Data Intensive Applications

Datos en Sistemas de Gran Escala

- **NoSQL mejor** que SQL. **Flexibilidad** de esquemas.
- Enfoques transaccionales:
  - **Online Transaction Processing (OLTP).** Orientado a unidades lógicas, transacciones.
  - **Online Analytics Processing (OLAP).** Orientado a análisis del conjunto de datos.
- **Almacenamiento:**
  - Relacional.
  - Columnar.
  - Cubos de Información.
    - Vistas con pre-cálculos estadísticos.
    - Agrupaciones por diferentes dimensiones.

Replicación

- **Leader based:**
  - **1 lider RW.**
  - **N followers RO.**
  - Replicación (**mirroring**) síncrona vs. asíncrona.
  - Problemas:
    - *read your own writes*,
    - *monotonic reads*.
- **Multi-leader based:**
  - **+Tolerancia a fallos.**
  - Problema agregado: *conflictos por occurencia*.
- **Leaderless based:**
  - Totalmente **distribuido**.
  - Sincronización mutua.
  - Topologías.
  - **Conflictos muy frecuentes** si no hay particionado.
  - Alternativa: **consenso** para escrituras (paxos?).

Particionamiento

Motivaciones

- **Performance.** Aumentar velocidades de R/W.
- **Conflictos.** Evitar colisiones/resoluciones de conflictos.
- **Redundancia.** Tolerancia a fallos.

Tipos de particionado

- **Horizontal** (por filas). Registro en UNA partición a la vez.
- **Vertical** (por atributos/dimensiones/campos). Registro en TODAS las particiones a la vez.

Función de partición

- Por **Key-Value**.
- Por **Key-Range**.
- Por **Hash-of-Key**.
  - Escribir todo a la vez.
  - Buscar buena distribución de datos entre todas las particiones.
- Mixtos:
  - Generar **N shards** para cada key. Da igual *en cuál* guardo.

  - Partición por claves secundarias.

Enrutamiento

- Si **conozco la función de particionamiento**, ir directamente.
  - Ojo: implica conocer la dirección de cada partición, etc etc.
- Si **no la conozco**, algo extra del lado del usuario.
  - Opción 1: ir a cualquier partición, si es miss, **ser redireccionado**.
  - Opción 2: ir a un **centinela** que sabe donde esta la data.

Distributed Shared Memory (DSM)

- **Objetivo:** ilusión de **memoria compartida centralizada**.
- **Ventajas:**
  - Intuitivo (facilita distribuir algoritmos no distribuidos).
  - Compartir información entre nodos que no se conocen.
- **Desventajas:**
  - Desalienta la distribución,
  - genera latencia,
  - cuello único de botella (SPOF).

Implementación naive

- Información en **memoria** de un **servidor** (*memory pages*).
- Clientes acceden mediante **requests**.
- Consistencia = **serialización de requests**.
- Baja performance p/ clientes.

Migración de Memory Pages

- Información en memoria del servidor, **delegada en los clientes**.
- Optimizar **localidad de acceso** pidiendo memory page **prestada**.
  - Otro pedido de la misma página queda bloqueado.
  - Alternativa: permitir sub-delegación.
- Consistencia garantizada.

Replicación de Memory Pages

RO

- Favorece escenario **muchas lecturas pocas escrituras**.
- **Leer** => replicación en modo RO.
- **Escrituras coordinadas** por servidor.
  - Ante cambios, invalida réplicas.

R/W

- Servidor tiene las páginas en memoria h/ ser pedidas.
- Cliente toma **control total** sobre la página replicada.
- Servidor => **secuenciador de operaciones**.
- Servidor aplica cambios ante caídas.

Distributed File Systems (DFS)

Motivaciones

- **Esquema centralizado de información persistente.**

  - Control de backups, acceso, y monitoreo.
- **Optimización de recursos** gracias a concentración.

Factores de diseño

- **Transparencia a los clientes.**
  - **Acceso** con credenciales.
  - **Localización.** Operar archivos como si fueran locales.
  - **Movilidad** interna no debe ser percibida.
  - **Performance y Escala.** Optimizaciones no afectan.
- **Concurrencia.** Sin operaciones adicionales.
- **Heterogeneidad de Hardware.** Adaptación.
- **Tolerancia a Fallos.** Permitir `at-least-once` o `at-most-once`.

Casos de estudio

Network File System (NFS)

- Objetivo: **independencia de plataformas**.
- Requiere abstracción del kernel: **VFS (Virtual File System)**.
- Arquitectura C-S utilizando RPC sobre TCP/UDP.
- Acceso a archivos mediante VFS.
  - Requiere invocación remota.
- Soporta POSIX.

Hadoop DFS (HDFS)

- Objetivo: **hardware de bajo costo**.
- Basada en GFS.
- No soporta POSIX => considerado **Data Storage**.

Factores de diseño **"Moving computation is cheaper than moving data."** (a computation requested by an application is much more efficient if it is **executed near the data** it operates on).

- **Tolerancia a fallos.** Adaptarse a fallos de HW.
- **Volumen y Latencia.** Favorece *streaming* y archivos volumétricos.
- **Portabilidad.** Utiliza TCP entre servidores y RPC con clientes.
- **Performance.** Favorece operaciones de lectura.

Arquitectura

- **Master-Slave.**
- **Namenode:** contiene metadata de archivo, coordina los datanodes.
- **Datanodes:** almacena datos de archivo *(file blocks)*.
- Cliente consulta namenode por el FS y la ubicación.
  - Luego, comunicación directa.

Almacenamiento

- Bloques de **tamaño fijo**.
- **Replicados** en distintos datanodes.
- Metadata mantenida en memoria, con **log de transacciones**.
- Cluster permite **re-balanceo de bloques**.

Big Table

- Claves, datos, columnas.
  - Almacena Clave-Datos.
  - Datos = conjunto de columnas.
  - Conjunto **disperso**.
- **Tablets:** conjunto de filas consecutivas s/ clave.
  - Unidad de balanceo.
  - Permite escalar el sistema.
  - Auto-splitting.
- **Jerarquía de tres niveles:** root metadata.

Arquitectura

- **Master.**
  - Locks en Chubby.
  - Asignación de tablets.
- **Tablet Servers** (R/W directo con clientes).

Auto-splitting

- Exitoso: **randomly distributed keys**.
  - R/W se handlean por rangos, por ej.
- No exitoso: **monotically increasing keys**.
  - Writes siguen yendo al mismo!