Elmélet

Relációs adatbázisok:

A relációs adatbázisok megbízhatók, ismertek, bizonyítottak az évek során, általános célokra felhasználhatók, de vannak korlátaik: merev schemával rendelkeznek, drágák, egy gépen való futtatásra tervezettek, felmerülnek lockolási problémák. A felmerülő problémákra 2006-2007 ben A Google bemutatta a Big Table-t, az Amazon pedig a DynamoDB-t mint megoldást. A Cassandra örökli mindkettőnek a tulajdonságait. A NOSQL nem az SQL hiányát jelenti. A NOSQL adatbázisok rugalmas schemával rendelkeznek, vertikálisan és horizontálisan is skálázhatóak. Vertikális skálázás (scale) esetén az adatbázis átkerül egy erősebb gépre. Horizontális skálázás(scale out) esetén az adatbázis elosztott rendszerként fog működni.

NOSQL adatbázisok:

- Key-Value Store, ilyen a Redis és a Riak
- Document Store, ilyen a CouchDB és a MongoDB
- Column Store, ilyen a Cassandra és az Apache Hbase

A Column store típusú adatbázis esetén az oszlopok neve soronként változhat, széles sorok. A Cassandra rendelkezik SQL interfésszel, ami a belső adatmodell tetejére lett építve, gyorasaság érdekében optimalizálva lett, ez az elsődleges kezelési felület. CQL esetén nincs olvasás írás előtt, az INSERT és az UPDATE ugyanúgy működik, csak a megadott oszlopokat írjuk, nem az egész sort, a Cassandra nem ellenőrzi, hogy előfordul-e kulcsütközés. Elsődleges adattípusok: int, tinyint, smallint, bigint, decimal, float, double, text (varchar), ascii, timestamp, date, time, uuid, timeuuid, blob, inet. Gyűjtemények: list, set, map<TKey, Tvalue>. Cassandrában a felhasználó is létrehozhat új adattípusokat. Az architectúra lényege a skálázhatóság, aminek része a particionálásból, replikálásból, konszisztencia, tartósság, teljesítmény. A particionálás alatt általában horizontális particionálást értünk, mint például a régio alapú particionálás. A particionálás hashelés segítségével történik. A hash algortmus gyors és konszisztens.

Particionálás konfigurálható:

- Murmur3Partitioner, ez az alapértelmezett, egyenletes eloszlású
- ByteOrderedPartitoner, ma már nem ajánlott
- RandomPartitioner, régen ez volt az alapértelmezett

Az adatok node-okra kerülnek. A replikáció segítségével elkerülhetjük a hibák miatti leállást, növelhetjük vele a gyorsaságot, egyszerre sokkal több requestre tudunk válaszolni. Nincsen master node. Nem master-slave architechtúra. A replikációs faktor megmutatja, hogy hány node tárolja el ugyanazt az adatot, éles környezetben ez általában három.

Replikációs stratégiák:

- SimpleStrategy
- NetworkTopologyStrategy (Rack & DC aware)

Több datacentre esetén a datacenterek rendelkezhetnek különböző darab nodeokkal, és különböző replikációs faktorral. Bármely node olvashat vagy írhat. Attól, hogy egy datacentre leáll, még az adatok

elérhetők maradnak. A replikáció általában lassú, minden node egyenrangú, az olvasás gyakran inkonszisztens. NOSql esetén a magas rendelkezésreállás, konszisztencia és partició tolerancia közül csak kettőt választhatunk. A Cassandra, Riak, Dynamo esetében a magas rendelkezésre állás a cél. A MongoDB, Hbase, Redis esetén a konszisztencia a cél.

Választáskor fontos megállapítani, hogy mennyire fontos számunkra a konszisztencia. Elsőre úgy gondolhatjuk, social media esetén nem fontos, webáruház esetén közepesen, bank esetén pedig nagyon. Valójában a legtöbb esetben meg tudunk lenni konzisztencia nélkül, webáruház esetén, küldünk egy emailt a vevőnek, hogy nem holnap, hanem három nap múlva érkezik a termék, bank esetén pedig meg lehet engedni, hogy az ügyfél minuszba menjen, erre úgyis kamatot fog fizetni.

Általában a konszisztencia szintje állítható.

Gosip Protocol - a node-ok folyamatosan kommunikálnak, elég, ha egy pár node-ot ismerünk, így is minden információt megkapunk.

Cassandraban a Partition key kötelező, a Clustering key opcionális, a Primary key a Partition key és a Clustering key-ből áll. Összetett kulcs esetén: PRIMARY KEY ((pk1, pk2), ck1, ck2). Az elsődleges kulcsnak egyedinek (unique) kell lennie.

A Cassandra nem tud bármilyen szintaktikailag és szemantikailag helyes lekérdezést végrehajtani. Csak olyan lekérdezések hajthatók végre, amelyek gyorsan elvégezhetők. Az adatbázis tervezésekor kell tudni milyen lekérdezéseket fogunk a későbbiekben végrehajtani rajta.

Amikor partition key alapján szűrünk, az értéket egyértelműen meg kell adni, az IN kulcsszó elfogadott, az egyenlőtlenségi operátorok nem.

A rekordok egy partición belül clustering key alapján (alapértelmezetten növekvő) sorrendbe rendezve vannak tárolva.

A törölt adatokat "sírkövekre" cseréljük, amelyek egy megadott ideig (alapértelmezetten 10 nap) vannak az adatbázisban. Túl sok "sírkő" csökkentheti a teljesítményt.

A CQL ben nincsenek joinok, mert ezek lassúak és költségesek, limitált a where (cask kulcs és index alapján szűrhetünk), limitált az ORDER BY és a GROUP BY.

A Cassamdra annak ellenére, hogy SQL interfésszel rendelkezik, nem egy relációs adatbázis, elosztott rendszereken működik, skálázható, gyors, hatékony, az SQL interfész a kényelmet szolgálja.

A Batch-ek atomiak, vagyis minden műveletet revertelünk hiba esetén, nem izoláltak, vagyis más műveletek láthatják a még nem alkalmazott változtatásokat, a műveletek végrehajtási sorrendje nem determinisztikus.

Itt is vannak materializát nézetek, de limitáltak, a materializált nézet elsődleges kulcsa kell tartalmazza a forrás elsődleges kulcsának összes oszlopát, csak egy új oszlop adható hozzá a meterializált nézet elsődleges kulcsához, statikus oszlopok nem megengedettek, a kulcs oszlopok nem tartalmazhatnak null értékeket, csak a 3.x verziótól elérhetők, törölhet adatokat, ha az elsődleges kulcs mezői változnak.

Támogat JSON formátumot, lehetővé teszi a könnyű JSON feldolgozást a kliensen.

Adatmodellezési szabályok:

- lekérdezés alapú tervezés - Már tervezés előtt előre kell tudni milyen lekérdezéseink lesznek
- de-normalizált táblák - Egy az egyhez kapcsolatok létrehozása beágyazott mezőként, egy az n-hez kapcsolat megoldása gyűjteményekkel vagy clustering kulcsokkal -másodlagos indexelés - Gyorsítja a nem kulcs oszlop alapú kereséseket, lassabb, mint a lookups by key, jelentős gyorsaságot eredményez, ha már az adat szűrve van partition key alapján
- duplikált táblák - Több táblát létrehozunk ugyanazzal az adattal, de más partition key-el, update esetén mindkettőt módosítjuk, bizonyos esetben index táblákat és materializált nézeteket is használunk.

Adatbázis tervezésének a lépései:

- Conceptuális modell létrehozása
- Táblák megtervezése és létrehozása
- Az adatmodell és a táblák ellenőrzése

A Cassandra jól használható ha:

- folyamatosan nagy mennyiségű adatot kell gyorsan adatbázisban eltárolni
- szükségünk van skálázhatóságra
- több datacenterre van szükségünk
- idősor adatok esetén (Time Series Data)

A Cassandra egy általános célú NoSQL adatbázis, amelyet különböző alkalmazásokhoz használnak az összes iparágban. Számos esetben a legtöbb más lehetőségnél jobb teljesítményt nyújt. Ezek közé tartoznak:

- Internet of Things (IOT) alkalmazások - A Cassandra tökéletesen alkalmas nagy mennyiségű, gyorsan érkező adatok fogadására és elemzésére olyan eszközökből, érzékelőkből és hasonló mechanizmusokból, amelyek sok különböző helyen találhatók.
- Termékkatalógusok és kiskereskedelmi alkalmazások - Azoknak a kiskereskedőknek, akiknek tartós bevásárlókosár-védelmükre, gyors termékkatalógus-bevitelre és keresésre, valamint ehhez hasonló kiskereskedelmi alkalmazástámogatásra van szükségük, a Cassandra az adatbázisuk választása.
- Felhasználói tevékenység nyomon követése és ellenőrzése - A média-, játék- és szórakoztató vállalatok a Cassandra-t használják a felhasználók interakcióinak nyomon követésére és figyelemmel kísérésére a filmjeikkel, zenéjükkel, játékaikkal, weboldalukkal és online alkalmazásaikkal.
- Üzenetküldő alkalmazások - A Cassandra az adatbázis-gerinc szolgáltatásául szolgál számos mobiltelefon-, távközlési, kábel/nélküli és üzenetküldő szolgáltató alkalmazásában.
- Társadalmi médiaelemzések és ajánló rendszerek - Az online vállalatok, weboldalak és közösségi média szolgáltatók a Cassandra-t használják az ügyfeleik beolvasására, elemzésére és elemzési és ajánlási lehetőségek nyújtására.
- Egyéb idősoros alapú alkalmazások - A Cassandra gyors írási képessége, széles sormegtervezése és az a képessége, hogy csak azokat az oszlopokat olvassa el, amelyek szükségesek bizonyos lekérdezések kielégítéséhez, jól alkalmazható bármilyen idősoros alapú alkalmazáshoz.

A Cassandra nem ideális választás, ha:

- Nem szükséges skálázhatóság □ilyen esetekben használjon relációs adatbázist
- Túl sok vagy ismeretlenek a keresési feltételek
- A konszisztencia fontosabb, mint a rendelkezésre állás

- Nehéz adatokat oszlopos formátumba átmappelni □a JSON támogatás és a felhasználó által definiált típusok segíthetnek
- Olvasásra tervezett terhelések □ A Cassandra írásra van optimalizálva
- Gyakori, széles körű

Ismert driverek:

- DataStax
- Astyanax
- Kundera
- ODBC

A Cassandra jól használható műveleti központokban (Operation Center).

A Cassandra nem része a Hadoopnak, de a Hadoop ekoszisztéma sok komponenséhez jól integrálódik. A Cassandra integrálható a Spark-kal.

A DataStax Enterprise szolgáltatás zökkenőmentes integrációt biztosít a Cassandra és a Solr között.

A Spotify és a Netflix is Cassandra adatbázist használ, mivel nagyon gyors, magas rendelkezésre állású, horizontálisan skálázható, támogatja a cross-site replikációt, alacsony késleltetésel rendelkezik, nagymennyiségű adatot tud gyorsan betölteni Crunch és Storm-ról. A Cassandra tárolja a felhasználó profilok attribútumait, metadatákat lejátszási listákról és előadókról. A replikációnak köszönhetően a Netflix akkor is elérhető volt, amikor az egyik datacenterüket elöntötte az árvíz.