los chunks está en caché en los clientes luego del primer uso para minimizar las consultas al máster.
- No se guarda en caché el contenido de los archivos.
- Se evita el uso de protocolos para la consistencia de las cachés.

Caching

- GFS no hace mucho uso de caching.
- Información sobre la ubicación de los chunks está en caché en los clientes luego del primer uso para minimizar las consultas al máster.
- No se guarda en caché el contenido de los archivos.
- Se evita el uso de protocolos para la consistencia de las cachés.
- Tampoco se usan cachés a nivel de servidor más que los búfers de Linux.

Logging

- GFS es un ejemplo clave del uso de logging en Google para el soporte de debugging y análisis de rendimiento.

Logging

- GFS es un ejemplo clave del uso de logging en Google para el soporte de debugging y análisis de rendimiento.
- Los servidores de GFS mantienen logs de diagnóstico extensivos que guardan eventos significativos y todas las solicitudes y respuestas de RPC.

Logging

- GFS es un ejemplo clave del uso de logging en Google para el soporte de debugging y análisis de rendimiento.
- Los servidores de GFS mantienen logs de diagnóstico extensivos que guardan eventos significativos y todas las solicitudes y respuestas de RPC.
- Estos logs son monitoreados continuamente.

Logging

- GFS es un ejemplo clave del uso de logging en Google para el soporte de debugging y análisis de rendimiento.
- Los servidores de GFS mantienen logs de diagnóstico extensivos que guardan eventos significativos y todas las solicitudes y respuestas de RPC.
- Estos logs son monitoreados continuamente.
- Son usados cuando se dan problemas en el sistema para identificar las causas subyacentes.

Administración de consistencia en GFS

- Dado que los datos son replicados en GFS, es importante mantener la consistencia de las réplicas durante las operaciones que alteran los datos, es decir, las operaciones write y record append.

Administración de consistencia en GFS

- Dado que los datos son replicados en GFS, es importante mantener la consistencia de las réplicas durante las operaciones que alteran los datos, es decir, las operaciones write y record append.
- Los clientes envían los cambios a todas las réplicas pero la solicitud de cambio se la envían sólo al primario.

Administración de consistencia en GFS

- Dado que los datos son replicados en GFS, es importante mantener la consistencia de las réplicas durante las operaciones que alteran los datos, es decir, las operaciones write y record append.
- Los clientes envían los cambios a todas las réplicas pero la solicitud de cambio se la envían sólo al primario.
- El primario se encarga entonces de coordinar las mutaciones en las réplicas.

Administración de consistencia en GFS

- Dado que los datos son replicados en GFS, es importante mantener la consistencia de las réplicas durante las operaciones que alteran los datos, es decir, las operaciones write y record append.
- Los clientes envían los cambios a todas las réplicas pero la solicitud de cambio se la envían sólo al primario.
- El primario se encarga entonces de coordinar las mutaciones en las réplicas.
- De este modo se crea una separación entre el flujo de datos y el flujo de control.

Chubby

- Chubby es un servicio crucial en el corazón de la infraestructura de Google.

Chubby

- Chubby es un servicio crucial en el corazón de la infraestructura de Google.
- Ofrece servicios de almacenamiento y servicios de coordinación para otros servicios de la infraestructura, incluyendo GFS y Bigtable.

Chubby

Los servicios de Chubby permiten cuatro capacidades diferentes:

- Proveen cerraduras distribuidas para sincronizar actividades en un ambiente a gran escala, no sincronizado.

Chubby

Los servicios de Chubby permiten cuatro capacidades diferentes:

- Proveen cerraduras distribuidas para sincronizar actividades en un ambiente a gran escala, no sincronizado.
- Proveen un sistema de archivos que permite guardar archivos pequeños de forma confiable (complementando los servicios de GFS).

Chubby

Los servicios de Chubby permiten cuatro capacidades diferentes:

- Proveen cerraduras distribuidas para sincronizar actividades en un ambiente a gran escala, no sincronizado.
- Proveen un sistema de archivos que permite guardar archivos pequeños de forma confiable (complementando los servicios de GFS).
- Pueden ser utilizados para ayudar en la elección de un primario dentro un conjunto de réplicas.

Chubby

Los servicios de Chubby permiten cuatro capacidades diferentes:

- Proveen cerraduras distribuidas para sincronizar actividades en un ambiente a gran escala, no sincronizado.
- Proveen un sistema de archivos que permite guardar archivos pequeños de forma confiable (complementando los servicios de GFS).
- Pueden ser utilizados para ayudar en la elección de un primario dentro un conjunto de réplicas.
- Se utiliza como un servicio de denominación (name service) dentro de Google.

Chubby

- Este recargo de funciones parece contradecir el principio de simplicidad en el diseño de Google.

Chubby

- Este recargo de funciones parece contradecir el principio de simplicidad en el diseño de Google.
- Sin embargo, en el fondo, Chubby ofrece un servicio fundamental que es una solución al problema de *consenso distribuido*.

Chubby

- Este recargo de funciones parece contradecir el principio de simplicidad en el diseño de Google.
- Sin embargo, en el fondo, Chubby ofrece un servicio fundamental que es una solución al problema de *consenso distribuido*.
- Las demás facetas afloran a partir de este servicio fundamental que está optimizado para el estilo de uso de Google.

Interfaz de Chubby

- Chubby provee una abstracción basada en un sistema de archivos, donde cada objeto de datos es un archivo.

Interfaz de Chubby

- Chubby provee una abstracción basada en un sistema de archivos, donde cada objeto de datos es un archivo.
- Los archivos se organizan de forma jerárquica por medio de directorios.

Interfaz de Chubby

- Chubby provee una abstracción basada en un sistema de archivos, donde cada objeto de datos es un archivo.
- Los archivos se organizan de forma jerárquica por medio de directorios.
- /ls/chubby-cell/directory-name/.../file-name

Interfaz de Chubby

- Chubby provee una abstracción basada en un sistema de archivos, donde cada objeto de datos es un archivo.
- Los archivos se organizan de forma jerárquica por medio de directorios.
- /ls/chubby-cell/directory-name/.../file-name
- /ls/local

API de Chubby

Role	*Operation*	*Effect*
General	*Open*	Opens a given named file or directory and returns a handle
	Close	Closes the file associated with the handle
	Delete	Deletes the file or directory
File	*GetContentsAndStat*	Returns (atomically) the whole file contents and metadata associated with the file
	GetStat	Returns just the metadata
	ReadDir	Returns the contents of a directory – that is, the names and metadata of any children
	SetContents	Writes the whole contents of a file (atomically)
	SetACL	Writes new access control list information
Lock	*Acquire*	Acquires a lock on a file
	TryAquire	Tries to acquire a lock on a file
	Release	Releases a lock

Elecciones de primarios

- Elegir una réplica como primario.

Elecciones de primarios

- Elegir una réplica como primario.
- Todos los candidatos tratan de obtener una cerradura asociada con la elección.

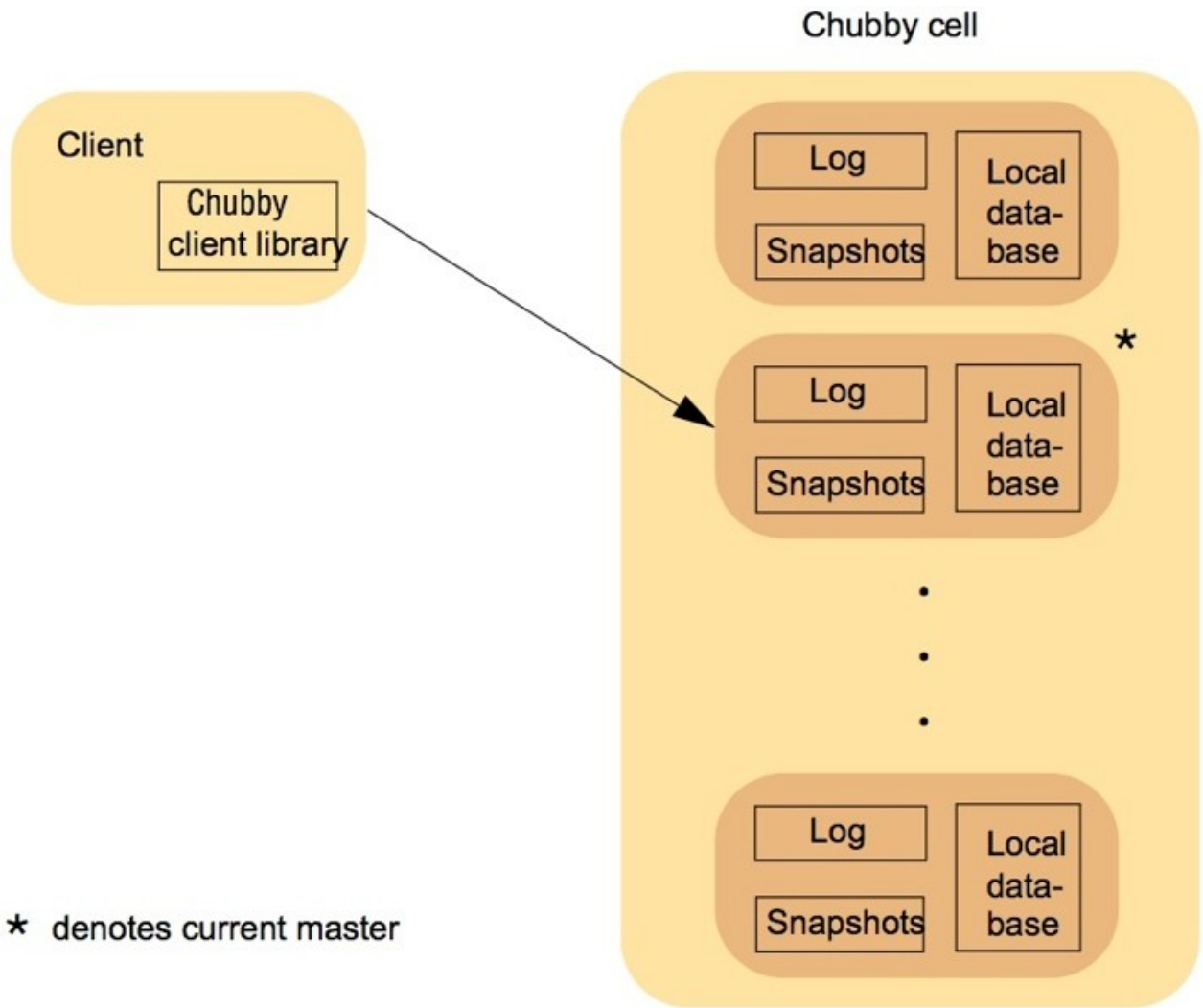
Elecciones de primarios

- Elegir una réplica como primario.
- Todos los candidatos tratan de obtener una cerradura asociada con la elección.
- Sólo uno lo adquiere y se convierte en primario (inscribe su id en el archivo).

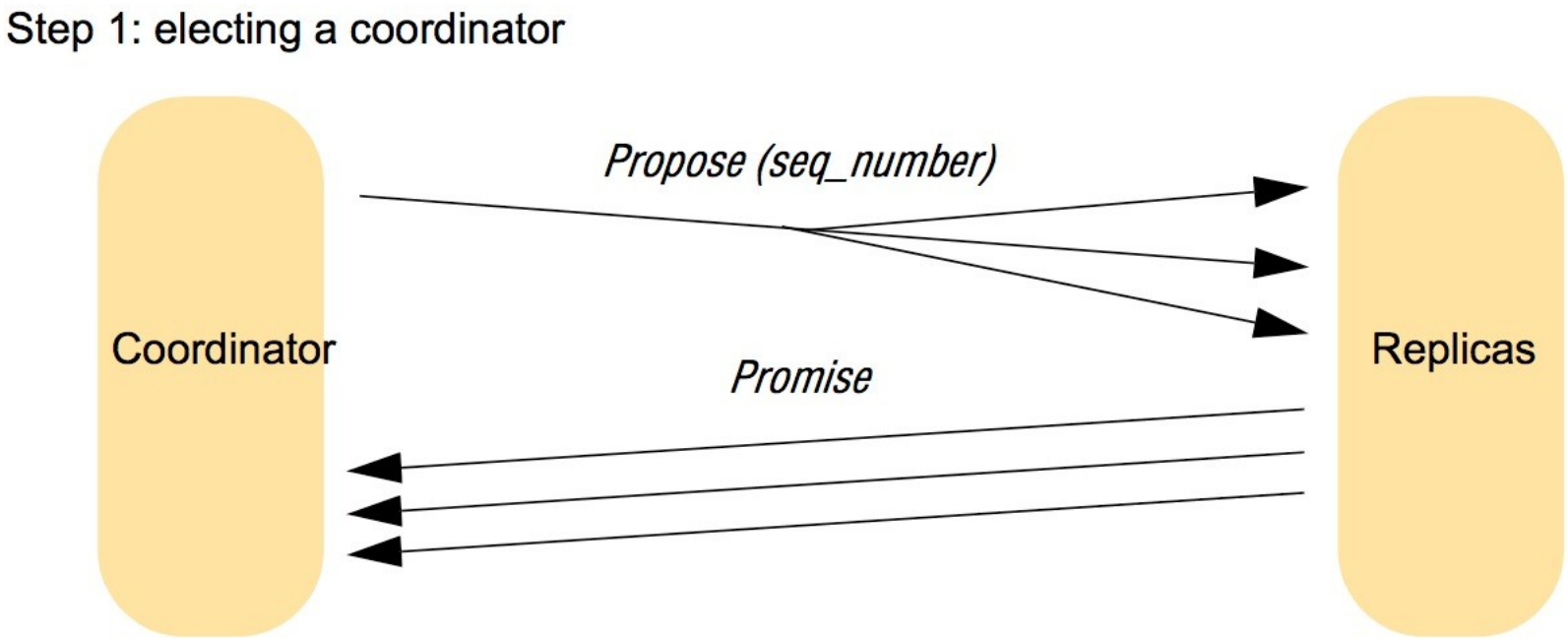
Elecciones de primarios

- Elegir una réplica como primario.
- Todos los candidatos tratan de obtener una cerradura asociada con la elección.
- Sólo uno lo adquiere y se convierte en primario (inscribe su id en el archivo).
- Todos los demás son secundarios.

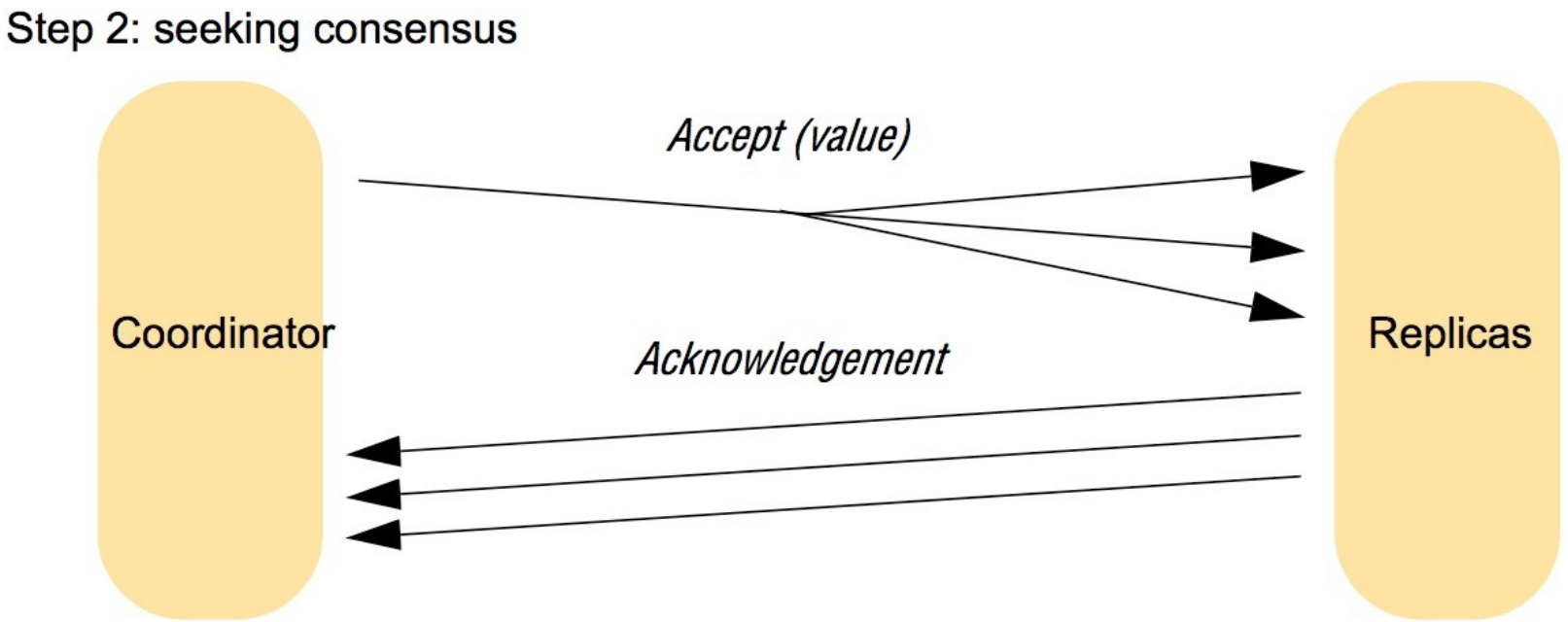
Arquitectura de Chubby



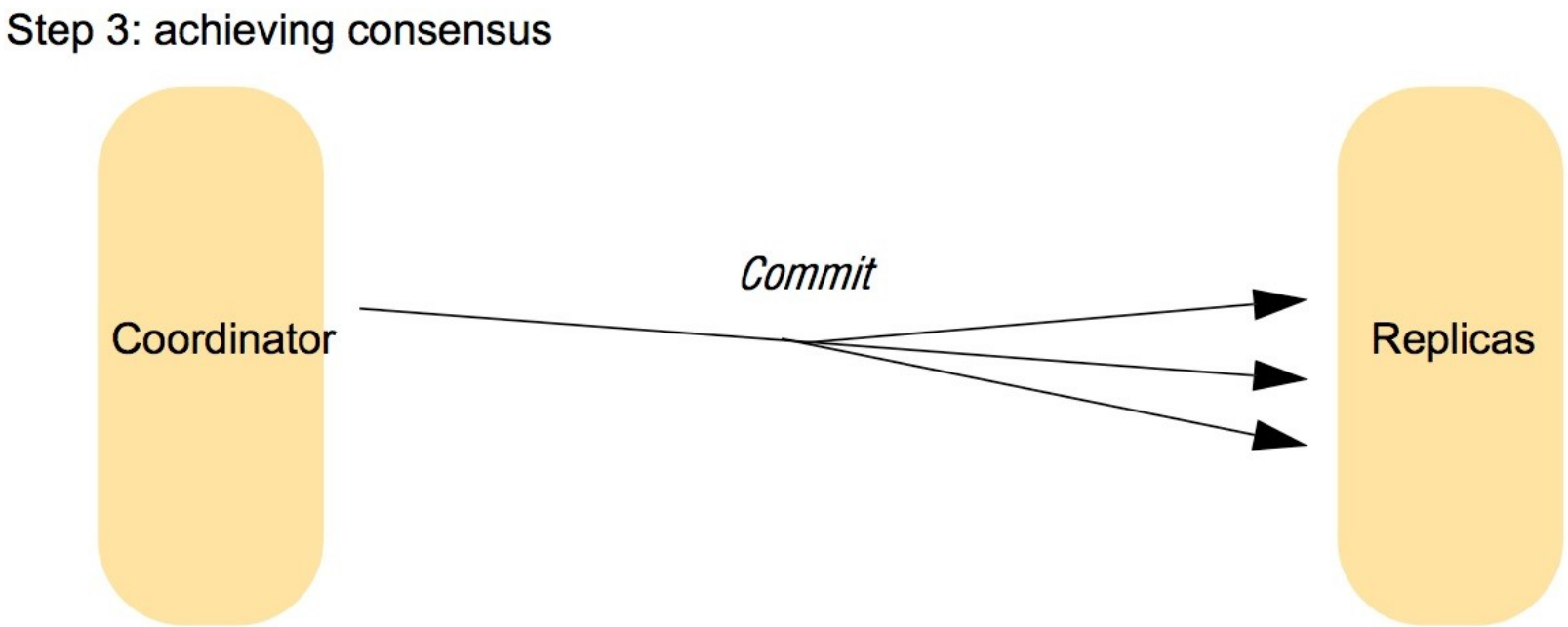
Intercambio de mensajes en Paxos



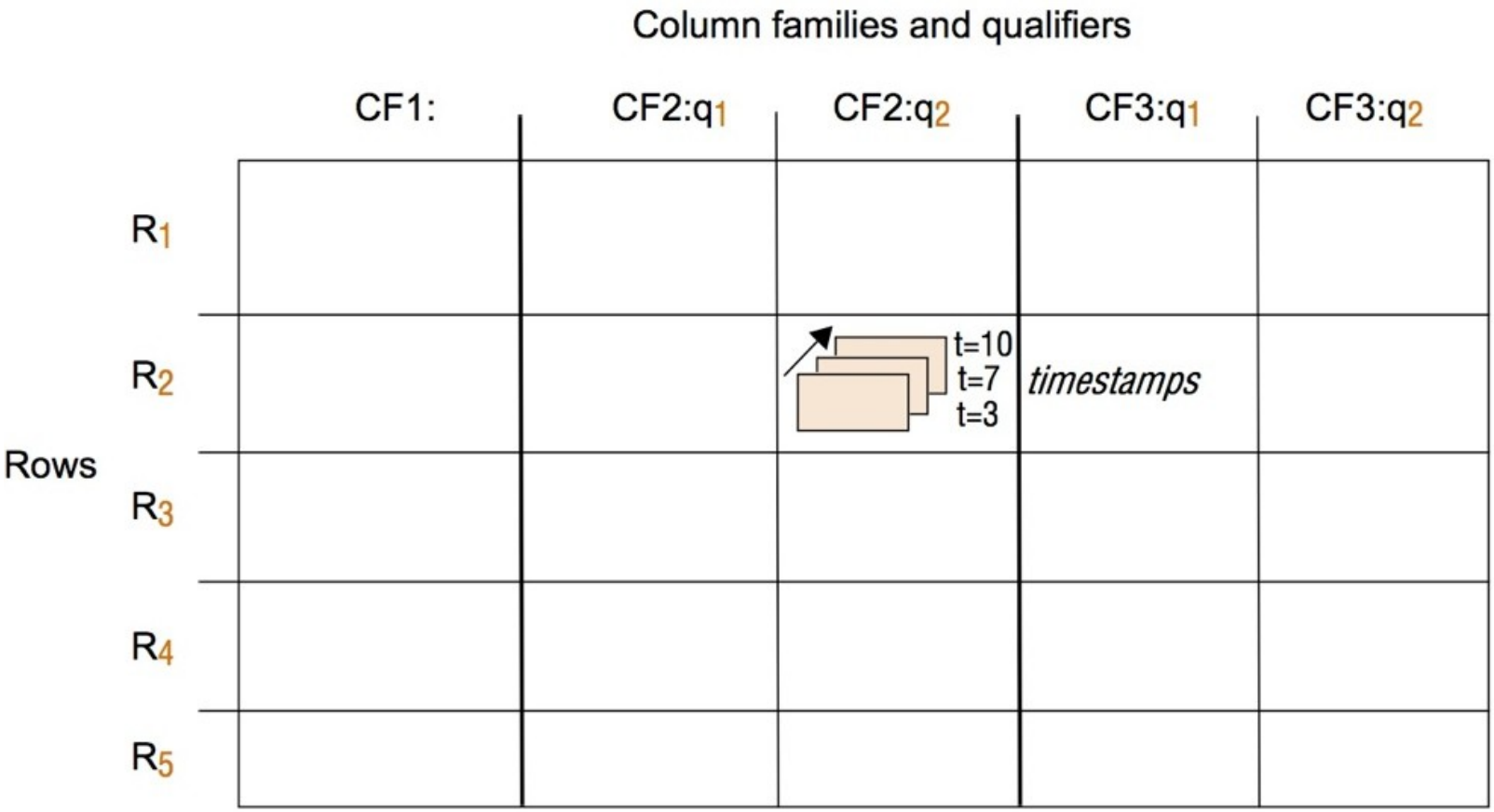
Intercambio de mensajes en Paxos



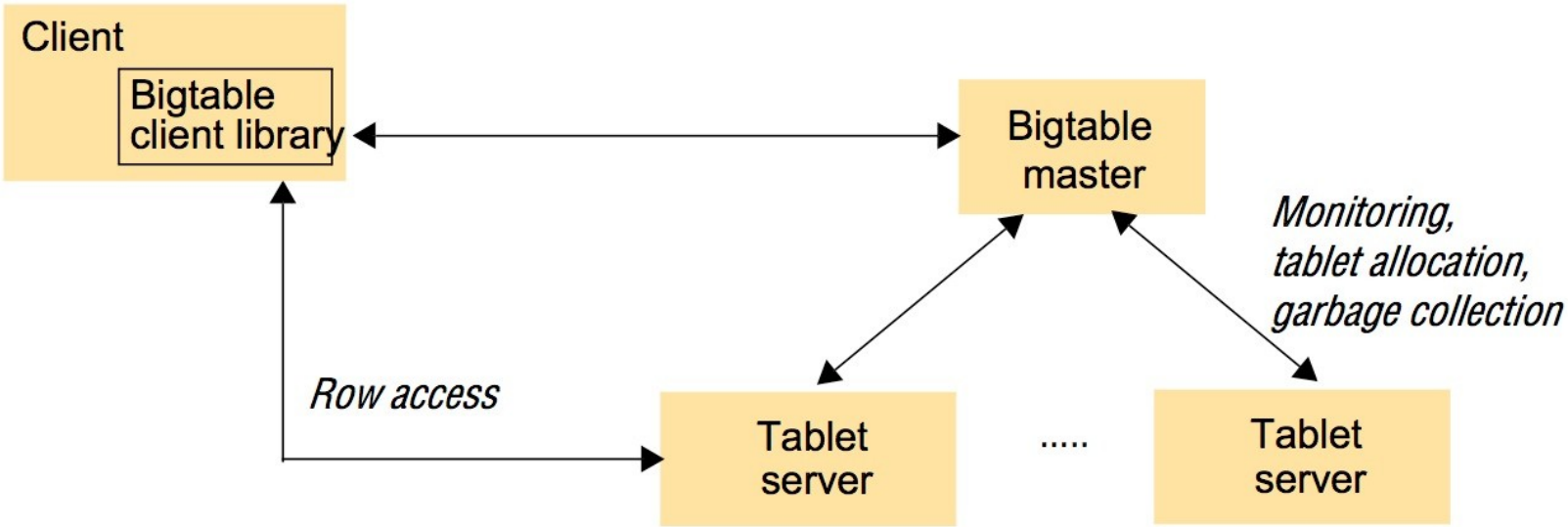
Intercambio de mensajes en Paxos



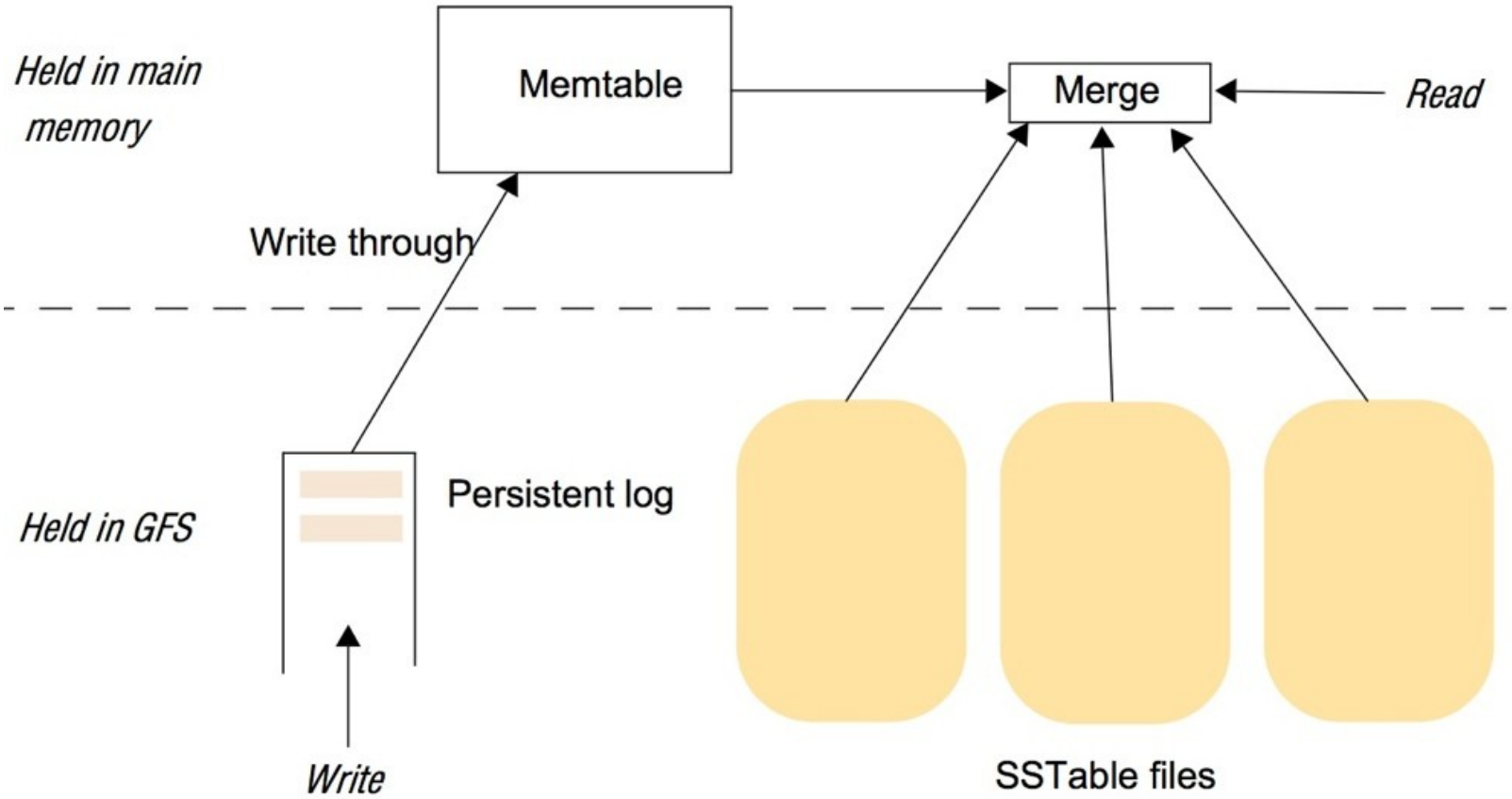
Abstracción de tabla en Bigtable



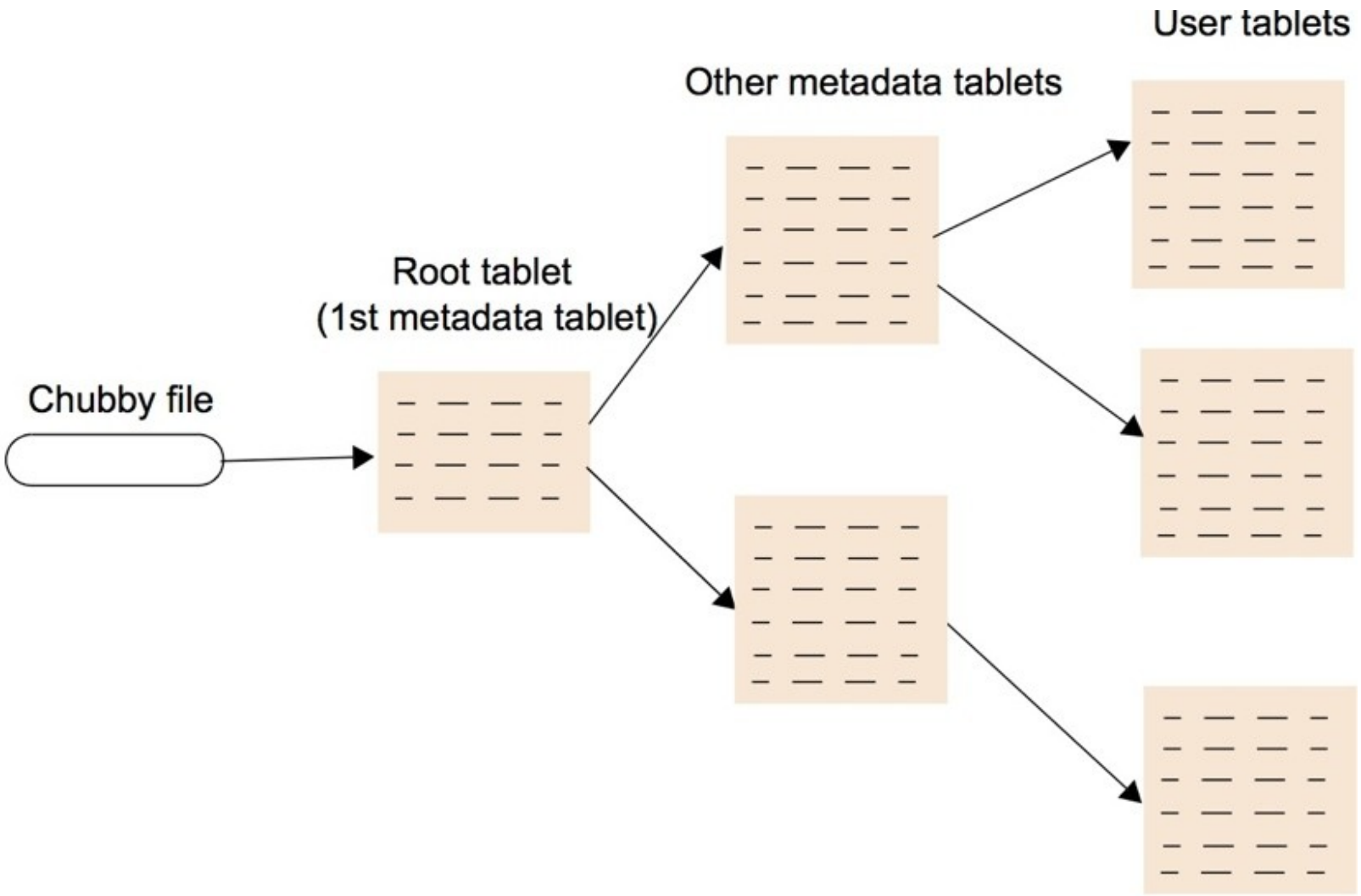
Arquitectura de Bigtable



Arquitectura del almacenamiento en Bigtable



El esquema de indexado jerárquico en Bigtable



Resumen de opciones de diseño

Element	*Design choice*	*Rationale*	*Trade-offs*
GFS	The use of a large chunk size (64 megabytes)	Suited to the size of files in GFS; efficient for large sequential reads and appends; minimizes the amount of metadata	Would be very inefficient for random access to small parts of files
	The use of a centralized master	The master maintains a global view that informs management decisions; simpler to implement	Single point of failure (mitigated by maintaining replicas of operations logs)
	Separation of control and data flows	High-performance file access with minimal master involvement	Complicates the client library as it must deal with both the master and chunkservers
	Relaxed consistency model	High performance, exploiting semantics of the GFS operations	Data may be inconsistent, in particular duplicated
Chubby	Combined lock and file abstraction	Multipurpose, for example supporting elections	Need to understand and differentiate between different facets
	Whole-file reading and writing	Very efficient for small files	Inappropriate for large files
	Client caching with strict consistency	Deterministic semantics	Overhead of maintaining strict consistency
Bigtable	The use of a table abstraction	Supports structured data efficiently	Less expressive than a relational database
	The use of a centralized master	As above, master has a global view; simpler to implement	Single point of failure; possible bottleneck
	Separation of control and data flows	High-performance data access with minimal master involvement	-
	Emphasis on monitoring and load balancing	Ability to support very large numbers of parallel clients	Overhead associated with maintaining global states

Outline

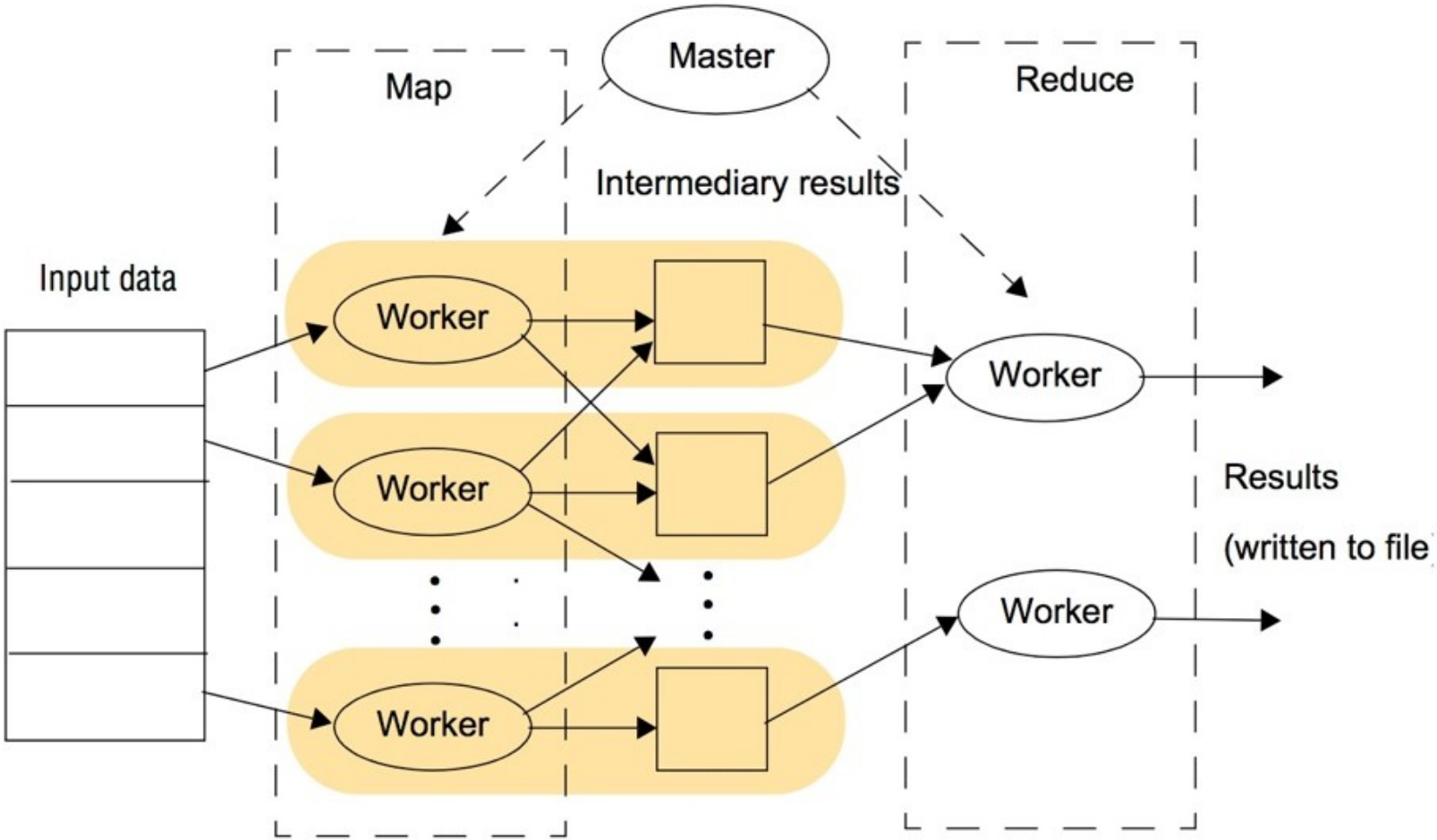
Servicios de computación distribuida

- MapReduce
- Sawzall

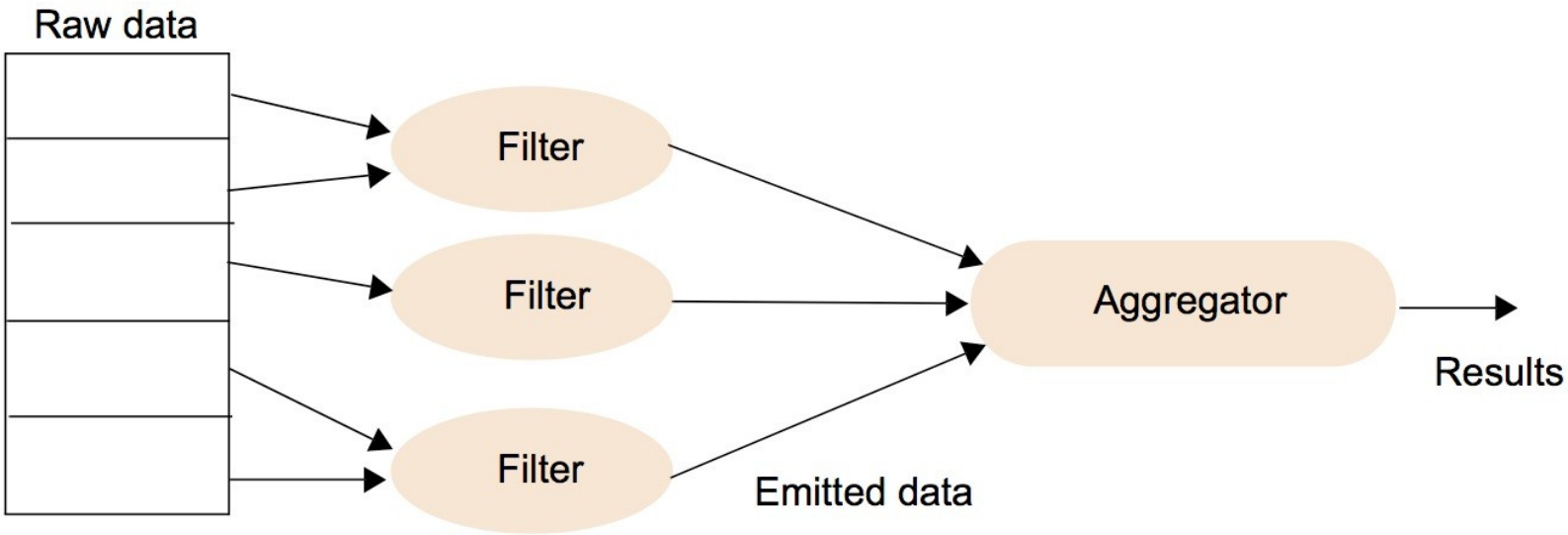
Ejemplos de uso de MapReduce

Function	*Initial step*	*Map phase*	*Intermediate step*	*Reduce phase*
Word count	*Partition data into fixed-size chunks for processing*	For each occurrence of word in data partition, emit <*word*, 1>	*Merge/sort all key-value keys according to their intermediary key*	For each word in the intermediary set, count the number of 1s
Grep		Output a line if it matches a given pattern		Null
Sort *N.B. This relies heavily on the intermediate step*		For each entry in the input data, output the key-value pairs to be sorted		Null
Inverted index		Parse the associated documents and output a <*word*, document ID> pair wherever that word exists		For each word, produce a list of (sorted) document IDs

Ejecución de un programa MapReduce



Ejecución de un programa Sawzall



Resumen de opciones de diseño

Element	*Design choice*	*Rationale*	*Trade-offs*
MapReduce	The use of a common framework	Hides details of parallelization and distribution from the programmer; improvements to the infrastructure immediately exploited by all MapReduce applications	Design choices within the framework may not be appropriate for all styles of distributed computation
	Programming of system via two operations, *map* and *reduce*	Very simple programming model allowing rapid development of complex distributed computations	Again, may not be appropriate for all problem domains
	Inherent support for fault-tolerant distributed computations	Programmer does not need to worry about dealing with faults (particularly important for long-running tasks running over a physical infrastructure where failures are expected)	Overhead associated with fault-recovery strategies
Sawzall	Provision of a specialized programming language for distributed computation	Again, support for rapid development of often complex distributed computations with complexity hidden from the programmer (even more so than with MapReduce)	Assumes that programs can be written in the style supported (in terms of filters and aggregators)