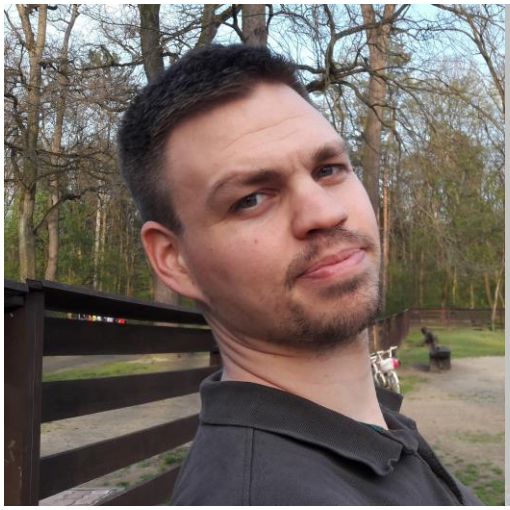



Dr. Hegedűs Péter, Dr. Ferenc Rudolf

Nagyméretű adatbázisok

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával.

Projekt azonosító: EFOP-3.4.3-16-2016-00014

Szegedi Tudományegyetem
Cím: 6720 Szeged, Dugonics tér 13.
www.u-szeged.hu
www.szechenyi2020.hu

Hadoop Distributed File System

Összefoglalás

Az olvasó ebből az olvasóleckéből képet kap az Apache Hadoop tárolásáért felelős Hadoop Distributed File System (HDSF) felépítéséről, központi elemeiről, valamint tervezési és működési elvéről. A leckék tárgyalják a HDFS tervezési elveit, adat replikálást, kommunikációs protokollt, illetve a megvalósításának részleteit, mint például az egyes fájlok elhelyezésének módját, blokkokra osztását, valamint az elérésüket biztosító parancsokat. Áttekintést nyújtunk az Apache Sqoop és Apache Flume eszközökről, amelyek segítségével relációs adatbázisokból vagy tetszőleges adat folyamról (stream) menthetünk adatokat a HDFS-re, vagy olvashatunk onnan vissza.
A lecke fejezetei:

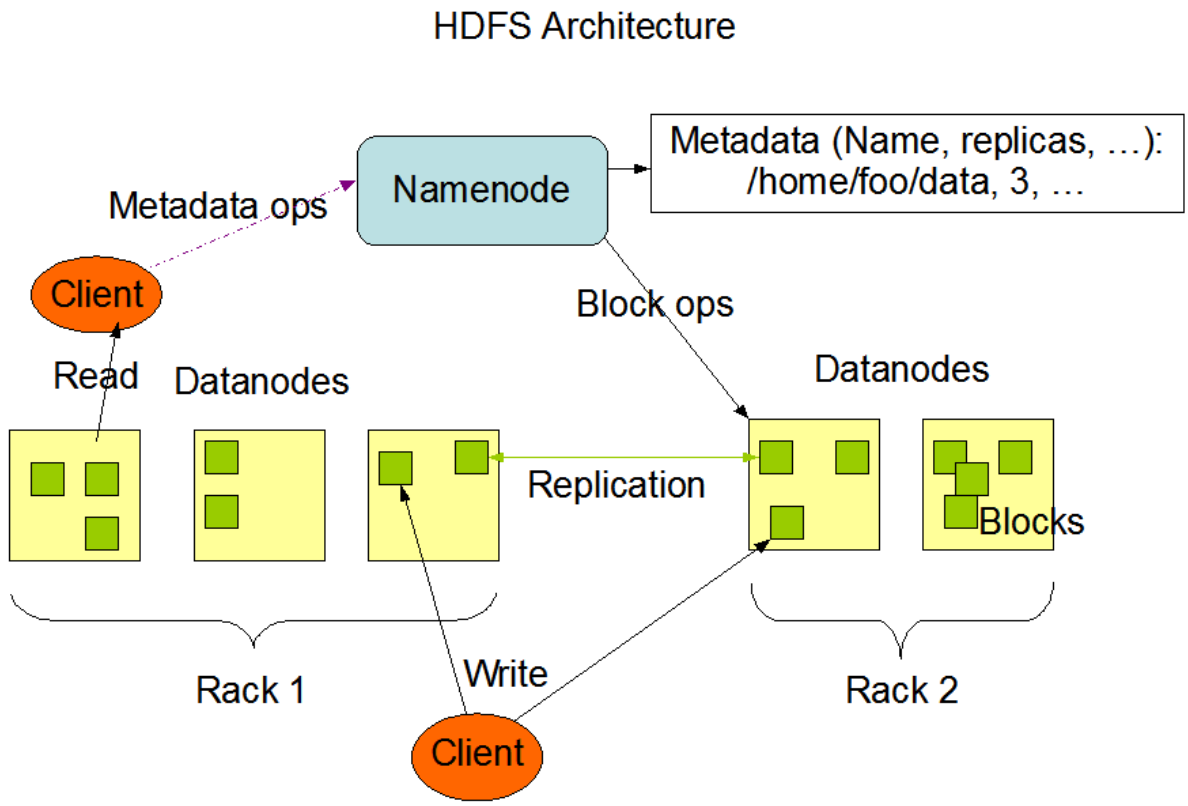
- 1. fejezet: **A Hadoop Distirbuted File System magasszintű bemutatása, felső szintű architektúrája (olvasó)**
- 2. fejezet: **A HDFS tervezése során követett irányelvek, valamint működésének alapvető jellemzői (olvasó)**
- 3. fejezet: **Nagymennyiségű és stream adatok közvetlen tárolása HDFS-en (olvasó)**

Téma típusa: **elméleti**
Olvasási idő: **55 perc**

📖 1. fejezet
Hadoop Distributed File System (HDFS)

Az Apache Hadoop keretrendszer skálázható és elosztott adattárolásáért felelős fájlrendszer komponens, amit hagyományos számítógépek klaszterén való működésre terveztek. Noha sokban hasonlít más elosztott fájlrendszerekhez, jelentős különbségek is vannak azokhoz képest. Magas hibatűrés jellemzi, és alacsony költségű hardware-en való futtatásra tervezték. Nagy sávszélességű adatelérést biztosít azon alkalmazások számára, akik hatalmas méretű adatokkal dolgoznak (a.k.a. BigData alkalmazások). Nem teljesen POSIX kompatibilis annak érdekében, hogy lehetővé tegye a rendszer adatokhoz való stream-elt hozzáférést. A HDFS fájlrendszert elsődlegesen nagy adatátviteli kapacitásra és batch alapú adatfeldolgozásra tervezték.
A rendszer ~~master/slave~~ primary/worker architektúrával rendelkezik a következő komponensekkel (lásd lenti ábra):

- `NameNode`: központi szerver, egyetlen primary csomópont, amely fogadja a kliens hozzáféréseket, és szabályozza a fájlokhoz való hozzáférést
- `DataNode`: tipikusan minden klaszter csomóponton fut egy DataNode, amely az alatta levő hardware segítségével tárolja az adatokat



> https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/images/hdfsarchitecture.png

A HDFS a legtöbb fájlrendszer által megszokott hierarchikus fájl szervezést használja: a fájlokat könyvtárakba rendezhetjük, és azokat tetszőleges mélységig egymásba ágyazhatjuk. A felhasználók vagy alkalmazások létrehozhatnak könyvtárakat és fájlokat tárolhatnak azokban. Létrehozhatók új fájlok, törölhetők azok, áthelyezhetők egyik könyvtárból a másikba vagy akár át is nevezhetők. A HDFS támogatja a felhasználói kvóták és hozzáférési engedélyeket (access permissions), de nem támogatja például a hard vagy soft linkelést. A `NameNode` felelőssége a fájlrendszer névterének karbantartása, azaz ő végzi az olyan fájlrendszer műveletek, mint a fájlok vagy könyvtárak megnyitása, lezárása vagy átnevezése, és ő tart nyilván minden meta-információt is a fájlokról. A HDFS-t alapból redundáns tárolásra tervezték, azaz egy fájlnak több másolatát is tárolja egy HDFS klaszter. A felhasználó vagy alkalmazások megadhatják, hogy egy-egy fájlnak hány másolatát hozza létre a HDFS, ez a replikációs faktor, amit szintén a `NameNode` tart nyilván.

A `DataNode`-ok végzik a technikai adattárolást, azaz a `NameNode` által nyilvántartott fájlrendszer struktúra elemeinek tartalma ott tárolódik. Egy fájl egy vagy több ún. blokkra darabolódik, amiket a `DataNode`-ok tárolnak. Az egy fájlhoz tartozó blokkok nem feltétlenül egy `DataNode`-ra kerülnek, sőt, a HDFS megpróbálja a blokkokat minél jobban szétosztani a klaszteren belül, hogy aztán a feldolgozásuk hatékonyabb lehessen (lásd MapReduce feldolgozás). Az egyes blokkok `DataNode`-hoz rendeléséért a `NameNode` felel, a fájlrendszer klienstől érkező olvasási és írási kéréseket pedig a `DataNode`-ok hajtják végre. Ők végzik továbbá a blokkok létrehozását, törlését és replikálását a `NameNode` utasításai alapján.

Fájlrendszer meta-adatok és adatok tárolása

A HDFS névteret a `NameNode` tárolja. Ehhez a `NameNode` egy tranzakció naplót vezet, amit EditLog-nak hívnak, és tartalmazza az összes változást, ami a fájlrendszerben történik. Például, egy új fájl létrehozásának hatására HDFS-en belül a `NameNode` egy bejegyzést ír az EditLog-ba, ami rögzíti ezt. Hasonlóan, ha a replikációs faktort módosítja valaki, az is egy új bejegyzést eredményez az EditLog-ba. A `NameNode` egy lokális fájlt használ a futtató operációs rendszer által nyújtott fájlrendszeren az EditLog tárolására. A teljes HDFS fájlrendszer névtér, valamint az egyes blokkok

és fájlok összerendelése szintén egy fájlban van lementve, aminek a neve FsImage. Az EditLog-hoz hasonlóan, ez is a `NameNode` lokális fájlrendszerében van eltárolva

A `NameNode` a teljes fájlrendszer névterét és a blokk térképet is betölti a memóriába futáskor. Indításkor, vagy előre meghatározott ellenőrzőpontok alkalmával a `NameNode` beolvassa az FsImage és EditLog fájlokat a lokális fájlrendszeréről, alkalmazza az összes EditLog tranzakciót a memóriában tárolt modellre, majd ezt az új állapotot kimenti a lokális fájlra felülírva a régi FsImage fájlt. Ezután az EditLog egy részét el is dobhatja, hiszen a változtatások már tárolásra kerültek az FsImage fájlba. Ezt a folyamatot ellenőrző pontnak (checkpoint) nevezik. Ez a folyamat biztosítja azt, hogy a HDFS-nek mindig naprakész képe van a fájlrendszer állapotáról azáltal, hogy pillanatképeket (snapshot) készít arról, és elmenti az FsImage fájlba. Annak ellenére, hogy az FsImage olvasása igen hatékony, annak az inkrementális szerkesztése nem az. Ezért nem módosítjuk az FsImage fájlt minden szerkesztés után, hanem eltárolják azt az EditLog-ba. A módosítások aztán az ellenőrzési pontnál egyszerre bekerülnek az FsImage fájlba. Az ellenőrző pontok előre meghatározott időközönként vagy tranzakció szám után léphetnek életbe.

A `DataNode`-oknak nincs tudomásuk a HDFS fájlokról, egyszerűen a nekik kiosztott blokkokat tárolják külön-külön fájlokban a saját lokális fájlrendszerükön. A `DataNode` nem minden fájlt ugyanabban a könyvtárban tárol, hanem bizonyos heurisztika alapján megpróbálja kitalálni az egy könyvtáron belül tárolt fájlok optimális számát, és ennek megfelelően hozza létre az alkönyvtárakat. Ennek oka az, hogy a lokális fájlrendszer nem biztos, hogy hatékonyan tudja kezelni, ha egy könyvtárban nagyon sok fájl van. Amikor egy `DataNode` elindul, végig pásztázza az általa tárolt összes fájlt, legenerálja azon HDFS blokkok listáját, amiknek ezek a fájlok megfelelnek, és ezt az információt elküldi a `NameNode`-nak. Ezt úgy nevezik, hogy *Blockreport*.

Kommunikációs protokoll

Minden HDFS kommunikáció TCP/IP protokollon történik. A kliensek egy előre konfigurált TCP porton keresztül kapcsolódnak a `NameNode`-hoz, és az ún *ClientProtocol* segítségével kommunikálnak. A `DataNode`-ok az ún. *DataNode Protocol* segítségével kommunikálnak a `NameNode`-al. Mindkét protokoll egy távoli metódus hívási (Remote Procedure Call, RPC) mechanizmusba van csomagolva, ami egy absztraktabb kommunikációs réteg. A `NameNode` soha nem kezdeményez RPC hívást, mindig csak válaszol a `DataNode`-tól vagy klienstől érkező kérésekre.

📖 2. fejezet
A HDFS tervezési elvei és működésének jellemzői

Tervezési elvek

A HDFS tervezése során jól megfogalmazott célokat követtek, valamint bizonyos feltételezésekkel éltek a rendszer részeivel, felhasználási módjával, stb. kapcsolatban, ami meghatározta a HDFS jelenlegi felépítést. Ezek a feltételezések és célok a következők:


Hardver hiba

A hardver meghibásodás normális eseménynek számít, nem pedig kivételnek. Egy HDFS klaszter több száz vagy akár több ezer szerver gépből is állhat, amelyek mindegyike a fájlrendszer adatainak egy darabját tárolja. A rengeteg komponens ténye, illetve az, hogy ezen komponensek mindegyikében nem elhanyagolható valószínűséggel hiba léphet fel azt jelenti, hogy mindig vannak olyan elemei egy nagy klaszternek, amelyek éppen nem működnek. Pontosan ezért a hibák észlelése, és a rendszer gyors, automatikus helyreállása ezen hibák után a HDFS egyik alapvető architekturális tervezési célja.


Adatok folyam alapú (streaming) elérése

A HDFS-en futó alkalmazások számára szükséges a tárolt adatok folyamszerű (stream) elérése. Ezek az alkalmazások nem általános alkalmazások, amelyek általános fájlrendszeren futnak. A HDFS-t első sorban az adatok kötegelt (batch) feldolgozására tervezték, nem pedig interaktív felhasználásra. A hangsúly a rendszer tervezésekor a nagy adat átviteli sávszélességen volt, nem pedig az adatok elérésének alacsony késleltetésén. A POSIX szabvány számos olyan komoly követelményt támaszt, amelyek betartása nem szükséges a HDFS-en futó alkalmazások számára. Ezért bizonyos POSIX irányelveket feláldoztak a nagy adatátviteli sebesség érdekében.


Nagy adathalmazok

A HDFS-en futó alkalmazások nagyon nagy adathalmazokkal dolgoznak. A HDFS-en tárolt állományok tipikus mérete néhány giga-bájttól akár terra-bájtos méretig is terjedhet. Ehhez nagyon nagy összesített adat átviteli sávszélességet kell biztosítani, és több száz csomópontos méretű klasztereket is könnyedén tudni kell kezelni. A HDFS-nek több tíz millió fájlt is tudnia kell kezelni egyetlen futó klaszteren.


Egyszerű koherencia modell

A HDFS alkalmazásoknak az egyszer írjuk, sokszor elérjük típusú fájl modellre van szükségük. Egy fájlnak, amelyet létrehoztunk, a tartalmát kiírtuk, majd lezártuk, későbbi módosítására nincs szükség, esetleg csak új adat hozzáfűzésre a fájl végéhez, vagy a fájl valamekkora részének levágására kell felkészülni. A fájlok tartalmához való adathozzáfűzés támogatott, de a tartalmának tetszőleges ponton történő szerkesztése nem lehetséges. Ez a feltételezés leegyszerűsíti az adat koherencia kérdését, valamint lehetővé teszi a szélessávú adatelérést. Egy MapReduce vagy egy web feldolgozó alkalmazás tökéletesen illeszkedik ehhez a modellhez.


A számítás mozgatása olcsóbb az adatok mozgatásánál

Egy alkalmazás által kért számítás sokkal hatékonyabban elvégezhető, ha azt az adatokhoz közel futtatjuk, amiken dolgozik. Ez különösen igaz abban az esetben, amikor az adatok mérete hatalmas. Ez a megközelítés minimalizálja a hálózati adatforgalmat, és nagyobb adatáteresztő képességet eredményez. Az alapvető feltételezés tehát az, hogy sokkal olcsóbb az adatfeldolgozást végző számítás átmozgatása arra a csomópontra, ahol a feldolgozandó adat található, mint az adat átmozgatása oda, ahol a számítás fut. A HDFS megfelelő interfészeket nyújt ahhoz, hogy az alkalmazások átmozgathassák magukat a feldolgozandó adat közelébe.


Heterogén hardver és szoftver platformok közötti átjárhatóság

A HDFS-t úgy tervezték, hogy könnyen átvihető legyen egyik platformról a másikra. Ezáltal a HDFS számos platformon nyújt megoldási lehetőséget számos alkalmazás típus számára.

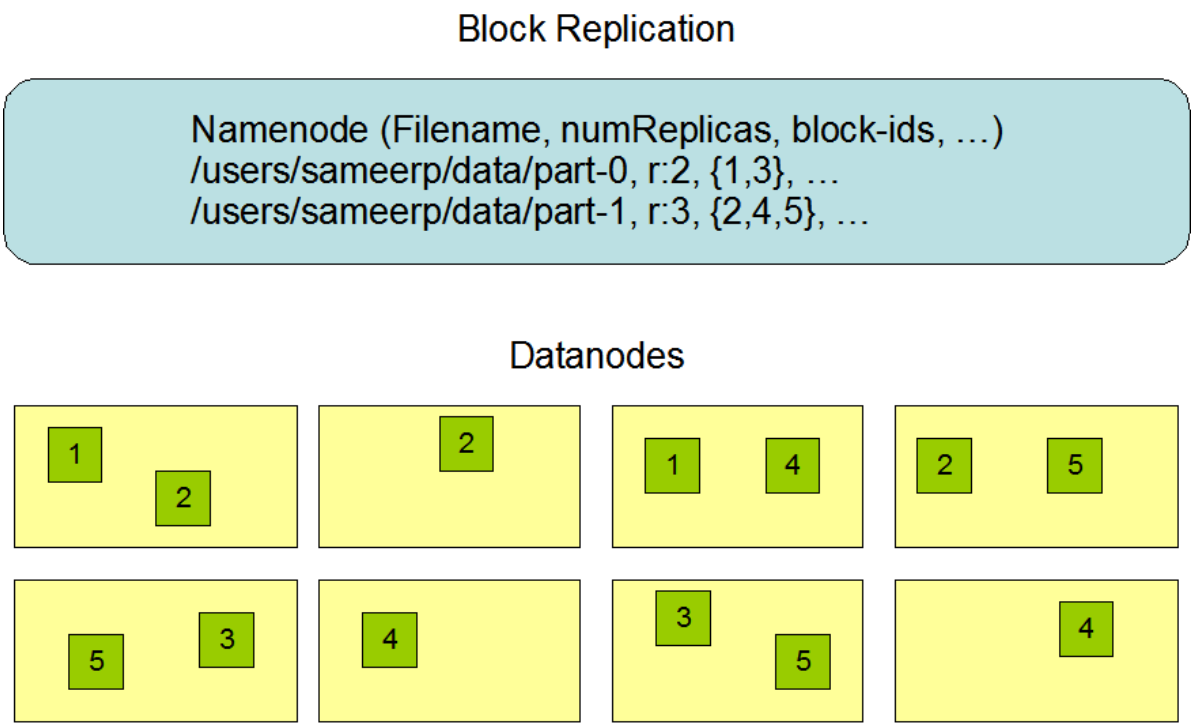
Adat szervezés és adat replikáció

A HDFS-t hatalmas fájlok megbízható módon történő tárolására tervezték, számítógépek klaszterén. Minden fájlt blokkok sorozataként tárol, ahol minden egyes blokk többszörösen is tárolásra kerül, hogy a rendszer hibatű legyen. Mind a blokk méret, mind a replikációs faktor konfigurálható.

Egy fájl minden blokkja egyforma méretű, kivéve legfeljebb az utolsó blokkot. Egy alkalmazás megadhatja egy fájl másolatainak számát. Ezt a fájl létrehozásakor kell megadni, de később is módosítható. A fájlok szigorúan csak egyszer írhatók HDFS-en, és csak egyetlen írója lehet egyszerre.

Minden döntést a blokkok replikálásáról a `NameNode` hoz meg, aki periodikusan fogadja a `DataNode`-ok által küldött *Heartbeat* és *Blockreport* üzeneteket. A Heartbeat üzenet jelzi, hogy a `DataNode` megfelelően működik, míg a Blockreport a `DataNode` által tárolt összes blokk listáját tartalmazza.

Az alábbi ábra egy példát mutat fájlok több `DataNode`-on történő tárolására, és az egyes blokkok replikálására.



> https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/images/hdfsdatanodes.png

Az adattárolás és replikák kezelése nem triviális feladat, a HDFS az alábbi elveket követi:

- *Replika elhelyezés:* a másolatok elhelyezése HDFS-en belül kritikus a megbízhatóság és teljesítmény szempontjából. Ebből a szempontból a HDFS elkülönül minden más megoldástól az ún. rack-aware (tárolási helyet figyelembe vevő) tárolási elve miatt. Ez a stratégia megpróbálja az egymáshoz tartozó blokkokat egy rack-en elhelyezkedő `DataNode`-okon tárolni, amik között nagy adatátviteli sebesség érhető el, míg a replikákat lehetőleg másik rack-en tartani, hogy amennyiben hardver hiba lép fel, lehetőség a másolatokat az ne érintse.
- *Replika kiválasztás:* a globális sávszélesség és az olvasási késleltetés minimalizálása érdekében a HDFS megpróbálja az adatot arról a rack-ről olvasni, amelyik az olvasóhoz a legközelebb van. Amennyiben az olvasó node-ján levő rack-ben megtalálható a kért adat másolata, az olvasás onnan történik. Amennyiben az adatok több adattárházon is átívelnek, az adat kéréssel azonos tárház preferált egy távolival szemben.
- *Safemode:* induláskor a `NameNode` egy speciális állapotba, a `Safemode`-ba kerül. Ebben az állapotban nem történik replikálás, hanem a `NameNode` a Heartbeat és Blockreport üzeneteket várja. Minden blokknak van egy minimális replikálási értéke, és egy blokk akkor tekinthető biztonságosan tároltnak, ha legalább ennyi `DataNode` bejelentkezik a blokk tárolásával. Miután a blokkok egy meghatározott részéről kiderül, hogy biztonságosan tárolt

(plusz 30 másodperc múlva) a HDFS kikapcsolja a Safemode-ot. Ezután megkezdi a nem biztonságosan tárolt blokkok replikálását a szükséges minimum példányszámig.
- *Adat blokkok:* a HDFS hatalmas méretű fájlok kezelésére lett tervezve. Azon alkalmazások számára ideális a HDFS használata, amelyek ilyen hatalmas méretű adatokon dolgoznak, és az adatokat egyszer írják, de később sokszor olvashatják, és stream sebességű hozzáférésre van szükségük.
- *Replikáció összefűzése:* replikált adatok írása esetén az adat egy csővezetéken (pipeline) keresztül továbbítódik egyik `DataNode`-ról a másikra. Azaz ha pl. egy 3-as replikációs faktorral rendelkező fájl blokkjait megkapja egy `DataNode`, először kiírja a fájlrendszerére az összes blokkot, majd egy csővezetéken a következő target `DataNode`-nak továbbítja azt, aki szintén kiírja és továbbítja az utolsó `DataNode`-nak.

Hibatűrés

A HDFS egyik alapvető célkitűzése az adatok megbízható tárolása akkor is, ha a klaszteren belül hibák jelentkeznek. Alapvetően három fajta meghibásodás történhet, `NameNode` meghibásodás, `DataNode` meghibásodás vagy hálózati problémák. A következőkben az egyes konkrét hibák lehetőségeit soroljuk fel, és azt, hogy a HDFS hogyan védekezik velük szemben:

- *Adat lemez hiba, Heartbeats és újra replikálás:* minden `DataNode` időközönként Heartbeat üzeneteket küld a `NameNode` felé. Egy hálózati probléma okozhatja azt, hogy a `DataNode`-ok egy halmaza elérhetetlenné válik a `NameNode` számára. Ezt a hibát a `NameNode` a Heartbeat üzenetek kimaradása által észleli. Az ilyen `DataNode`-okat a `NameNode` nem élőnek jelöli, és nem küld számukra további I/O kéréseket. Mivel a nem élő `DataNode`-okon lévő adatok többé nem elérhetők a HDFS számára, bizonyos fájlok replikációs faktora a meghatározott érték alá eshet. Az ilyen fájlokat a `NameNode` újra, más `DataNode`-okon replikálja. Az újra replikálás több okból is történhet, pl. a `DataNode` elérhetetlenné válik, a replikált adat sérül, meghibásodik a merevlemez, megnövelték a replikációs faktort, stb.
- *Klaszter kiegyensúlyozás:* a HDFS támogatja az adatok kiegyensúlyozását, ami azt jelenti, hogy automatikusan áthelyezhet adatokat egyik `DataNode`-ról a másikra, ha egy `DataNode`-on egy határérték alá csökken a szabad tárhely mértéke. Bizonyos fájlok iránti hirtelen megnövekedett igény miatt is megnövelheti a replikák számát a HDFS és áthelyezheti azokat más `DataNode`-okra (ez nincs teljes körűen implementálva egyelőre).
- Adat integritás: előfordulhat, hogy egy adat blokk hibásan érkezik meg egy `DataNode`-ról, például a tároló eszköz meghibásodása miatt, hálózati adatvesztés miatt, vagy szoftverhiba miatt. A HDFS kliens ellenőrző kódot számít a fájlok tartalmára, amit létrehozáskor külön eltárol egy rejtett fájlban. Ezt az ellenőrző kódot aztán felhasználja az adat olvasásakor arra, hogy megállapítsa sértetlen-e a kapott adat. Amennyiben eltérést tapasztal, megpróbálja az adatot egy másik `DataNode`-on lévő replikából betölteni.
- *Meta-adat lemez meghibásodás:* az FsImage és az EditLog központi adat struktúrák a HDFS működése szempontjából, amelyek meghibásodása az egész HDFS klaszter leállásához vezethetnek. Ennek kiküszöbölése érdekében a `NameNode` konfigurálható úgy, hogy ezen fájlokból több példányt is tároljon, amelyeket aztán egyszerre szinkron módon frissít. Ez azonban teljesítmény csökkenéssel járhat, azonban ennek mértéke elenyésző, hiszen a HDFS adat intenzív ugyan, de nem meta-adat intenzív. E mellett a `NameNode`-ok több példányban való futtatása is támogatott.
- *Snapshot-ok:* az adatok lementhetők egy bizonyos időpillanatban, ez a snapshot. A snapshot-ok egyik felhasználása a HDFS példány visszaállítása egy korábbi jól működő állapotba meghibásodás után.

📖 3. fejezet
HDFS műveletek és nagy mennyiségű adat mozgatása

HDFS kliens parancsok

A HDFS hasonló hierarchikus fájl szerkezetet alkalmaz, mint a legtöbb fájlrendszer. Nem teljesen POSIX kompatibilis, de sok POSIX műveletet támogat a kliens segítségével. Korábbi olvasóleckében láthattunk már példát ilyen fájlműveletekre, néhány példa:

- Könyvtár listázása: `$ bin/hadoop fs -ls /path/to/dir`
- Fájl tartalmának kiíratása: `$ bin/hadoop fs -cat /path/to/file/sample.txt`
- Lokális fájlrendszerről fájl másolása HDFS-re: `$ bin/hadoop fs -put/path/to/local/file1 /path/to/hdfs/file1`
- Teljes lista: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html#copyFromLocal

Relációs adatok mozgatása Apache Sqoop segítségével

Sokszor előfordul, hogy nagy mennyiségű adat áll rendelkezésünkre, de valamilyen oknál fogva azt nem tudjuk egy sima `put` műveletre a HDFS-re másolni feldolgozásra. Egyik ilyen tipikus eset, hogy az adataink egy relációs adatbázisban vannak tárolva. Ehhez a felhasználási esethez használható az Apache Sqoop [3] eszköz, mely SQL to Hadoop implementáció, azaz lehetővé teszi az adatok relációs adattáblából HDFS-re történő importálását, vagy pedig a HDFS állományok exportálását relációs adattáblába.

A Sqoop egy parancssori eszköz, amely viszonylag egyszerűen használható [4], például az alábbi két példa mutatja egy MySQL adattábla betöltését HDFS-re, valamint egy HDFS állomány kimentését adatbázisba.

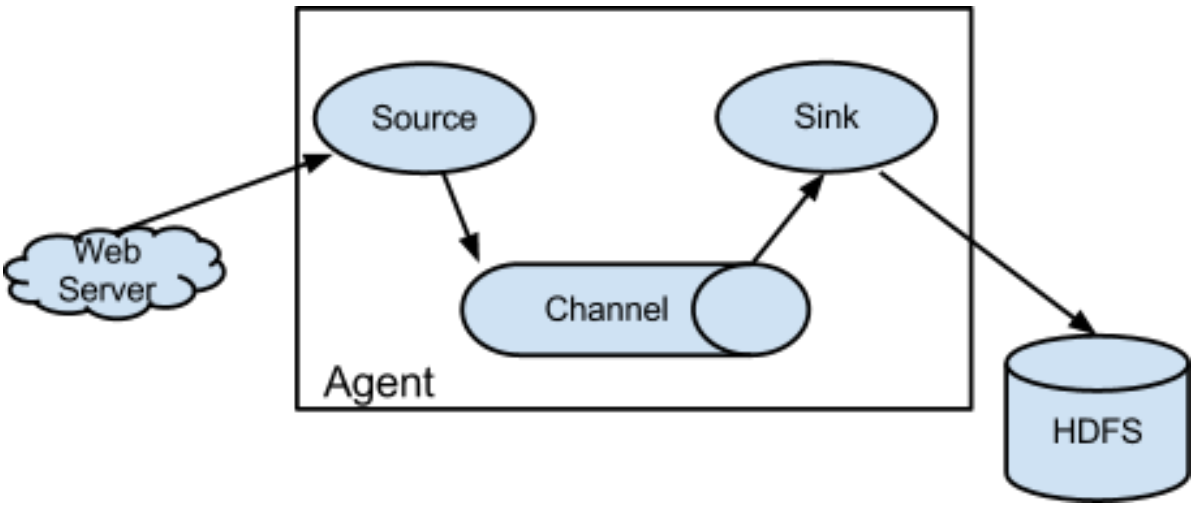
```
$ sqoop import --connect jdbc:mysql://database.example.com/employees
```

```
$ sqoop export --connect jdbc:mysql://db.example.com/foo --table bar --export-dir /results/bar_data
```

Adat stream-ek mentése HDFS-re Apache Flume segítségével

Egy másik tipikus eset, amikor nem működik az adatok egyszerű `put` művelettel történő másolása, ha az adataink folyamatosan, stream szerűen állnak elő. Például egy web alkalmazás log bejegyzései. Ilyenkor az adatok folyamatos HDFS-re történő tárolására van szükség, amihez több eszköz is nyújt támogatást, ezek közül az egyik az Apache Flume [5].

A Flume segítségével adat forrásokat (source) és adatnyelőket (sink) definiálhatunk, az adat forrásokból stream szerűen érkeznek az események, amelyeket egy csatorna az adatnyelőhöz vezet, ami eltárolja azt. A Flume számos adatforrást [6] és adatnyelőt támogat [7], ami persze bővíthető is. Számunkra a legérdekesebb a HDFS adatnyelő, amely tetszőleges stream forrás adatait valós időben tárolni tudja HDFS-re (lásd ábra). A Flume agent működéséről és a konfigurációjának módjáról a gyakorlathoz tartozó olvasólecke tartalmaz további hasznos információt.



Ellenőrző kérdések

1. Mire szolgál egy HDFS klaszteren belül a `NameNode` és mire a `DataNode`?
2. Milyen meta-adatokat tárol a `NameNode` a HDFS fájlrendszerről és hogyan frissíti azokat?
3. Milyen hálózati kommunikációs protokollt használnak egymás közt a HDFS komponensei?
4. Sorolj fel legalább három tervezési elvet, amit figyelembe vettek a HDFS tervezésénél!
5. Milyen módon tárolja a `DataNode` a fájlokat?
6. Mit jelent az adat replikálás, és hogyan működik?
7. Sorolj fel néhány lehetséges meghibásodási pontot egy HDFS klaszteren belül és magyarázd el milyen módon védekezik az ellen a HDFS?
8. Sorolj fel három fájlműveletet, amit a HDFS kliens támogat a fájlrendszer kezeléséhez!
9. Mire szolgál az Apache Sqoop?
10. Mire használható az Apache Flume?

Referenciák

[1] https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[2] https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html#copyFromLocal

[3] https://sqoop.apache.org/

[4] https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html

[5] https://flume.apache.org/

[6] https://data-flair.training/blogs/flume-source/

[7] https://data-flair.training/blogs/flume-sink/