Spark Streaming and Cloud

Mind a NoSQL és a BigData rendszerek szétosztott számítást használnak hogy tárolják és feldolgozzák a nagy mennyiségű adatot. A NoSQL lényege hogy közel valós időben adjon válaszokat, míg a BigDatanál a lényeg hogy bármilyen és bármennyire adatra tudjunk lekérdezéseket végrehajtani.

A batch processing pedig azt jelenti hogy feldolgozzuk egy hosszú idő alatt összegyűjtött összes adatot. Ezt használják a Big Data nyelvek, nem valós időben futnak mivel várniuk kell a következő batchre azaz következő adagra. Streaming jelentése hogy feldolgozzuk az adatokat amint azok létre vannak hozva. A streamelés problémája, hogy nem elég tudni milyen adatot kapunk, de azt is tudnunk kell, hogy eddig milyen adataink vannak ami egyáltalán nem triviális feladat. Egy másik probléma hogy az esemény megtörténésének ideje nem feltétlenül ugyanolyan a sebessége mint a feldolgozásénak, szóval késve kaphatunk meg információt ami már a múltban történt. Pont ezen okok miatt megbízhatatlannak tartották sokáig.

A lambda architektúra megpróbálja balanszolni a batch processing és streaming előnyeit. Szóval batch feldolgozást használ hogy tárolja az adatokat, azaz átláthatóak legyenek és közben valós időben dolgozza fel az adatokat a streamből.

Egy másik architektúra a kappa, amely egy szoftware architerktúra ami a streamből dolgozza fel az adatokat a lényege hogy feldolgozza a stream adatait és ezt kombinálja a batch processinggel egy verem (stack) segítségével. Mindent steramként dolgoz fel, egy analízis motort alkalmaz, ezáltal a kódolás és karbantartás egyszerübb –f Keep it short and simple (KISS) – nem változtatható adattipusokat alkalmaz, szóval az adatok állandóak és attól függően hogy nekünk mire van szükségünk változtathatjuk az adatokat.

A kérdés hogy hogyan dolgozzuk fel az eseményeket (events), erre van segítségünkre az Apache Kafka amely egy szétosztott adat raktár amely valós idős feldolgozást képes biztosítani. A Kafka csatornákból áll amely különböző témákról üzengetnek amelyeket a felhasználok megkaphatnak ám minden csatorna és minden fogyasztó külön van kezelve. Meg lehet adni, hogy mennyit ideig legyenek az egyes üzenetek tárolva és később még fogyasztókat is hozzá lehet adni, nem gondd hogyha az egyik fogyasztó gyorsabban dolgozza fel az üzeneteket mint egy másik mivel mindegyik haladhat a saját tempójában. A Klafka klusztereit brókereknek nevezzük, és a témákat képes particionálni mint a NoSQL esetében. A fogyasztókkal is számolni kell, létre lehet hozni fogyasztó csoportokat amelyek szétosztják a fogyasztók között az üzeneteket.

A másik kérdés, hogy hogyan dolgozzuk fel a nagy mennyiségű adatot. Ebben seggít nekünk a Spark streaming APIja. A streamingnél a cél hogy átalakítsuk a bemenő adatokat – fájlok, Kafka, socket – kimenő adatokká : fájlok, Kafka, Foreach, konzol. Egy ilyen átalakítást triggerek indítanak be, amelyek lehetnek: Microbatch alapján (kisebb adatok), időintervallum alapján, vagy csak egyszer dolgozza fel ami pontosan az mint a batch processing. A kimenetnél vagy hozzáadunk a létrehozottakhoz, vagy felülírhatjuk azokat amelyek léteznek. A Sparknál az adatfolyamot úgy képzelhetjük el mint egy táblát amelnyek nincsen megadva hogy mekkora, azaz folyamatosan adhatunk hozzá sorokat, minden új adat egy új sort eredményez. Valójában egy végtelen nagy Dataframeként lehet elképzelni.A Spark képes olvasni és írni is az adatfolyamot. A legtöbb transzformáció elérhető a streamingnél: aggregációk, joinok.

Az események idejét általában megtaláljuk az adatok között, és időintervallumok (time window) alapján csoportosítjuk. Több fajtáját ismerjük:

* Tumble window: 5 percenként nézzök meg az adatokat, minden 5 percben új ablak indul
* Sliding window: 10 percenként tesszük, de a következő már elindul az 5.perc után szóval 2 ablak mindig egyszerre dolgozik. Például átlagolásnál használják.
* Session window: nem előre definiált hossz, hanem akkor kezdődik amikor jön adat és ha már nem jön az ablak megszűnik, majd ha újra elindul akkor egy új ablakot hoz létre. A későbbi eseményeket mennyire érdemes feldolgozni, azt a vízjellel döntik el, de ez nem technikai részlet inkább üzleti megközelítést igényel.

Cloud computing

Számítógépes szolgáltatások az interneten keresztül, mondhatni bérelhető szolgáltatások. Típusai:

* Infrastukrúra mint szolgáltatás: szerverek, hálózatok, adatközpontok bérlése
* Szofware mint szolgáltatás: szoftwarek bérelése
* Szervertelen: Olyan applikációkat tervezhetünk amelyeknél nem szükséges kigondolni az infrastruktúrát

Az előnyei hogy a cloud hatalmas szóval könnyen bővíthető, akaráhol elérhető a világon, kifejezetten megbízhatóak és robusztus védelemmel bír. Amikor felhőre gondolunk legelőször a Hadoopra gondolunk, mivel léptezik Hadoop mint szolgáltatás. Előnyös mivel nem kell megfizetni a fenntartását, gyorsan létrehozható több kluszter, annyit fizetünk amennyit használjuk, és automatikusan növelhető ha szükséges, de csak azért fizetünk amennyit használunk. Hozzá kell tenni, hogy limitált a konfigurációval kell dolgozni, bizonyos országonál regulációk érvényesek és persze számolni kell az árral is hosszútávú felhasználásnál.

A felhőben raktározhatunk adatokat, amely pontosan úgy néz ki mint egy fájlrendszer amelyet a Hadoopnál láttunk, azaz szétosztott és replikált, meglepően olcsó és 99.999999999%-ban megbízható. Pontosan ezért szokták a HDFS helyett ezt a felhő raktárat használni, szóval így az adat nem számítógéphez kötött és több kluszter tud ugyanazzal az adattal dolgozni. Próbálunk elharárolódni az adatlokalitásttól, hála annak hogy egyre gyorsabbak a hálózatok és bár a HDFS még mindig gyorsabb de a cloud előnyei letagadhtatlanok és megérik a lassulást. Már úgy tekinthetünk rá mint kóda ami átveszi az infrastruktúra szerepét: létrehozza a klusztereket, felinstallál szofwareket, és ha végzett az adott kluszter akkor mentés után terminálja azt. Pénzt lehet spórolni úgy is, hogy spot instanceokat használunk ami azt jelenti, hogy akkor futtatjuk le a lekérdezéseket amikor a szolgáltatások alig vannak kihasználva, vagy esetleg ha nem sürgős őket befejezni.

A szervertelen architektúrában a felhő szolgáltatója osztja széz dinamikusan az erőforrásait. De a szerverek maguk nem fontosak számunkra, azt a szolgáltató dönti el, hogy mit kinek és hogyan oszt el. A tradicionális esetben az erősforrásokért és a szolgáltatásért fizetünk, a szervertelennél csak a használatért – abszolút mindegy miben volt írva, csak az számít hogy meddig futott.

Ha karrierekről van szó két ág meghatározó:

* Data Scientist : tissztítja és rendezi az adatokat, analizálja őket és belátást nyer az adatokba, és modelleket épít belőle. Fontos számára a statisztika és matek, gépi tanulás és Big Data tudás.
* Data Engineer: fejleszt, tesztel és karbantartja az adat pipelineokat, infrastruktúrát épít, Javában, Scálában fejleszt, ért a felhőhöz és a Big Datához