Apuntes 1: Viernes 7 de agosto

Diego Granados Retana 2022158363

Bases de datos II

14 de agosto, 2023

Apuntes de la clase:

En una computadora intervienen demasiados componentes. Tenemos el disco, que es el componente más lento de la computadora. Hace algunos años cuando solo teníamos discos mecánicos que daban vuelta y teníamos que bajar la cabeza lectora, el disco todavía era demasiado lento con respecto a los otros componentes. El file system es un árbol n-ario que define cuántos archivos puedo tener y qué tamaño tienen. Usamos un pedazo del disco para memoria virtual. En una parte del disco se encuentra formateado cómo van a quedar los datos en la memoria. Me permite tener los datos contiguos para encontrarlos rápidamente.

Si tenemos un archivo conformado por tres bloques de información, y nos piden una página de ese archivo donde un pedacito está en un bloque y otro pedacito está en otro, necesitaríamos varias búsquedas a disco.

Todo programa que se hace en un sistema operativo utiliza un file descriptor, es un identificador único. Todo programa funciona como emisor y receptor. Todo programa recibe datos, normalmente un standard input. El standard input es usualmente el teclado. Se usa un file descriptor. Todo programa va a tener un file descriptor para el standard input. El standard output, en los inicios de la arquitectura de computadoras, es el monitor. Vamos a tener un stream que se conoce como stderror, que es el manejo de errores de un programa. Cuando hacemos un print desde un programa que creamos, siempre trata de salir por el standard output. Cuando generamos un error, se despliega en el standard error. Cualquier conexión socket o archivo que abramos se maneja con file descriptors. Estos dispositivos se relacionan con un stream.

Cuando tengo un programa que dura mucho tiempo haciendo una consulta o sirviendo un dato que le está pasando una base de datos, normalmente un programa o thread que esté atendiendo un cliente, al quedarse en memoria principal porque la computación del resultado dura mucho, lo único que está haciendo es usar memoria, procesador y file descriptors. Todo esto hace que la base de datos se vuelva lenta si sigo abriendo procesos. Por ejemplo, cuando se hace un query muy pesado, que tarda segundos o minutos en responder, muchos queries en paralelo me van a bloquear la base de datos o agotar los recursos de la base. Los file descriptors son limitados en la computadora. Cada elemento en un sistema operativo que recibe y emite un stream de datos, va a consumir un file descriptor. El SO tiene un número máximo de file descriptors que tiene. Esto puede ser configurable en algunos sistemas operativos. Después, uno puede asignar cuántos file descriptors puede tener abiertos una base de datos.

Cuando instalamos la base de datos con helm charts, este automáticamente hace esto por nosotros.

Cuando ejecutamos un programa el OS se trae el código al user space en la memoria, lo mete en la lista de programas que están esperando el CPU, la cual más fácilmente funciona como un round robin, el CPU pide la página de memoria en la caché, eso genera un caché miss, eso hace que el OS suba la página a la caché para que el programa se pueda ejecutar. Si sí está en caché, el CPU recibe un caché hit, entonces se puede ejecutar.

Cuando la página no está en la memoria ni en la caché, se genera un page fault. Esto levanta al OS, el cual le instruye al DMA que traiga la página a memoria para que pueda seguir el proceso de ejecución.

Cuando tengo procesos que son intensivos en cuanto al uso de CPU, son CPU bound. Ejemplo: Si tenemos consultas a la base de datos, que en las consultas a la base de datos hacemos demasiados procesamiento lógico-matemáticos, como un agrupamiento y después una suma. Eso consume muchas entradas de cálculo a nivel de CPU. Un proceso CPU bound siempre va a tener un requerimiento alto de CPU. No existe una receta para definir cuándo uno ocupa un buen CPU o más memoria. Tenemos que analizar lo que conocemos como workload y use case. El workload me dice el tipo de queries y el use case el tipo de datos y organización que voy a tener en la base de datos. Si el workload necesita mucho procesamiento, esto va a necesitar mucho CPU.

El concepto de I/O Bound se refiere a un proceso que necesita interactuar demasiado con disco. En SO hablamos de un proceso que necesita interactuar mucho con la red. En un sistema distribuido de bases de datos, tiene que interactuar mucho con la red, y si es una BD que está en una sola máquina, se refiere a que tiene que acceder mucho a disco. Hay un use case donde una tabla tiene un histórico de transacciones y un usuario pide que devuelvan todas las transacciones que hay en la tabla. Probablemente la tabla va a ser más grande que la memoria principal. Vamos a tener que empezar a mover datos entre disco hacia memoria principal. Esto nos va a causar que el uso de entrada y salida en la computadora sea muy alto. Esto nos está dando un I/O Bound. Es muy común que cuando yo tengo un común I/O bound, yo tenga de compañía un uso de CPU alto, porque a cada rato voy a tener que levantar el SO para que el DMA me traiga las páginas de discos. El use case es el tipo de datos que vamos a manejar, relacionado con la industria donde se está trabajando.

El throughput o el rendimiento que me ofrece un disco lo manejamos en IOPS, son operaciones de entrada y salida por segundo. Supongamos que tenemos una base de datos de AWS y hay que estimar su precio. Un servidor o cualquier tipo de servicio se llama EC2. Uno busca las instancias que hay. Hay instancias de propósito general, que normalmente tiene una cantidad de CPU y de memoria relativamente baja. Una instancia especializada para computar usa procesadores más poderosos. IOPS: operaciones de entrada y salida por segundo. Una instancia enfocada en memoria es porque tenemos un dataset muy grande y necesitamos que el dataset entre en la memoria principal de la computadora. Storage optimize: manejamos volúmenes de datos tan grandes que necesitamos que el rendimiento de los discos sean bastante altos.

IOPS son tan importantes porque si queremos instalar una DB, tenemos que escoger las máquinas que va a usar. On demand: yo pago mensualmente por el uso de la máquina. Amazon Elastic Block Storage: equivalente al disco. Un disco de 1 TB por lo general nos va a dar 3000 IOPs. Esto significa que podemos hacer 3000 escrituras, lecturas, o ambas, por segundo. Si esto no es suficiente para la DB (dura mucho para responder), tenemos que aumentar la cantidad de IOPS que vamos a tener. Al cambiar los IOPS, aumenta el rendimiento que va a tener el computador.

Necesitamos tener muy claro el concepto de dónde en una nube se queda el rendimiento de la base de datos.

El CPU tiene cores. Cuando un CPU tiene 4 cores, significa que el CPU internamente puede manejar la ejecución en paralelo, al mismo tiempo, de 4 procesos al mismo tiempo. Esto quiere decir en una base de datos que si tenemos un CPU con solo 1 core y teníamos 4 transacciones, lo que me iba a generar que el CPU solito se lo iban a tener que repartir entre 4 transacciones, el sistema operativo, más todos los programas de usuario. Siempre que había una interrupción, siempre tenía un solo core para que me atendiera eso.

En un CPU con varios cores, tiene varios juegos de registros, entonces cuando despiertan al sistema operativo, solo hago context switch en solo 1. Puedo en paralelo a máxima capacidad, que 4 transacciones corran al mismo tiempo. Esto mejora el rendimiento de la base de datos.

El thread de CPU es un scheduling a nivel de hardware que se hace a nivel de core. El CPU está preparado para repartir el trabajo dentro del CPU. Esto implica que no hay intervención del SO. Los cambios de contexto son bastante rápidos. Si tengo un procesador con 4 cores y tiene hyper threading, voy a tener la noción a nivel de hardware que tengo 16 CPUs independientes para hacer procesamiento. Estos CPU's son de uso comercial y son muy caros. La línea de procesadores de Intel para esto son Xeon. Las computadoras más caras en cloud providers los utilizan.

Data locality: Si tenemos un dataset, el conjunto de datos que tenemos para trabajar, y hay un dato. Si uno hace una búsqueda de algo, usualmente uno se trae también los datos alrededor porque hay una alta probabilidad de que se utilicen también. Así me ahorro ciclos de reloj si el usuario pide data relacionada o cercana. El comportamiento de usuario es que cuando pide cierta información, el usuario va a solicitar información relacionada o local. En una sola transacción a disco, voy a irme y traerme la información.

Cuando una computadora dejó de ser suficiente para mantener una base de datos, nació la necesidad de escalar. Esto tiene dos facetas. La primera es scale up o escalamiento vertical. Cuando una computadora se queda muy pequeña para manejar la base de datos, como que el CPU o memoria dejó de ser suficiente, lo único que puedo hacer es crecer esa computadora, como agregar más disco, memoria, CPU's, etc. Siempre me voy a enfrentar a la dificultad de que la tarjeta madre o el SO no aguanten meter más memoria o disco. En algunos casos tengo que botar el hardware anterior y comprar uno nuevo. Esto me mete en problemas es que el sistema operativo tal vez no está preparado para manejar hardwares con mucha memoria, CPU y disco, tienen un límite en qué pueden mover. Otro aspecto es el precio, entre más grande la computadora más alto es su precio. En la mayoría de los casos tengo que descartar hardware, también. Estoy desperdiciando hardware.

Hay otro aspecto de protección de datos. Cuando guardamos datos de ciudadanos de la unión europea, uno no puede vender el hardware, uno tiene que destruir el hardware. Un disco duro que haya contenido información de ciudadanos europeos yo no lo puedo vender, tengo que llevarlo a una entidad certificada y destruirlo.

Del otro lado del Scale Up, tenemos el Scale Out o escalamiento horizontal. Cuando la máquina que tenemos no es suficiente, compramos otra nueva, y seguimos comprando máquinas iguales. Uno de los problemas es la red y la otra es la coordinación.

El primer concepto de sistemas distribuidos importante es master/slave o primary stand/by. Master / Slave: la forma en la cual hemos llamado a la mayoría de sistemas distribuidos en bases de datos, pero la industria está sacando el uso de estos conceptos y están siendo reemplazados por primary/stand by Primary/ Stand By: Reemplazo de Master / Slave. Esto quiere decir que tenemos una base de datos. Todos los motores tienen este enfoque. Vamos a tener una base de datos que va a ser un espejo. Voy a tener una base de datos que siempre va a estar como primary o master y otra que va a estar en stand by. Supongamos que cuando un usuario viene y habla con la primary, como hacer un update, mediante la red el update va a fluir de una base de datos a la otra. Si un usuario llega y hace un select, el select va a ser servido de la base de datos que está marcada como primary. Si por alguna razón la primary se cae, el sistema lo que hace es decirle a la stand by que es la nueva base de datos primary y la que se cayó se marca como inaccesible. Cuando la inaccesible vuelve a estar disponible, se marca como stand by, y la master le pasa por red todos los updates para que

esté preparada a que la nueva primary se muera. El esquema siempre necesita alguien que haga el desempate, que se llama witness. Es el proceso que está viendo a ver cuál de las bases de datos se cayó. En un sistema distribuido tenemos que identificar el Single Point of Failure, especialmente cuando estamos manejando datos. Esto quiere decir cuál es el eslabón más débil de mi aplicación. Cuando tenemos una base de datos que corre en un solo servidor, si se cae se murió. Este es el single point of failure. Si solo hay un witness y se cae, hay un single point of failure porque si se cae el witness, quién es el árbitro. Cuando hacemos arquitectura, tenemos que ir identificando los single points of failure y meterles redundancia.

La red nos va a meter un problema, el bandwidth. Esta es la capacidad que tiene la red para mover datos. Si tengo un usuario que está haciendo muchos writes o inserts a la base de dato, si la red no tiene la suficiente capacidad para mover todas las escrituras a tiempo, la otra base de datos no va a estar lista para tomar el trabajo en caso de que la primary se muera. Si estamos en una base de datos donde la consistencia es importante, como una relacional, y la stand by no tiene todos los datos actualizados, no puede tomar el control. La red me mete ese nivel de complejidad. En scale out hay problemas de red y de coordinación.

Replicación: tenemos una base de datos, la cual replica a otra base de datos. Todas las escrituras que entran a una, se pasan a la siguiente base de datos. A nivel de protección de datos tenemos varias bases de datos. Para perder la aplicación en línea, se tienen que caer todos los servidores y esto tiene una probabilidad muy baja. El principal problema es el dinero porque es muy caro, pero la replicación la necesito para lo que normalmente manejamos en todo sistema o servicio que prestamos, que es el SLA, el Service Level Agreement. Nosotros podemos instalar la base de datos, pero si instalamos la base de datos, es nuestra responsabilidad. Esto implica que tenemos que manejar tanto la seguridad física, la de red, la replicación, la ubicación geográfica, los updates, asegurarnos que el hardware esté bien, etc. Normalmente, encargarse de todo esto se convierte en un gran problema.

Hace 20 años, ¿qué pasaba? Cuando un banco tenía un problema y necesitaba una base de datos, el sistema de tecnologías de información tenía que salir a la calle y buscar lo que comúnmente se conoce como un DBA, un database administrator. Se contrataba a una persona especializada en bases de datos que se encargara de instalar, mantener la base de datos con todas sus características. Un DBA es una persona muy especializada. Si el trabajo de los bancos no es administrar bases de datos, el banco no necesita contratar un DBA; pero antes se necesitaba porque no había otra opción. Actualmente es que uno contrata bases de datos de alguna empresa especializada. Lo que dan es un SLA que dice que uno asegura que la base de datos va a tener una disponibilidad mensual mayor a 99.5% y si no tengo esa disponibilidad mensual, devuelvo un 10% de lo que me pagó. Si está entre un 99 y un 95% devuelvo 25% y si está menor, devuelvo un 100%

A uno siempre le ofrecen un SLA, el cual está relacionado con replicación. Entre más replicación yo tenga, más probabilidad yo tengo de cumplir un SLA. Tenemos geolocalización.

La geolocalización es bastante importante en términos de replicación y velocidad en la cual yo puedo servir a mis usuarios. En las torres gemelas había una empresa que tenía una base de datos que la replicaba en la otra torre. Siempre pensaron que las bases de datos iban a estar seguras porque era imposible que en un mismo evento se perdieran los dos edificios y los dos se perdieron.

Normalmente, tenemos que tomar en cuenta dentro de los SLAs, el factor de desastre natural o desastre de fuerza mayor. Normalmente, lo que se hace es que quienes nos venden bases de datos y cuando instalamos las bases de datos manualmente, como en un cloud provider, seleccionamos una región. Quiere decir una ubicación geográfica. En Estados Unidos hay un montón de regiones. Las regiones normalmente tienen el concepto de AZ, una región de disponibilidad, availability zones. Están diseñados de forma que sean impares.

Cada availability zone está hecha para que estén separadas unas de las otras por al menos 100 kilómetros. Si tenemos que diseñar una base de datos que esté altamente disponible y que tenga un SLA, uno selecciona una región en Amazon y pone una réplica en cada región. La probabilidad de que un desastre afecte a tres regiones es sumamente baja. Uno puede observar que se caiga una availability zone de viaje. Normalmente son regionales, pero existen servicios multi-región. La replicación se da no solo dentro de una misma availability zone sino que también los datos se replican hacia otra región. Esto me da más seguridad para la base de datos.

Con la geolocalización se da algo interesante. En un servicio que utiliza base de datos que sirve clientes, y los clientes están en USA y Argentina. Hacemos la base de datos en Estados Unidos y la replicamos en varias AZ y regiones. No obstante, los usuarios todavía deben de viajar desde Argentina hasta USA para traer los datos, que se hace por la red. Siempre que viajo por la red va a haber un delay, un retraso en el request y que se devuelvan los datos. Cuando tengo muchos usuarios, la base de datos mantiene más tiempo abiertos los puertos y threads, que causa más probabilidad de agotar file descriptors y memoria, etc. Podemos hacer replicación de datos en ubicaciones geográficas más grandes o cercanas de dónde se encuentran los usuarios. El delay nunca nos lo vamos a quitar. Los paquetes tienen que viajar por cables desde mi ubicación geográfica a otra. Entre más yo reduzca la distancia, lo más probable es que tenga un mejor rendimiento en mi base de datos.

Fail over: antiguamente, a nivel de base de datos conocíamos el concepto de switch over. Nuestros usuarios estaban consumiendo de esa base de datos y de casualidad la base de datos se caía. Una persona que era la encargada de mantener el sistema se daba cuenta de que la base de datos principal se cayó y hacía un switch over de la aplicación para que leyera de la nueva base de datos, promovía la nueva base de datos. Switch over implica un proceso manual que iba a tener lo que nosotros conocemos en aplicaciones como un downtime. Esto implica que desde el momento que la persona se daba cuenta que la base de datos estaba abajo y hasta que cambiaba la base de datos, íbamos a tener un tiempo en el cual el sistema estaba abajo. Pierdo usuarios y dinero.

Después, apareció el fail over. Básicamente, automáticamente hace un cambio entre las bases de datos. Va a haber un componente en la red que va a detectar que existe una falla en la base de datos principal y automáticamente me hace el movimiento de usuarios de una base de datos a la otra. Inicialmente esto lo hacía el witness, pero ahora utilizamos un concepto de load balancer. Este load balancer desacopla los usuarios de la base de datos. El usuario no sabe cuál es la base de datos que lo está sirviendo. El load balancer automáticamente responde.

A nivel de base de datos, el load balancer automáticamente detecta cuando una base de datos se cae o cuando está arriba, detecta quién es el primario y quién está en stand by y sabe coordinar y redireccionar los requests. En sistemas más profesionales, si agrego nuevas réplicas, el load balancer se da cuenta que hay nuevas réplicas y que están sincronizadas. Ahora todo se maneja con fail over.

High Availabiltity viene a decir que el tiempo para recuperarse de una falla es sumamente alto. Para que se recupere rápidamente y maximizar el SLA, necesitamos replicación, remover los Single Point of Failure, geolocalización, fail over y con esas cosas, voy a lograr que la base de datos se maximice en el tiempo que va a estar arriba.

Todo cliente y empresa va a decir que la base de datos esté arriba 100% del tiempo, pero esto es prácticamente imposible. Yo puedo diseñar una base de datos un 99.5% del tiempo, pero es muy cara. Se hace un análisis costo-beneficio. Deciden qué tanto tiempo soporta la empresa tener un sistema abajo. Lo

que hacemos es hacer el diseño de alta disponibilidad enfocado en tener un SLA que no sea mayor al objetivo. Diseñamos tratando de diseñar el objetivo que la base de datos no esté abajo más de 12 horas en un mes, por ejemplo.

Coordinación: Normalmente está relacionado con lo de un shared resource. Si tengo 3 servidores de DB peleando por el shared resource, estos servidores de base de datos tienen que coordinarse de alguna forma para obtener el shared resource. A través de mensajes entre servidores por la red, obtengo la coordinación necesaria para obtener el recurso compartido. Mientras esto ocurre, tengo programas esperando el uso del CPU, tengo uso de memoria y file descriptors.

La base de datos en un sistema distribuido me mete el problema de la red e intercambio de mensajes. Mis procesos se van a quedar un poco más de tiempo en memoria. Aumenta la probabilidad de que la DB se quede sin recursos.

Quality of Service: la capacidad que tiene una base de datos para dar servicio a los usuarios. Esto se refiere a que cuando se diseña la base de datos, tomando en cuenta todos los aspectos, la base de datos aguanta un máximo de 100 usuarios. Como la base de datos aguanta un máximo de 100 usuarios. El concepto de QoS es que el sistema debe estar preparado para atender a 100 usuarios y a rechazar a los usuarios adicionales cuando el rendimiento del sistema se va a ver afectado. A los 100 usuarios que les podía dar buen servicio, les voy a dar mal servicio si atiendo a más de 100 usuarios.

Auto-Scaling: Scaling policy. Esto quiere decir que definimos una base de datos que tiene 4 gigas de ram y tres CPUs. El cloud provider empieza a observar la base de datos y ve que la base de datos está usando las 4 gigas de ram completos y utilizando al 100% los tres cpus. En el scaling policy uno dice que si el uso de CPU es mayor un 95% y el uso de la memoria es mayor a un 95% por al menos 15 minutos, escale por un monto que uno definió, como que meta un procesador y 2 gigas de RAM más. Dependiendo del uso de la base de datos, el cloud provider le da más recursos y conforme baje el uso, empieza a hacerla más pequeña también. Uno paga por lo que uno usa. Un ejemplo de esto está DynamoDB de Amazon; RDS con Aurora; CosmosDB con las bases de datos Cassandra, MySQL, PostgreSQL, SQL Server y Gobling; Cloud Spanner de GCP.

Data partition: shard/partition. Este concepto se usa en bases relacionales y no relacionales cuando son bases de datos en un sistema distribuido. Supongamos que tenemos un dataset que se puede ver como una tabla que tiene 10000 entradas o rows. Si tenemos dos bases de datos, podemos replicar esas dos bases de datos, lo cual está bien. A nivel de base de datos de SQL que usa el término de particiones en una tabla de 10000 rows hay que hacer 10000 comparaciones. ¿Cuál es la forma más efectiva o el algoritmo para resolverlo? Divide y vencerás. El divide y vencerás agarra los registros de la tabla y los particiona en partes de 2000 records, por ejemplo. La partición de datos que voy a utilizar se escoge de acuerdo con la columna. De acuerdo con esa columna que escogemos para particionar, dividimos los datos, de forma tal que el acceso sea rápido. Escogemos una columna que me permita hacer la partición lo más equitativamente posible. La base de datos en lugar de hacer una búsqueda sobre toda la tabla, identifica la partición que tiene el dato y lo devuelve. Reduce la cantidad de comparaciones que tengo que hacer. Con una base de datos que corre en un sistema distribuido, yo puedo agarrar y decir que la partición 1 va a estar guardado en el servidor 1 y en el 2. La partición 2 va estar guardado en el servidor 2 y en el 3. Si La columna con la que hicimos la partición tiene otra columna relacionada, podemos hacer un query sencillo que se resuelve paralelamente en múltiples servidores. Las particiones pueden ser independientes y esto causa un mejor rendimiento.

A nivel de base de datos no relacional tenemos el concepto de shard. Agarramos el dataset y lo partimos en pedazos que vamos a ir distribuyendo en los servidores. Tienen una característica importante. Al shard le

puedo asignar replicación. En una base de datos relacional, la partición es un pedazo de información. El shard en una base de datos NoSQL, le puedo configurar la replicación a nivel de shard y la geolocalización. En una no relacional, en lugar de replicar toda la base de datos, podemos dividir la información de acuerdo con geolocalización. Así, tenemos la información de Argentina cerca y la de Estados Unidos cerca de USA.

Si a un shard le pongo tres réplicas y el desarrollador pone todos los shards en el mismo servidor y región, los shards solos se dan cuenta que están en la misma availability zone, entonces se transfieren a zonas adecuadas para maximizar el high availability.

Storage: los discos son mecánicos y son un poco lentos. Tienen platos que rotan que tienen múltiples cabezas de escritura y lectura. El brazo se coloca en el plato adecuado y esperar a que la rotación llegue al lugar donde están los datos y los pueda leer.

Los discos de estado sólido son chips de memoria que son más lentos que la memoria principal que una computadora.

Mucha gente piensa que los discos duros ya no se utilizan. La verdad es que se utilizan. Existen use cases para los cuales estos discos son especiales y rebasan en rendimiento a SSDs. Por ejemplo, los logs son un tipo de información que siguen una característica que se llama time series, lo que los hace especiales para db de series de tiempo. Siempre van a insertar datos y los datos siempre se van a insertar al final. No es común que uno vaya y modifique el dato, solo pegamos al final. En un disco mecánico, como solo hacemos escrituras al final, yo no tengo que estar moviendo la cabeza lectora para buscar la posición donde va a leer. Por lo tanto, un use case de logs son especialmente buenos para discos mecánicos. Todo lo que sea escritura secuencial, se ve beneficiado por discos mecánicos. La diferencia en precio en un HDD con un SSD es abismal. Si desarrollamos una base de datos que tiene una característica como esta, es importante que tengamos en cuenta con hardware distinto, igualando el rendimiento y reduciendo el costo de la base de datos.

RAIDS: arreglos de discos. Normalmente tenemos múltiples discos. Al sistema operativo se le presenta una abstracción que hace que el OS piense con un solo disco gigantesco, pero en realidad el hardware de RAID está escribiendo en varios discos al mismo tiempo. La forma en la cual se organizan los datos corresponde a niveles de raid.

Raid 0: agarrar un archivo y lo particiona en bloques. Cuando el sistema operativo envía al bloque hacia el raid level, el raid asigna el bloque 1 al disco 1, el 2 al disco 2, el 3 al disco 3, el 4 al disco 1, y así sucesivamente reparte los bloques en los discos disponibles. El raid va a mejorar el rendimiento por el hecho de que yo puedo escribir en paralelo y leer en paralelo. Si yo tengo un disco solito conectado al OS, yo escribo bloque por bloque, yo puedo escribir en paralelo, y la cantida de bloques que puedo escribir está delimitada por la cantidad de discos que tenga el raid. Normalmente una base de datos con mucha escritura y lectura, este nivel es muy bueno.

Raid 1: Es conocido como un RAID espejo. Si yo tengo 5 discos, los 5 discos van a guardar una copia de la información. La información se parte en bloques, pero los bloques se van a guardar igual en todos los discos. Esto nos garantiza replicación y redundancia. La única forma de perder una base de datos es perder todos los discos. En un raid 0, con que solo se pierda 1 disco, pierdo la base de datos, ya que no puedo recuperar los bloques de información.

RAID 2: No se utiliza mucho en la práctica. Guarda los datos en varios discos, pero aparte de que guarda los datos, va a generar bits de paridad. Una paridad es básicamente la operación XOR. Si yo tengo un bloque o un stripe en un disco 0100, en otro 0101, y en otro disco 0010, hacemos una operación de xor a nivel de bit

en la posición. Hacemos el cálculo de los bits de paridad. El disco de paridad va a mantener la paridad de los bloques de datos que tengo almacenados. Una vez que tengo la paridad, en el raid dos, voy a tener al menos 3 discos de paridad a nivel de bits. Si pierdo algún disco completamente, yo puedo hacer un xor entre los discos restantes, junto con el de la paridad y voy a recuperar el campo del disco perdido. Yo puedo ir recuperando el disco a partir de volver a realizar los xor. El raid level 2 va guardando el bloque de datos y genera bits de paridad a nivel de bit. Para perder los datos completamente, necesito perder la cantidad de discos de paridad más 1 No se utiliza porque las operaciones son a nivel de bits. Se tienen varios discos de paridad porque si tenemos datos en el disco 0, 1, 2 y 3 y calculamos la paridad en el disco 4, la máxima cantidad de discos que se raid permite perder para perder toda la información es perder el disco de paridad+1. Si yo duplico la paridad implica que tengo que perder dos discos de paridad + 1. Estoy disminuyendo la probabilidad de que tenga una falla. Es simplemente que yo meto redundancia. Entre más redundancia, menor probabilidad de pérdida.

El raid level 3 es igual al 2 con que las operaciones de paridad funcionan a nivel de byte

El raid level 4: igual al 2 y 3, pero funciona a nivel de bloques. Nos deja los bits de paridad todos en un disco.

El raid 5 reparte los bloques de paridad en diferentes discos. Va escalonado. La paridad está distribuida y no la voy a perder en un solo disco.

El raid 6 mete más discos de paridad.

Raid 10: se comporta como un raid 0, y debajo en lugar de tener discos tiene dos raid levels 1.

Raid 0+1. Se comporta como raid level 1 y por debajo se comporta como un raid level 0. Esto me causa que yo tenga redundancia y que yo pueda hacer lecturas en paralelo gracias al raid level 0.

NAS: discos que se ofrecen en red que viajan a través de una red.

SAN: discos que se ofrecen a través de una red de fibra óptica que están conectados directamente a la computadora.