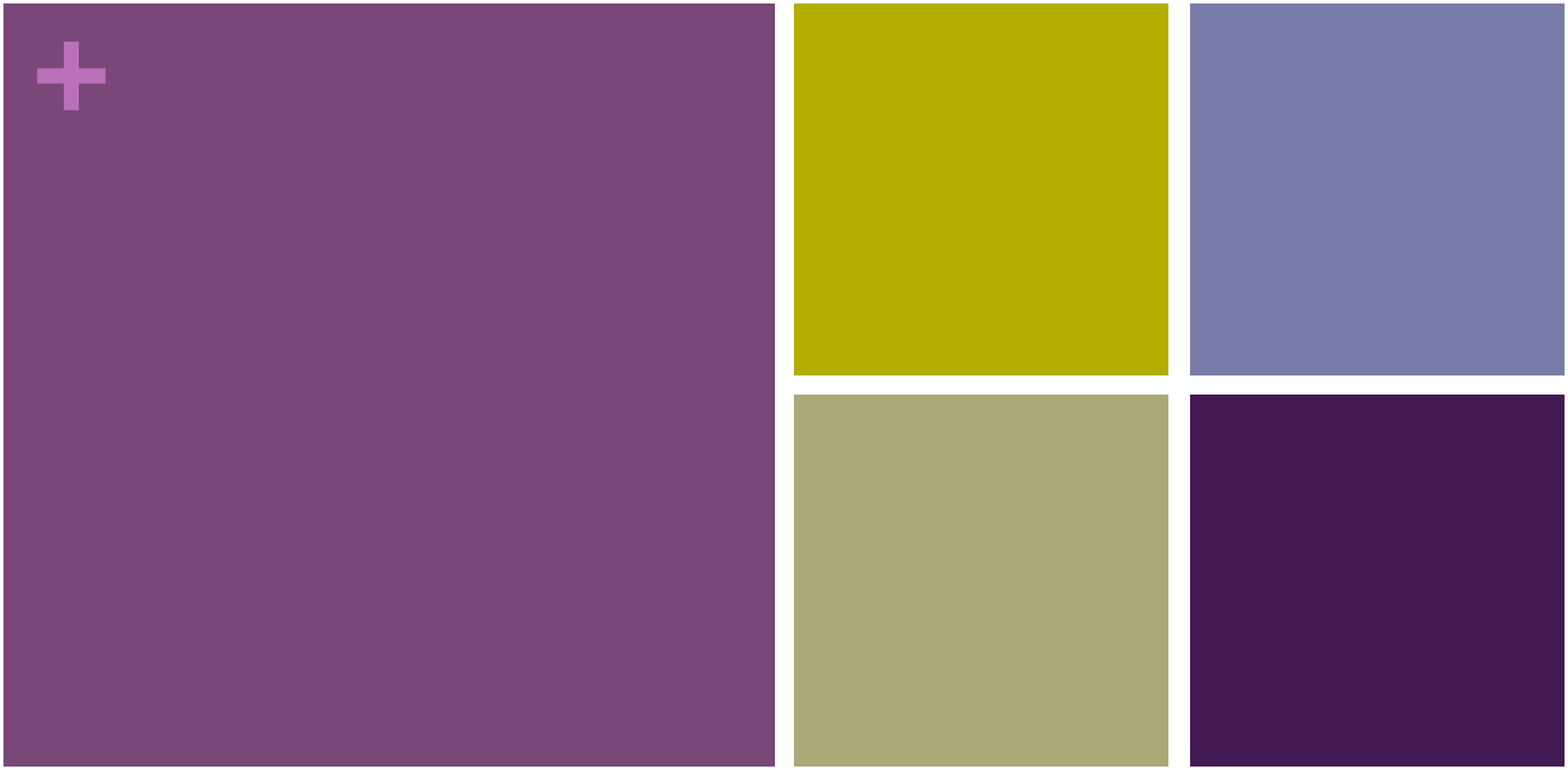


Sistemas Distribuidos

Diseño de Sistemas Distribuidos: Google

Rodrigo Santamaría

Diseño de Sistemas Distribuidos

- Introducción: Google como caso de estudio
- Servicios
- Plataforma
- Middleware

Introducción

- Crear un sistema distribuido no es sencillo
  - Objetivo: obtener un sistema consistente que cumpla con los requisitos identificados
    - Escala, seguridad, disponibilidad, etc.
  - Elección del modelo
    - Tipo de fallos que asumimos
    - Tipo de arquitectura
  - Elección de la infraestructura
    - Middleware (RMI, REST, Kademlia, etc.)
    - Protocolos existentes (LDAP, HTTP, etc.)

Introducción

Google como caso de estudio

- Google es un ejemplo de un diseño distribuido exitoso
  - Ofrece búsquedas en Internet y aplicaciones web
  - Obtiene beneficios principalmente de la publicidad asociada
- Objetivo: "organizar la información mundial y hacerla universalmente accesible y útil"
- Nació a partir de un proyecto de investigación en la Universidad de Stanford, convirtiéndose en compañía con sede en California en 1998.
- Una parte de su éxito radica en el algoritmo de posicionamiento utilizado por su motor de búsqueda
  - El resto, en un sistema distribuido eficiente y altamente escalable

Introducción

Requisitos

- Google se centra en cuatro desafíos
  - *Escalabilidad*: un sistema distribuido con varios subsistemas, dando servicio a millones de usuarios
  - *Fiabilidad*: el sistema debe funcionar en todo momento
  - *Rendimiento*: cuanto más rápida sea la búsqueda, más búsquedas podrá hacer el usuario → mayor exposición a la publicidad
  - *Transparencia*: en cuanto a la capacidad de reutilizar la infraestructura disponible, tanto internamente como externamente (plataforma como servicio)

Introducción

Requisitos: escalabilidad

- Google ve este desafío en términos de tres dimensiones
  - *Datos*: el tamaño de Internet sigue creciendo (p.ej. por la digitalización de bibliotecas)
  - *Peticiones*: el número de usuarios sigue creciendo
  - *Resultados*: la calidad y cantidad de resultados sigue mejorando
- Logros
  - 1998: sistema de producción inicial
  - 2010: $88 \cdot 10^9$ millones de búsquedas al mes
  - Sin perder eficiencia: tiempo de consulta < 0.2s
- Ejemplo [Dean 2006]
  - Asumimos que la red está compuesta de ~$20 \cdot 10^9$ páginas
    - Cada una de 20KB → tamaño total de 400TB
  - Un ordenador que lea 30MB/s tardaría 4 meses en explorarla
    - 1000 ordenadores lo harían en menos de 3h

Introducción

Requisitos: fiabilidad

- Google ofrece un nivel de fiabilidad del 99.9% en sus contratos de pago
  - Generalmente lo ha cumplido, pero en 2009, sufrió una caída de 100 minutos durante un mantenimiento rutinario
- Para mantener la fiabilidad, es necesario anticipar los fallos (HW y SW) con una frecuencia razonable
  - Técnicas de detección de fallos
  - Estrategias para enmascarar o tolerar fallos → principalmente mediante la replicación de la arquitectura física

Introducción

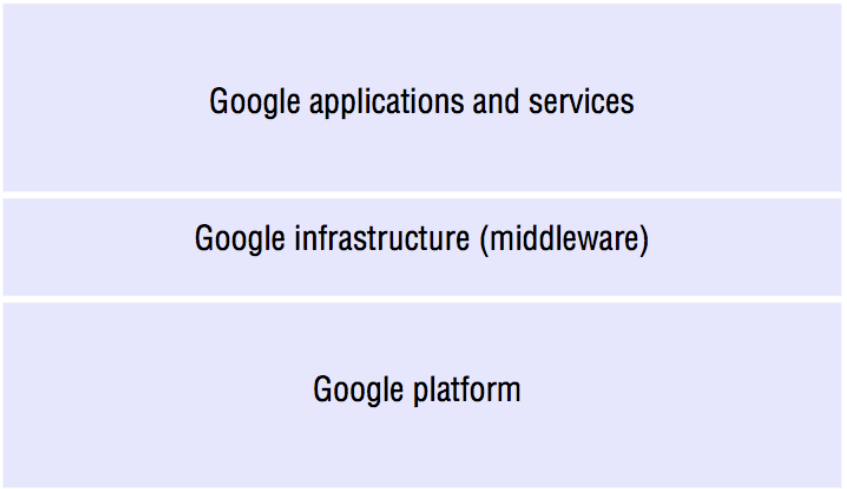
Requisitos: rendimiento

- Involucra a todas las fases de cada servicio
- Por ejemplo, en cuanto al motor de búsqueda
  - Involucra a crawling, indexing y ranking
  - Objetivo: operaciones de búsqueda < 0.2s
- Involucra a todos los recursos subyacentes
  - Red
  - Almacenamiento
  - Procesamiento

Introducción

Requisitos: transparencia

- Requisito fundamental para el uso de la plataforma de Google como servicio, entendida como:
  - Extensibilidad
  - Soporte para el desarrollo de nuevas aplicaciones
- Para ello Google desarrolla su propia infraestructura (middleware) entre la arquitectura física y las aplicaciones y servicios



Google

Principios de diseño

- **Simplicidad**: 'cada API tiene que ser tan sencilla como sea posible y no más'
  - Aplicación de la navaja de Occam
- **Rendimiento**: 'cada milisegundo cuenta'
  - Medición de los costes de operaciones primitivas (acceso a disco y memoria, envío de paquetes, etc.)
- **Testeo**: 'si no se ha roto, es que no lo has probado lo suficiente'
  - Régimen estricto de pruebas y monitorización

Diseño de Sistemas Distribuidos

- Introducción
- Servicios
  - Motor de búsqueda (crawling, indexing, ranking)
  - Cloud computing
- Plataforma
- Middleware

Motor de búsqueda

- Objetivo:
  - Dada una consulta, obtener una lista ordenada de resultados relevantes, a partir de una búsqueda en la red.
- Subsistemas del motor:
  - Crawling: obtener información
  - Indexing: procesar información
  - Ranking: clasificar información

Motor de búsqueda

Crawling

- **Tarea**: localizar y recuperar contenidos de la Web y pasarlos al sistema de indizado.
- **Ejecución**: servicio software *Googlebot*
  1. Lee recursivamente una página web
  2. Recolecta todos sus enlaces
  3. Planifica posteriores operaciones de crawling en dichos enlaces
     - Esta técnica (*deep searching*), es muy efectiva, permitiendo penetrar en prácticamente todas las páginas de la Web
- La ejecución se realiza en batería. En el pasado, el Googlebot se ejecutaba una vez cada pocas semanas
  - No es suficiente para páginas de noticias o con otro tipo de contenido cambiante (blogs, redes sociales, etc.)

Motor de búsqueda

Crawling

- Con la nueva arquitectura de búsqueda Caffeine, se introduce en 2010 una nueva filosofía de crawling/indexing
  - El crawler está está en funcionamiento continuo, y tan pronto como se explora una página, se realiza su indexado.
- Este modo reduce la antigüedad de los índices en un 50%
  - Se pasa de un procesamiento por lotes global a un procesamiento continuo incremental.
    - Es una estrategia clásica de mejora (mp3, compresión de archivos, copias de seguridad…)

Motor de búsqueda

Indexing

- **Tarea**: producir un índice de contenidos de la red similar al de un libro (título, autor, tema, etc.)
- Ejecución: *indizado* inverso de palabras
  - Dada una página web, se identifican los recursos textuales (varios formatos: html, pdf, doc) clave: posición, tamaño de letra, capitalización.
  - También se realiza un índice de enlaces encontrados en la página
- Utilizando este índice, se reduce el número de páginas candidatas de miles de millones a unas decenas de miles, según el poder discriminativo de las palabras buscadas.

Motor de búsqueda

Ranking

- **Tarea**: clasificar los índices en función de su relevancia
- **Ejecución**: algoritmo PageRank
  - Basado en los sistemas de ranking de las publicaciones científicas: un artículo científico es importante si ha sido citado por otros colegas del área.
    - Análogamente, una página es importante si ha sido enlazada por un gran número de páginas.
  - También tiene en cuenta factores relacionados con la proximidad de la búsqueda a la palabras claves de la página obtenidas en el indizado inverso

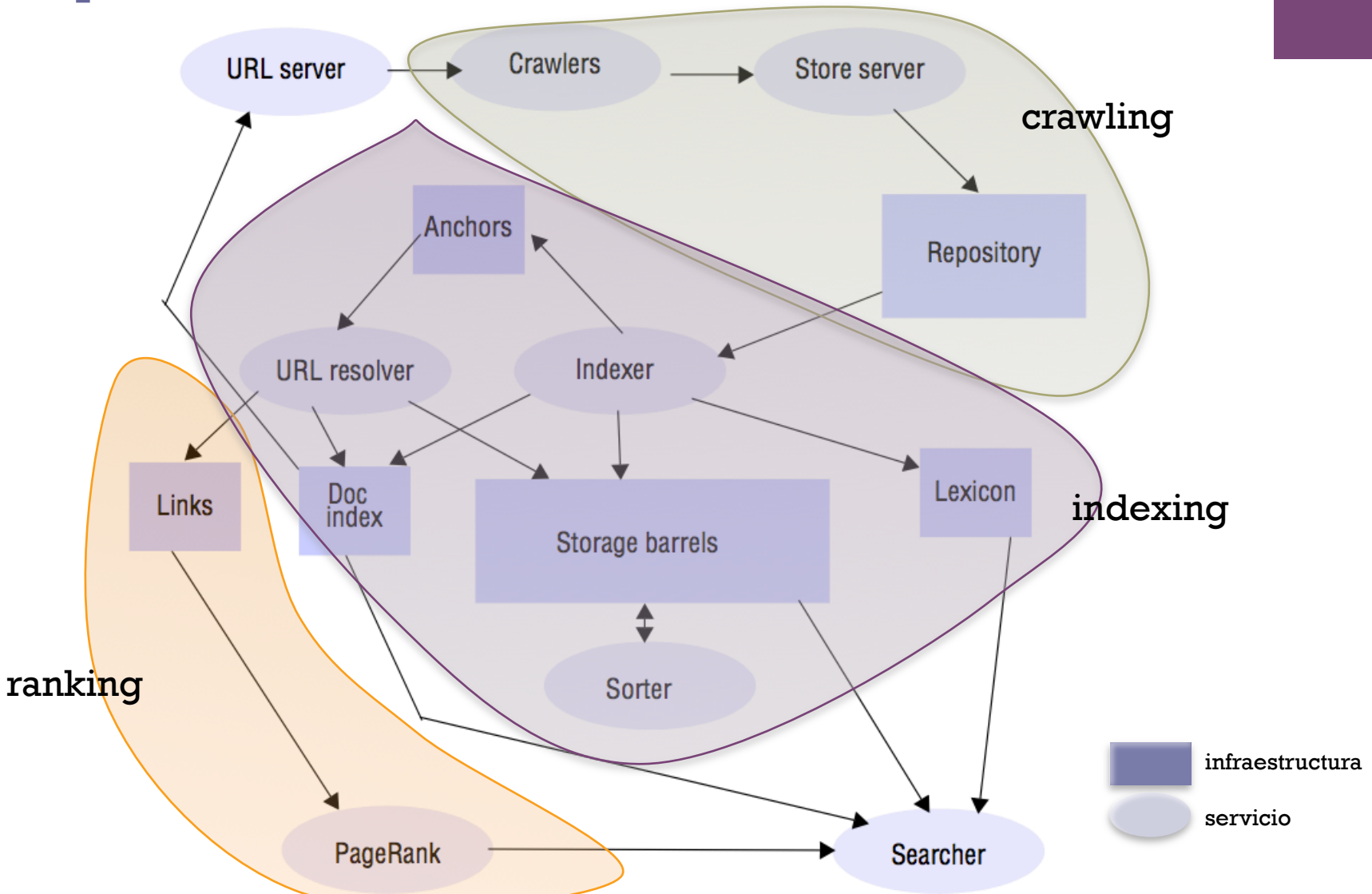
Motor de búsqueda

Ejemplo de búsqueda

- Imaginemos que buscamos "distributed systems book"
  - El buscador seleccionará páginas que contengan las tres palabras en sus índices
    - Preferentemente en el título o alguna sección importante (p. ej. listas de libros de sistemas distribuidos)
  - El buscador priorizará aquellas con un alto número de enlaces, poniendo primero las más enlazadas, y desde páginas más 'importantes':
    - *amazon*, *wikipedia*, *bookdepository* son priorizadas por su relevancia en la red.

Motor de búsqueda

Arquitectura



Modificado a partir de [Brin and Page, 1998]

Motor de búsqueda

Arquitectura

- *Repository*: datos web recogidos por el crawler, comprimidos
- *Barrel(s)*: almacén de *hits*, es decir, entradas que contienen el identificador del documento, la palabra encontrada en el documento, su tamaño de letra y capitalización
- *Sorter*: reordena el barrel por identificadores de palabra para generar el índice invertido
- *Anchors*: información sobre los enlaces en los documentos
- *URL resolver*
  - Preparación de los enlaces (*links*)
  - Obtención de nuevos enlaces para alimentar al Crawler (*Doc index*)

Motor de búsqueda

Arquitectura

- Los detalles específicos de esta arquitectura han variado, pero los servicios clave (crawling, indexing, ranking) siguen siendo los mismos.
- Modificaciones en
  - Almacenamiento optimizado
  - Bloques de comunicación más reutilizables
  - Procesamiento por lotes a continuo

Motor de búsqueda

Críticas

- Peligro de autocomplacencia [Maurer, 2007]
  - Google nos dice qué es importante → no miramos más allá
- Importancia = nº enlaces?
  - Aquello más enlazado es más famoso, no necesariamente más relevante o interesante por el usuario
  - Discriminación de los sitios nuevos
- Peligro de manipulación
  - Neutralidad: Google prioriza sus servicios en las búsquedas [NexTag, 2011]
  - Google bombing: enlazar con un determinado texto a una página
    - Enlaces 'miserable failure' a páginas relacionados con George Bush
  - Search Engine Optimization (SEO): modificar los contenidos de una página en base a los buscadores y los usuarios (~marketing)

Google

Otros servicios: Cloud computing

- Modificación de un cliente ligero que permite usar archivos y aplicaciones remotas sin perder autonomía.
- Software como servicio: Google Apps
  - Alternativa a las suites de escritorio:
    - Gmail, Google Docs, Google Sites, Google Calendar
    - Google Maps, Google Earth
- Plataforma como servicio: Google App Engine
  - Pone a disposición del usuario parte de la infraestructura distribuida de Google para sus Google Apps y su servidor de búsqueda
    - Para que pueda desarrollar sus propias aplicaciones web mediante su plataforma
    - Mediante el uso de APIs

Diseño de Sistemas Distribuidos

- Introducción
- Servicios
- Plataforma
  - Modelo físico
  - Componentes
- Middleware

Arquitectura

Modelo físico

- Utilización de gran número de PCs como base para producir un entorno para computación y almacenamiento distribuido
  - Cada unidad de PC utilizada cuesta entorno a 1000$
  - Típicamente tiene 2TB de disco y 16GB de DRAM
  - Corre con una versión reducida del kernel de Linux
- Modelo de fallos
  - Al utilizar PCs 'de coste', hay un riesgo de fallos
  - Según [Hennessy and Patterson, 2006]
    - 20 máquinas tienen fallos software cada día (se reinician manualmente!)
    - 2-3% de los PCs tienen un fallo hardware al año (el 95% son fallos en los discos o en la DRAM) → Un fallo HW por cada 10 fallos SW.
  - Se diseñarán estrategias de tolerancia de fallos

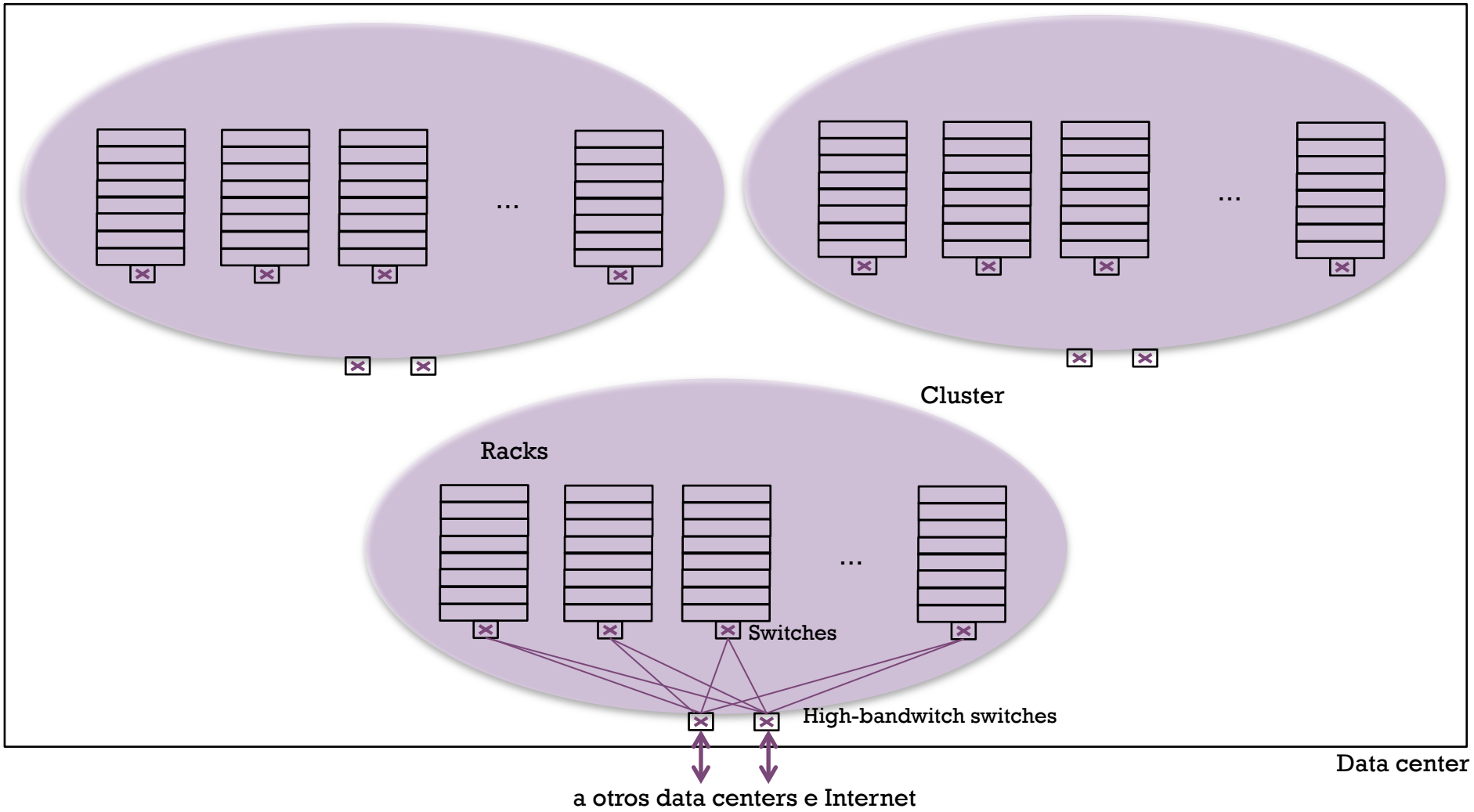
Arquitectura

Componentes [Hennessy and Patterson, 2006]

- Rack
  - Entre 40 y 80 PCs
  - Switch Ethernet que provee conexión en el rack y hacia fuera
- Cluster
  - 30+ racks
  - 2 switches de banda ancha (redundancia)
  - Unidad básica de gestión
- Data Center
  - Decenas repartidos por el mundo
  - Cada uno hospeda uno o más clusters.

Arquitectura

Componentes



Arquitectura

Data Centers

- 12 dedicados, 24 compartidos (2008)

http://royal.pingdom.com/2008/04/11/map-of-all-google-data-center-locations/

http://www.google.com/about/datacenters/locations/index.html

Arquitectura

Capacidad de almacenamiento

- Un PC → 2 terabytes
- Un rack de 80PCs → 160 terabytes
- Un cluster de 30 racks → 4.8 petabytes
- Asumiendo unos 200 clusters entre todos los data centers
  - → 960 petabytes ~ 1 exabyte (10^{18} bytes)

Diseño de Sistemas Distribuidos

- Introducción
- Servicios
- Plataforma
- Middleware
  - Comunicación
  - Almacenamiento
  - Computación

Middleware

- Para poder ofrecer los servicios ofertados a partir de la plataforma distribuida disponible, hace falta toda una infraestructura (middleware) intermedia
- Por lo general, Google implementa sus propias soluciones middleware en vez de utilizar estándares
  - Son muy parecidas a las soluciones existentes
  - Pero están optimizadas para los requisitos de servicio y las características de la plataforma
- En esta sección veremos por encima cómo es este middleware en términos de comunicación, almacenamiento y computación distribuida

Middleware

Secciones

- **Paradigmas de comunicación**: invocación remota y multicast
  - *Protocol buffers*: ofrecen un formato de serialización común
  - *Publish-subscribe*: servicio para la diseminación de eventos
- **Datos y coordinación**: almacenamiento y acceso coordinado a datos
  - *GFS*: sistema de ficheros distribuido para grandes volúmenes de datos
  - *Chubby*: almacenamiento de volúmenes pequeños de datos y coordinación
  - *Bigtable*: base de datos distribuida construida sobre GFS/Chubby
- **Computación distribuida**: paralelización sobre la arquitectura física
  - *MapReduce*: computación distribuida sobre conjuntos grandes de datos
  - *Sawzall*: lenguaje de alto nivel para la ejecución de computaciones distribuidas

Middleware

Comunicación – protocol buffers

- Siguiendo el principio de simplicidad, Google adopta un servicio de invocación remota mínimo y eficiente
- Para ello utiliza los búferes de protocolo (*protocol buffers*), un mecanismo general de serialización para
  - Almacenamiento, más sencillo que XML
  - Comunicación, intercambiándose mediante RPC
- Los búferes de protocolo son neutrales respecto a
  - Plataforma
  - Lenguaje
  - Protocolo RPC

Middleware

Comunicación – protocol buffers

- Utilizan un lenguaje sencillo de especificación de *mensajes*
- Cada *mensaje* se parece a un objeto:
  - Con campos enumerados
    - Secuencialmente
  - Y etiquetados
    - Política de existencia (repetido, requerido, opcional)
  - Con mensajes anidados
- Para cada campo, el compilador genera métodos de gestión
  - exists, clear, set, get

```
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;
}
```

Middleware

Comunicación – protocol buffers

- Los búferes de protocolo son más sencillos que XML
  - No son tan abundantes en tags
    - 1/3 a 1/10 del tamaño en XML
  - Métodos de acceso y numeración secuencial
    - 10 a 100 veces más rápidos en operación y acceso
- La comparación no es del todo justa
  - XML tiene un propósito general
    - Tiene una semántica mucho más rica
    - Busca la interoperabilidad entre todo tipo de sistemas

Middleware

Comunicación – protocol buffers

- Los búferes de protocolo se intercambian mediante RPC mediante una sintaxis adicional para generar interfaces y métodos

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

- Interfaz de un servicio de búsqueda *SearchService*
  - Con una operación remota *Search* que toma un parámetro del tipo *RequestType* y retorna uno de tipo *ResponseType*
- Sólo se puede enviar un parámetro y retornar otro
  - Simplicidad
  - Filosofía tipo REST: operaciones simples, manipulación de recursos
- El compilador se encarga de generar los stubs para la comunicación RPC a partir del código

Middleware

Comunicación – publish-suscribe

- Ampliación de los protocol buffers para tareas de diseminación de eventos
  - RPC tiene un rendimiento bajo para este tipo de tarea
    - Necesidad de conocer la identidad de todos los destinatarios
- Google no ha hecho públicos los detalles del sistema pero básicamente es un sistema de notificación de eventos
  - Un servicio publica los eventos que genera
  - Los suscriptores expresan su interés en determinados eventos
  - El sistema asegura la entrega de las notificaciones de eventos del servicio a sus suscriptores

Middleware

Almacenamiento - GFS

- Google File System es un sistema de archivos distribuido
  - Similar a otros de propósito general como NFS o AFS
  - Diseñado para los requisitos de Google:
    - Debe funcionar de manera fiable sobre la arquitectura física
      - Los componentes (HW y SW) pueden fallar
    - Optimizado para los ficheros utilizados (no muy numerosos, pero de gran tamaño)
      - Millones de ficheros de tamaño medio de 100MB
      - Algunos ficheros superando 1GB
    - Optimizado para el tipo de consultas
      - Consultas y modificaciones secuenciales, no aleatorias
      - Nivel de acceso concurrente alto

Middleware

Almacenamiento - Chubby

- Servicio con varias capacidades
  - Provee acceso sincronizado a actividades distribuidas
    - Bloqueos, semáforos, exclusión mutua, etc.
  - Provee un sistema de ficheros para ficheros de tamaño pequeño
    - Complementa a GFS
  - Provee un servicio de nombres dentro de Google
- Puede parecer que no cumple el principio de simplicidad pero en el fondo, todas sus capacidades se derivan de un servicio central de *consenso distribuido,* Paxos

Middleware

Almacenamiento – Chubby/Paxos

- El algoritmo Paxos para consenso distribuido se basa en la elección de un coordinador
  - El consenso está en la elección del coordinador, a partir de ahí será el coordinador el que tome la decisión
  - Se asume que el coordinador puede fallar
  - Paxos implementa un proceso de elección flexible que puede resultar en varios coordinadores coexistiendo (el nuevo y el/los viejo/s).

Middleware

Almacenamiento – Chubby/Paxos

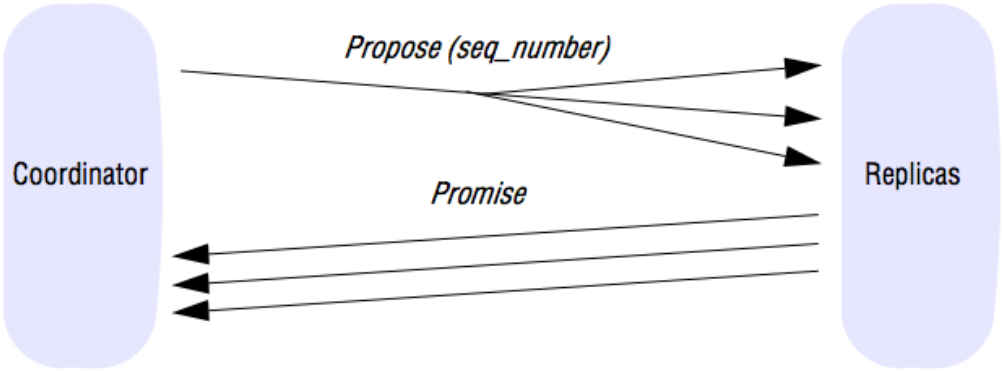
- Elección del coordinador
  - Sean *n* procesos (*réplicas*) con identificadores únicos i_r
  - Cada coordinador recibe un identificador (*nº de sec.*) único
    - Cada réplica mantiene como coordinador el nº de secuencia más alto de los que ha recibido (s_r)
  - Si hay una elección, cada réplica *r* se 'presenta' con un nº de secuencia s'_r
    - $s'_r > s_r$ (mayor que el coordinador 'viejo')
    - $s'_r \bmod n = i_r$ (único entre todas las réplicas)
  - *p* hace un broadcast de un mensaje *propose* con s'_r
  - Las otras réplicas responden
    - *promise* si votan por dicho coordinador (no han recibido un nº mayor)
      - Si tiene constancia de algún valor para el consenso, lo envía
    - *nack* si no votan por dicho coordinador
  - Si recibe una mayoría de mensajes *promise*, es elegido coordinador
    - El conjunto de todas las réplicas que votaron a favor es el *quorum*

Middleware

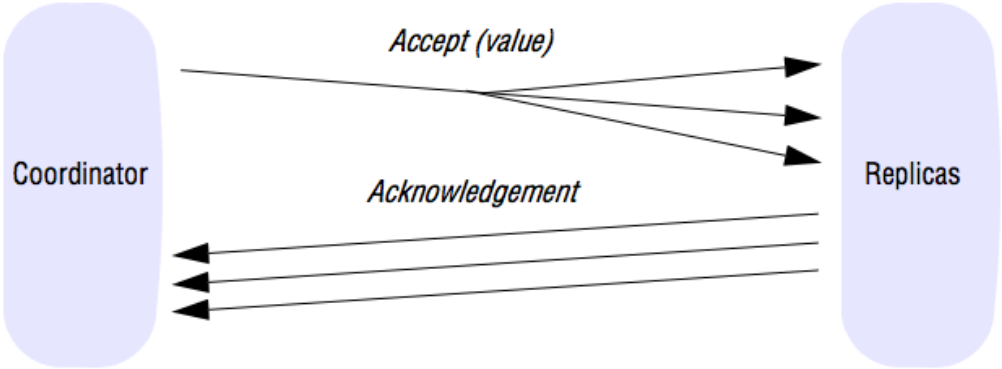
Almacenamiento – Chubby/Paxos

- Elección del valor
  - El coordinador electo selecciona un valor
    - Si uno o más miembros del quorum le envió un valor en su mensaje *promise*, el valor que elija el coordinador debe ser uno de ellos
    - Si no, el coordinador puede elegir el valor que quiera
  - El coordinador envía *accept* con el valor elegido al quorum
    - Todo miembro del quorum debe enviar un *ack*
  - El coordinador espera (indefinidamente) a recibir una mayoría de *acks*
    - Si una mayoría de réplicas ha hecho *ack*, se ha logrado el consenso. El coordinador hace un broadcast con el mensaje *commit* para notificar a todas las réplicas el acuerdo
    - Si no, el coordinador abandona la propuesta y el proceso reinicia

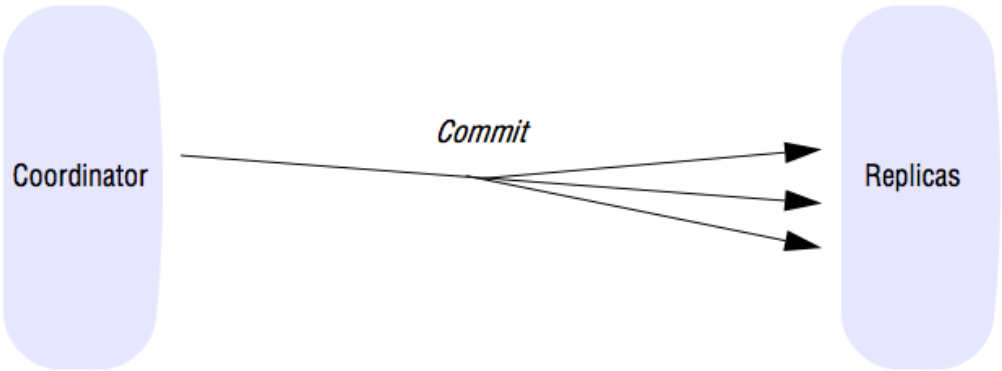
Step 1: electing a coordinator



Step 2: seeking consensus



Step 3: achieving consensus



Middleware

Almacenamiento – Chubby/Paxos

- Consenso
  - En ausencia de fallos, está asegurado
  - Con fallos (caída del coordinador o de una réplica, o mensajes perdidos/duplicados) también se asegura
    - Por la elección de números de secuencia al elegir coordinador
    - Por el mecanismo de quorum: si se acuerdan dos mayorías, debe haber al menos una réplica en común que votó por ambas
- Terminación
  - El algoritmo podría no terminar si dos candidatos compiten incrementando continuamente sus números de secuencia

Middleware

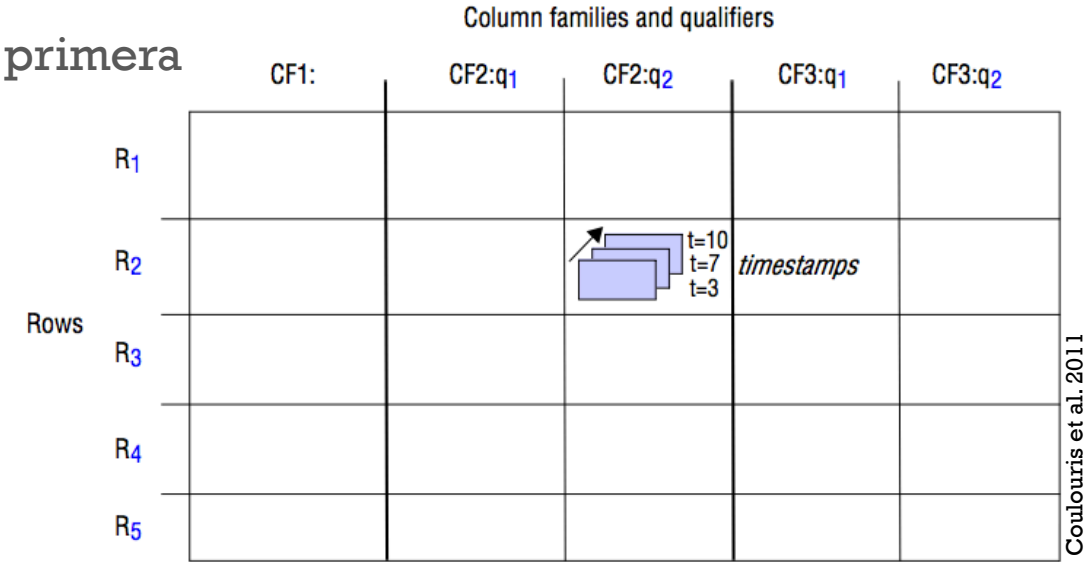
Almacenamiento – Bigtable

- GFS es un sistema de almacenamiento de ficheros grandes
  - Pero 'planos', no indexados, accedidos secuencialmente
- Necesidad de acceso estructurado a los datos
  - Pero una base de datos distribuida es difícil de escalar y tiene bajo rendimiento, y no se necesita toda su funcionalidad
- Bigtable mantiene el modelo de tablas de una BD, pero simplifica la interfaz, centrándose en un almacenamiento y recuperación eficientes.
  - Cada tabla suele estar en el rango de los terabytes
  - Las tablas estarán almacenadas mediante GFS/Chubby

Middleware

Almacenamiento - Bigtable: interfaz

- Cada tabla se accede a través de tres dimensiones
  - Fila: indica el objeto
    - P. ej. la página web *www.bbc.co.uk/sports*
  - Columna: indica el atributo
    - P. ej. los *contenidos*, los *enlaces* o el *lenguaje*
  - Timestamp: indica la versión
    - Siendo la más reciente la primera



Coulouris et al. 2011

Middleware

Almacenamiento - Bigtable: infraestructura

- Una bigtable ocupa varios TB
  - Para su almacenamiento, se divide en *tabletas* de 100-200MB
  - Cada tableta se almacena como un conjunto de ficheros en un determinado formato
- La infraestructura de Bigtable se encarga de la división/ reconstrucción en tabletas
- La infraestructura de GFS se encarga del almacenamiento de las tabletas
- La infraestructura de Chubby se encarga de la monitorización de sus estados y del balanceo de carga

Middleware

Computación - MapReduce

- La infraestructura de almacenamiento y comunicación maneja grandes conjuntos de datos distribuidos
  - Necesitamos operar con esos datos de manera distribuida
- MapReduce es un modelo de programación para aplicaciones que oculta al programador los aspectos distribuidos
  - Paralelización
  - Recuperación de fallos
  - Administración de los datos
  - Balanceo de carga

Middleware

Computación - MapReduce

- MapReduce sigue el patrón clásico en la computación en paralelo:
  - Dividir los datos (complejos) en trozos (simples)
  - Realizar el cómputo de los trozos simples (obteniendo resultados intermedios separados)
  - Unir los resultados intermedios en el resultado final
- Ejemplo: sumar todos los elementos de una matriz $10^6 x 10^6$
  - Dividimos la matriz en 10^6 matrices 1000 x 1000
  - 10^6 procesadores calculan las sumas parciales
  - Se hace la suma total a partir de las 10^6 sumas parciales

Middleware

Computación - MapReduce

- MapReduce utiliza para la paralelización cuatro pasos
  - Divide los datos en trozos de igual tamaño
  - *Map*: para cada trozo realiza la operación paralelizada, produciendo uno o más pares <clave, valor>
  - Se ordenan los pares intermedios según sus claves
  - *Reduce*: se obtiene el resultado final fusionando los pares intermedios
- Ejemplo: contar el nº de veces que aparecen 1+ palabras
  - Se divide el conjunto de textos en el que buscar
  - *Map*: para cada subconjunto, cada vez que se encuentre una de las palabras buscadas, se emite un par <palabra, 1>
  - Se ordenan todos los pares según la clave (palabra)
  - *Reduce*: Se cuenta el nº de pares para cada palabra

Middleware

Computación - MapReduce

- MapReduce se implementa en una biblioteca que oculta los detalles asociados con la paralelización y la distribución
  - Construida sobre RPC y GFS principalmente
  - Es habitual usar como entrada y salida tablas de Bigtable
  - El programador sólo tiene que centrarse en especificar las funciones *map* y *reduce*
- Este modelo de programación ofrece varias ventajas
  - Simplifica el código de los programas
    - El indexado pasó de 3800 a 700 líneas de código al adoptarlo
  - Facilita la actualización y comprensión de los algoritmos
  - Separa la lógica de la aplicación de las tareas de distribución

Middleware

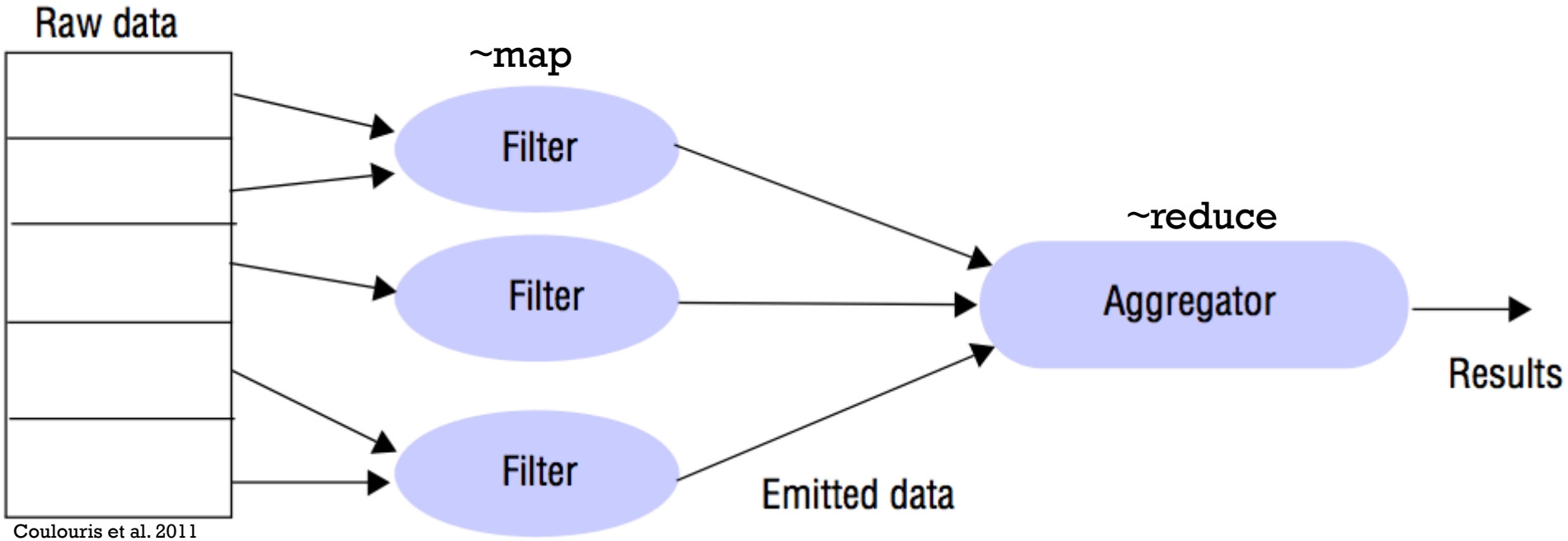
Computación - Sawzall

- Es un lenguaje de programación para realizar análisis de datos en paralelo en la arquitectura física de Google
- Se basa en su infraestructura
  - MapReduce para ejecuciones en paralelo
  - GFS para el almacenamiento de datos
  - Protocol buffers para obtener un formato común
- Simplifica la construcción de programas más allá de MapReduce
  - Programas de 10 a 20 veces más pequeños
  - Utiliza un patrón de paralelización similar

Middleware

Computación - Sawzall

- Los datos de entrada o *raw data* están compuestos de *registros*, que son separables y procesables en paralelo por *filtros,* produciendo datos *emitidos*
- Los *agregadores* fusionan los datos emitidos en el resultado



Coulouris et al. 2011

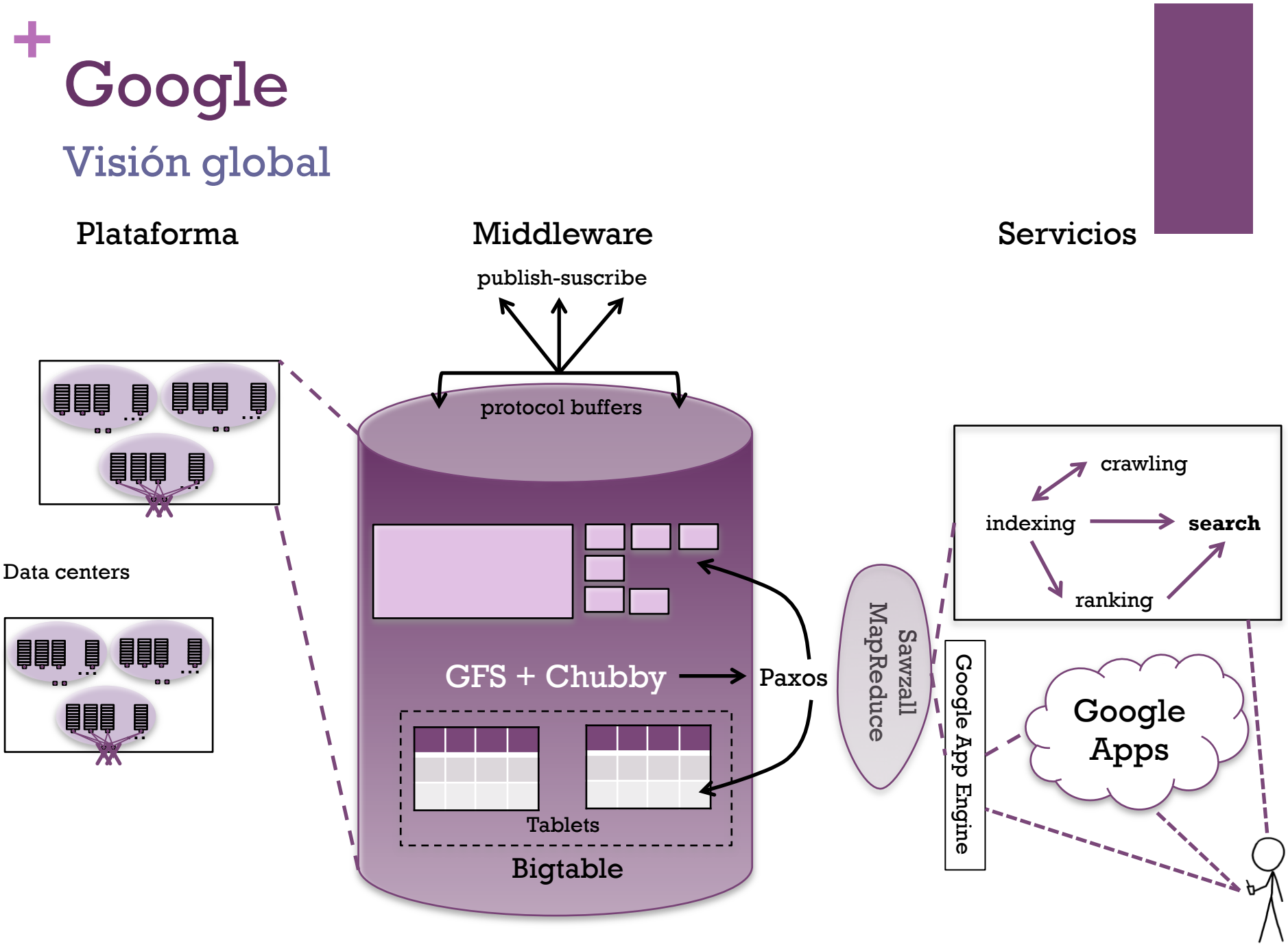
Middleware

Computación - Sawzall

- Las ventajas sobre MapReduce vienen principalmente de tener una sintaxis más amplia que las funciones map/reduce
- Por ejemplo, existe un conjunto de agregadores predefinido tales como
  - *sum* para sumar todos los valores emitidos
  - *collection* para construir un conjunto con los valores emitidos
  - Otros de naturaleza estadística, como *quantile* o *top*

Google

Visión global



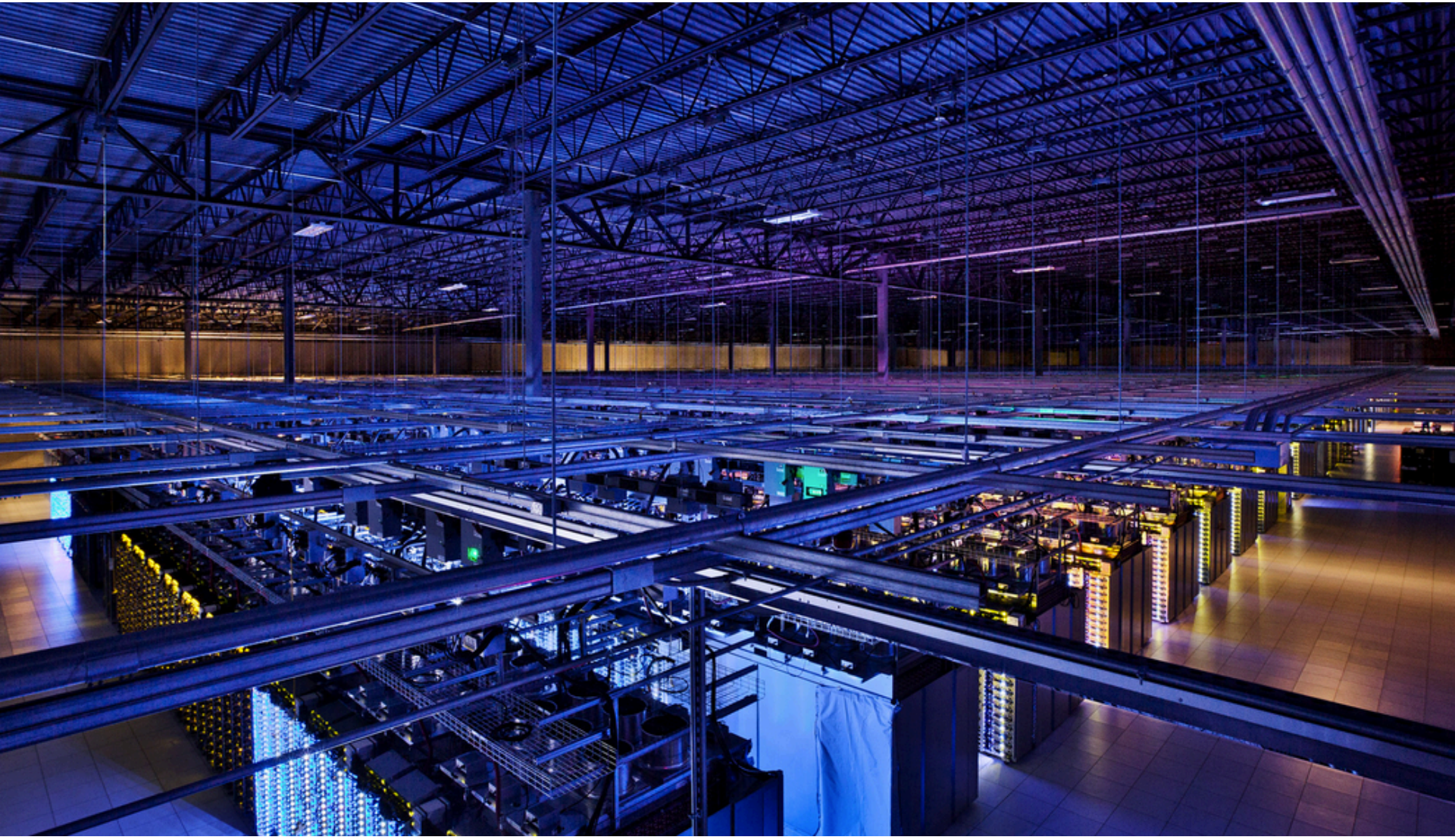
Google

Conclusiones

- La infraestructura de Google es un buen ejemplo de casi todos los problemas distribuidos, y de su resolución mediante distintas aproximaciones
- Al diseñar un sistema distribuido es esencial
  - Identificar los requisitos primordiales para nuestro sistema
  - Diseñar una plataforma acorde a dichos requisitos
  - Aprovechar las soluciones distribuidas existentes
    - O bien la teoría detrás de ellas, para generar nuestras propias soluciones
- Google diseña sus propias soluciones distribuidas basándose en soluciones conocidas, poniendo máxima atención en la escalabilidad y la fiabilidad.

Referencias

- G. Colouris, J. Dollimore, T. Kindberg and G. Blair. *Distributed Systems: Concepts and Design (5th Ed)*. Addison-Wesley, 2011
  - Ch. 21
- http://googleblog.blogspot.com



Google Data Center en Council Bluffs, Iowa

http://www.google.com/about/datacenters/gallery/#/tech/2195/