Nagy adathalmazok elosztott feldolgozása

Dr. Hajdu András, Debreceni Egyetem, Informatikai Kar

Mi a Big Data?

- Nem létezik egzakt Big Data definíció.
- A „big data” kifejezést a mai értelemben először *Cox* és *Ellsworth* (NASA) használta 1997-ben a szuperszámítógépes szimulációik során előállt extrém adatmennyiség feldolgozási kérdéseinek leírására.

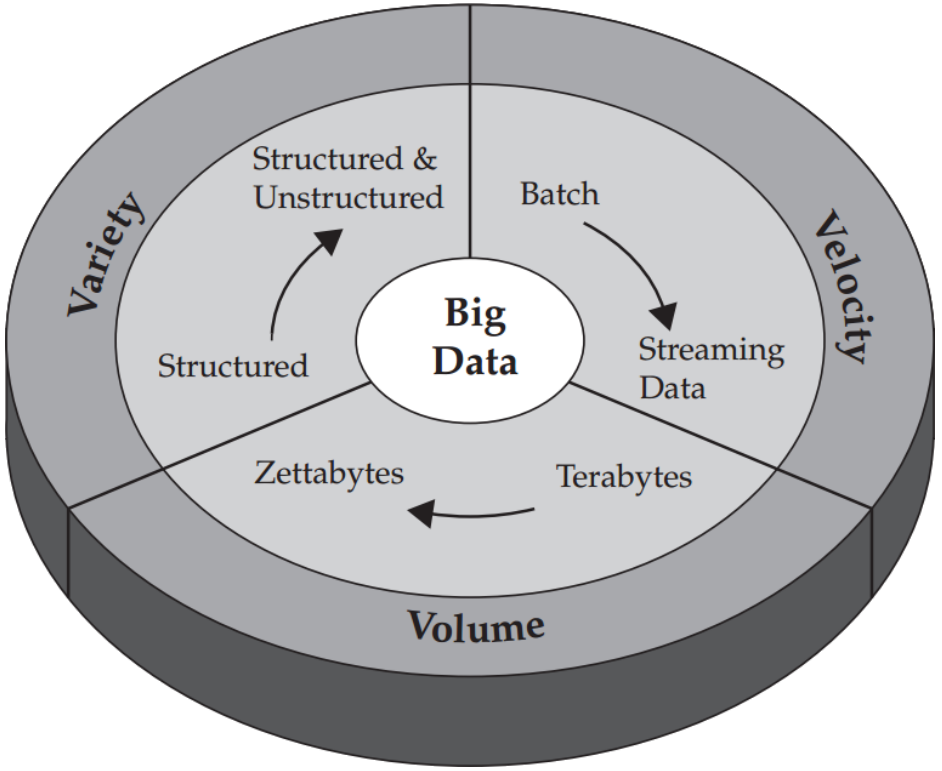
Mi a Big Data?

- „A big data olyan adat, ami meghaladja a hagyományos adatbázis rendszerek feldolgozási kapacitását. Az adat túl nagy, túl gyorsan mozog, vagy nem illeszkedik az adatbázis architektúra megkötéseihez. Ahhoz, hogy értéket nyerjünk ki ezekből az adatokból, alternatív feldolgozási módszert kell választani.”
  E. Dumbill: „Making sense of big data”, Big Data, vol. 1, no. 1, 2013

- „Big Data az, amikor az adat mérete maga is a probléma részévé válik.”
  Mike Loukides, O’Reilly

A Big Data jellemzői

- **Gartner „3V”:**
  - volume – terjedelem
  - velocity – sebesség
  - variety – sokszínűség
- **IBM „4V”:**
  - 3V
  - veracity – valódiság



Forrás: *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, McGraw Hill, 2011*

A Big Data jellemzői

- **Terjedelem (Volume):**
  - Az előállt nyers adat mennyisége.
  - Jellemző az akár exabájtos nagyságrend.
  - Az adat terjedelme nem csak tárolási kérdés (pl. milyen adat kerüljön tárolásra és meddig, ritkán használt adatok periodikus törlése stb.), jelentős feldolgozási problémákat is felvet.

A Big Data jellemzői

- **Terjedelem (Volume):**
  - Az IDC *„The Digital Universe in 2020”* tanulmánya szerint a digitális univerzum megközelítőleg 4 zettabájt méretű (2013) és a jelenlegi növekedési ráta alapján 2020-ra elérheti a 35 zettabájtos méretet is.

    Például:
    - Facebook: exabájt kapacitású cold storage adatközpont átadása 2013 októberében
    - CERN Data Center: 220 PB (2013 július)

A Big Data jellemzői

- **Sebesség (Velocity):**
  - Az adat előállásának és változásának gyorsasága, illetve az a sebesség amivel az adatot fogadni, értelmezni és feldolgozni kell.
  - Jellemzően közel valósidejű vagy valósidejű/stream jellegű adatfogadásra és -feldolgozásra van szükség.

A Big Data jellemzői

- **Sebesség (Velocity):**
  - Fontos az adatok „időértéke” is:
    az adott pillanatban rendelkezésre álló adatok valós idejű elemzése lehetővé teszi jobb adatvezérelt döntések meghozását, például termékajánlás, biztonsági kockázatok elemzése, stb. esetén.

A Big Data jellemzői

- **Sebesség (Velocity):**
  - Például:
    - A New York Stock Exchange naponta átlagosan 1 TB kereskedelmi információt gyűjt össze és tárol.
    - Az LHC másodpercenként 1 PB adatot állít elő → ennek kb. 1%-a kerül tárolásra.
    - A SKA Telescope az átadása után várhatóan napi 1 EB adatot fog előállítani.

A Big Data jellemzői

- **Sokszínűség (Variety):**
  - Az adatok forrásának, strukturáltságának és típusának változatossága.
  - Strukturált adat:
    - Jól definiált formában állnak rendelkezésre (pl. relációs vagy OO adatbázisokban).
    - A tárolt adatoknak csak kb. 15%-a strukturált.

A Big Data jellemzői

- **Sokszínűség (Variety):**
  - Strukturálatlan adat:
    - Nem követ rögzített formátumot, sorrendet vagy egyéb szabályt.
    - Nem megjósolható.
    - Pl.: emberek által generált tartalmak (dokumentumok, képek, hangok, videók)

A Big Data jellemzői

- **Sokszínűség (Variety):**
  - Félig strukturált adat:
    - Heterogén forrásokból származó adatok integrációja és megosztása rendszerek között → egyre jelentősebb.
    - Absztrakt leírás: gráfokkal (csomópont - adatelem, élek - relációk, címkék - attribútumok).
    - Pl.: XML állományok + dokumentumok

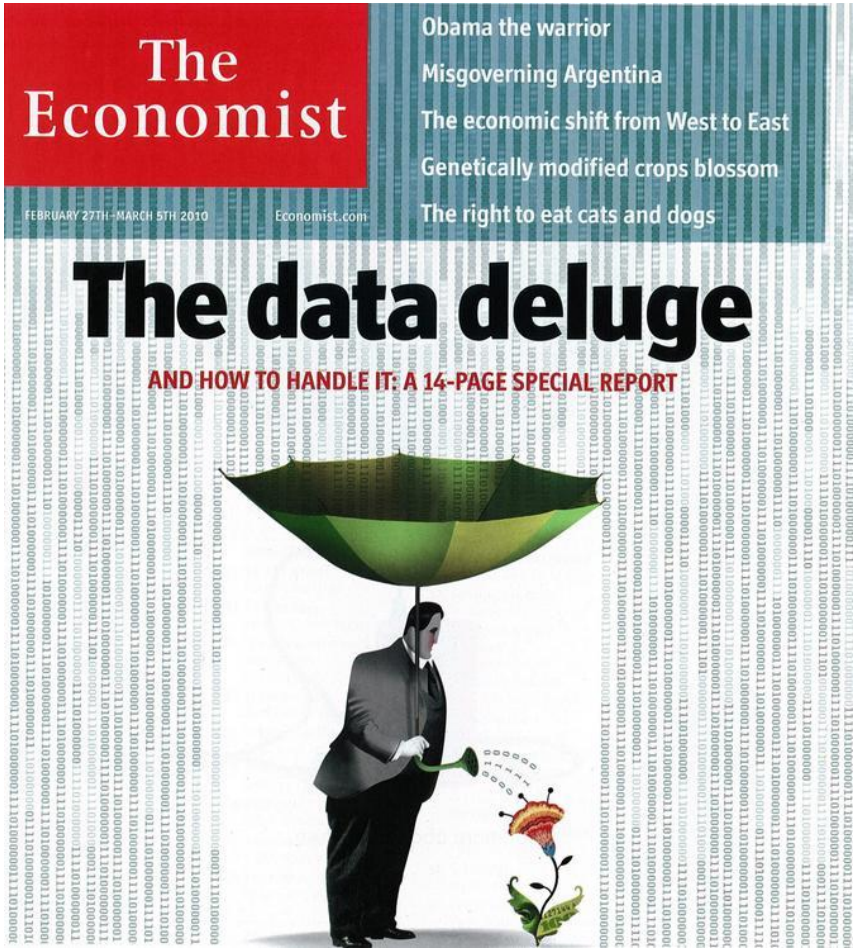
A Big Data jellemzői

- **Valódiság (Veracity):**
  - A rendszer által feldolgozott és tárolt adatok minősége, pontossága, és származásának megbízhatósága.
  - A 3V eredményeként nehéz az adatok valódiságát biztosítani.
  - Jelentős kérdés az adatok valódiságának ellenőrzése.
    - Pl.: Az USA gazdaságának évi 3 milliárd dolláros költséget jelent az adatok karbantartása és „tisztítása” (2012).

A Big Data jellemzői

- **Valódiság (Veracity):**
  - Kezelendő problémaforrások: ellentmondásos vagy többértelmű adatok, duplikátumok, modell approximációk, feldolgozási késleltetésből eredő hibák, hamis adatok, spam, stb.
  - Crowd computing megoldás: egy kellően nagy csoport, személyenként kis erőbefektetéssel is képes olyan problémákat megoldani ami szoftveresen nem lenne lehetséges. Pl.: improve-and-vote rendszerek.

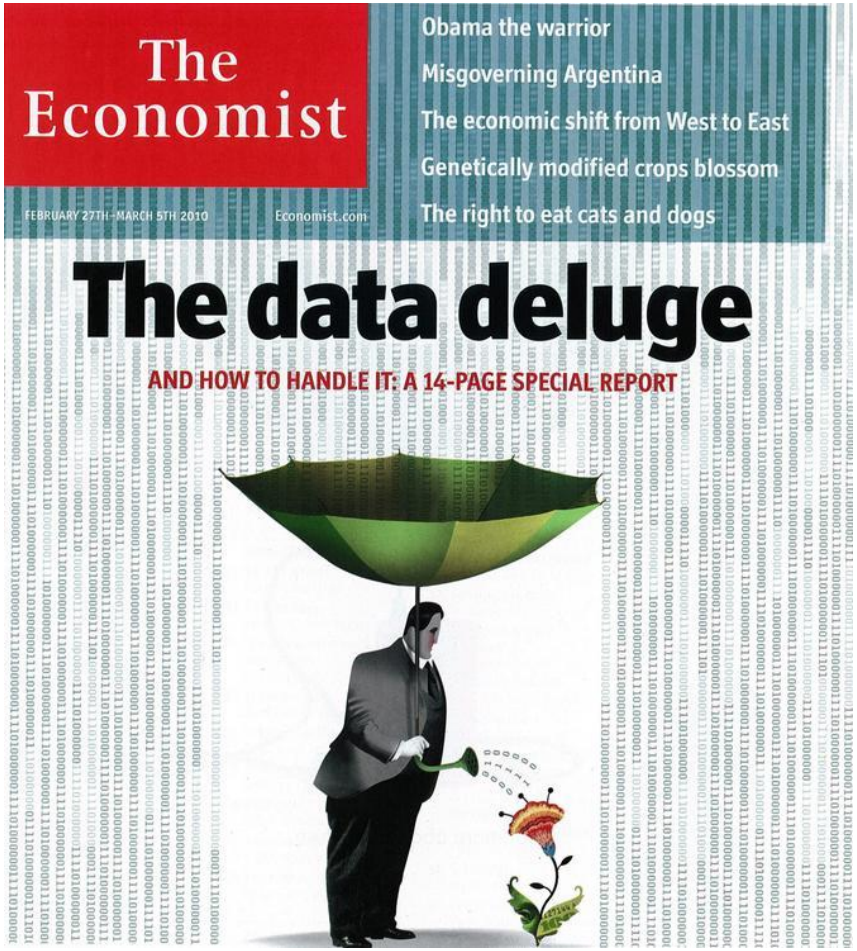
A Big Data jelentősége



„Az adatáradat” - The Economist, 2010. február 25.

- A számító- és tárolókapacitás árának csökkenésével egyre nagyobb mennyiségű adat gyűjtése és tárolása válik lehetővé.
- A hálózatba kapcsolt eszközök számának jelentős növekedésével az adatok összegyűjtése egyre könnyebbé válik.

A Big Data jelentősége



„Az adatáradat” - The Economist, 2010. február 25.

- Leggyorsabban növekedő adatforrások:
  - Tranzakciós és rendszernaplók
  - Különböző eszközök szenzorai
  - Social media: mindenki számára lehetővé vált tartalmak (szöveg, hang, kép, videó, GPS koordináták) könnyű és gyors megosztása.
- **Cél:** érték kinyerése a rendelkezésre álló nagy mennyiségű adatból.

A Big Data néhány alkalmazási területe

- **Tudományos kutatás**
  - Az összegyűjthető és előállítható adatok mennyiségének robbanásszerű növekedése számos tudományterületen tette szükségessé a Big Data eszközök alkalmazását.
  - A kollaboráció és az interdiszciplinaritás egyre inkább előtérbe kerül → különböző adatok összekapcsolása.

A Big Data néhány alkalmazási területe

- **Tudományos kutatás**
  - Konkrét hipotézisek ellenőrzése mellett új tudást is ki lehet nyerni a rendelkezésre álló adatokból → hipotézismentes adatvezérelt kutatás.

A Big Data néhány alkalmazási területe

- **Tudományos kutatás – példák:**
  - Élettudományok: A biotechnológia fejlődésével és a szekvenálás költségének csökkenésével egyre több adat áll rendelkezésre, amit a hagyományos módszerekkel már nem lehet vizsgálni.
    Alkalmazási területek: genetika, metagenomika, proteomika, evolúciós biológia, farmakológia, stb.

A Big Data néhány alkalmazási területe

- **Tudományos kutatás – példák:**
  - Csillagászat: Square Kilometer Array (SKA) átadása után petabájtos nagyságrendű adatok feldolgozására lesz szükség.
    Ennek célja: az általános relativitáselmélet tesztelése, kozmológiai kutatás, sötét energia vizsgálata, galaxisok létrejöttének vizsgálata, stb.

A Big Data néhány alkalmazási területe

- **Egészségügy és orvostudomány**
  - Elektronikus beteg rekord (EPR): egy beteg kórtörténetét tartalmazza.
    Diagnózisok, alkalmazott gyógyszerek, terápiák leírása és eredményei, védőoltások dátuma, allergiák és érzékenységek, radiológiai felvételek, laboratóriumi és egyéb tesztek eredményei, stb.

A Big Data néhány alkalmazási területe

- **Egészségügy és orvostudomány**
  - Az egészségügyi folyamatokra jellemző, hogy általában több szervezeti egység vesz részt bennük, amelyek saját IT infrastruktúrával rendelkeznek
    → heterogén adatforrásokból kell jellemzően félig strukturált adatokat kinyerni és feldolgozni.

A Big Data néhány alkalmazási területe

- **Egészségügy és orvostudomány**
  - Ezekből adatbányászati eszközökkel lehet új ismeretet származtatni. Például:
    - betegségek közötti összefüggések,
    - krónikus betegségek kialakulásának okai,
    - fertőzések terjedésének megfigyelése és előrejelzése,
    - régiók jellemző megbetegedései (környezeti hatások feltárása) stb.

A Big Data néhány alkalmazási területe

- **Egészségügy és orvostudomány**
  - Prediktív modellek létrehozásával lehetségessé válhat valós idejű döntéstámogatást nyújtani különböző egészségügyi osztályozási problémák során, diagnózis felállításában, az egészségügyi szolgáltatás minőségének javításában és annak költségeinek csökkentésében is.

A feldolgozás kérdései

- **Adattárolás**
  - Hagyományos relációs adatbázis-kezelő rendszerek (RDBMS) előnyei:
    - komplex tranzakciók,
    - másodpercenként akár több ezer lekérdezés kezelése,
    - hatékony lekérdező nyelv.
  - De: RDBMS-ekre gyakran nagyon nehéz leképezni a Big Data problémákat.

A feldolgozás kérdései

- **Adattárolás**
  - Ennek okai:
    - az alkalmazott séma már az adat beérkezése előtt rögzített,
    - nagy mennyiségű félig strukturált és strukturálatlan adat kezelése nehézkes,
    - az ACID tulajdonságok sok esetben nem praktikusak,
    - az architektúra nem mindig teszi lehetővé a könnyű skálázhatóságot (pl. nehezen partícionálható, inhomogén adatok esetében).

A feldolgozás kérdései

- **Hardver architektúra**
  - A hagyományos RDBMS-ekhez használt hardverek leggyakrabban nagy teljesítményű, párhuzamos adatfeldolgozásra optimalizált számítógépek, amelyekhez külső tárolók kapcsolódnak.

A feldolgozás kérdései

- **Hardver architektúra**
  - Ez az architektúra hatékony véletlenszerű adatelérést és párhuzamos feldolgozást biztosít, de magas költséggel jár.
  - A számítóteljesítmény nehezebben skálázható, mint a tárolókapacitás.
  - A kommunikáció a tároló és a feldolgozást végző gép között overhead.

A feldolgozás kérdései

- **Hardver architektúra**
  - Nagy adatmennyiség feldolgozásához kis költséggel bővíthető, jól skálázható hardverre van szükség → *klaszter architektúra*.
    - Kisebb teljesítményű általános célú számítógépek összekapcsolása hálózaton keresztül.
    - Jellemzően minden csomópont egy elosztott fájlrendszeren (pl. NFS) tárolt adatokon dolgozik.

A feldolgozás kérdései

- **Hardver architektúra: klaszter**
  - Számítás-intenzív feladatokhoz jól illeszkedik.
  - De: mivel az adatok egy elosztott fájlrendszeren érhetők el, ezért jelentős a hálózati adatforgalom
    → ez szűk keresztmetszet lehet nagy adathalmazok esetén.

A feldolgozás kérdései

- **Kihívások**
  - Adatmodell: hogyan osztható szét az adat hatékonyan a klaszter csomópontjaira?
  - Programozási modell: hogyan készíthető könnyen olyan hatékony alkalmazás, amely képes a klaszteren futva az adatokat feldolgozni?
  - Hogyan kezeljük a gépek meghibásodásait (csomópont kiesés, diszk hiba, stb.)?

Az Apache Hadoop

Az Apache Hadoop

- Az **Apache Hadoop** egy nyílt forráskódú, hibatűrő, elosztott rendszer nagy adathalmazok tárolásához és feldolgozásához.

- A Hadoop keretrendszere
  - elosztott adatmodellt,
  - elosztott programozási modellt, és
  - klaszter erőforrás-kezelési megoldást
- is biztosít.

Az Apache Hadoop

- Adatmodell: *HDFS* elosztott fájlrendszer
  - Egyszerű adatmodell, bármilyen típusú és formátumú adat illeszkedik hozzá.
- Programozási modell: *MapReduce*
  - Automatikus feladatelosztást és párhuzamosítást tesz lehetővé, így egyszerű absztrakciót biztosít a fejlesztők számára.
- Erőforrás kezelés: *YARN*
  - Feladatai: klaszter erőforrás-menedzsment, a skálázhatóság támogatása, job ütemezés és kezelés.

A Hadoop és RDBMS-ek összehasonlítása

RDBMS: schema-on-write

- A sémát az adatok bevitele előtt létre kell hozni.
- A betöltés során az adatot az adatbázis belső struktúrájának megfelelő formátumba kell transzformálni.
- A séma bővítése vagy módosítása szükséges minden új típusú adat bevitele előtt egy rekordba.

Gyors olvasás, jó kezelhetőség

Hadoop: schema-on-read

- Az adatot egyszerűen fel kell másolni a tárolóra, nincs szükség transzformációra.
- Új típusú adatok bármikor bekerülhetnek.
- Az olvasás/feldolgozás során kell egy megfelelően implementált ETL eszközt alkalmazni.

Gyors írás, flexibilitás

Az Apache Hadoop

- **Háttér:**
  - Az Apache Hadoop a Google File System (2003) és a MapReduce programozási paradigma (2004) nyílt forráskódú implementációja, ami a Nutch keresőalgoritmus (2005) skálázási problémáinak megoldásra készült.
  - *2008:* az Apache Software Foundation önálló, kiemelt projektje lett.

Az Apache Hadoop

- **Háttér:**
  - *2009:* Jim Gray's Sort Benchmark – 1 TB rendezése 62 másodperc alatt, 1 PB rendezése 16,25 óra alatt Hadoop-pal a Yahoo-nál.
  - Napjainkban az egyik legelterjedtebb rendszer nagyon nagy mennyiségű adat elosztott feldolgozáshoz.

Apache Hadoop – hatékonyság

- **A Hadoop hatékonyan alkalmazható,**
  - ha a feldolgozandó adat nagy méretű, komplexitású, vagy gyorsan generálódik,
  - ha összetett adatelemzésre van szükség, és
  - ha az adatokon végzendő elemzési feladat időben változhat.
- **De: a Hadoop nem hatékony**
  - relációs adatbázisok kiváltására, és
  - ad-hoc jellegű elemzésekhez.

A Hadoop Distributed File System (HDFS)

- A **Hadoop Distributed File System (HDFS)** egy elosztott fájlrendszer, amely a Google File System-en (GFS) alapul.
- **Rövid áttekintés**
  - A HDFS a Hadoop fájlrendszer komponense, amely egy már létező fájlrendszer (pl. Ext3) felett működik.

A Hadoop Distributed File System (HDFS)

- **Rövid áttekintés**
  - A fájlrendszer metaadatait egy dedikált csomóponton elkülönítve, míg az alkalmazások adatait a klaszter többi csomópontján tárolja.
  - A fájlrendszerbeli kommunikáció TCP/IP-alapú protokoll használatával történik.
  - A tárolás szervezésével támogatja a nagy sebességű elosztott feldolgozást.

HDFS – architektúra

- **Tervezési célok**
  - Hibatűrés:
    - Jellemzően nagy, olcsóbb szerverekből álló klaszterek támogatása, ahol a hardveres és hálózati hibák gyakoribbak.
  - Nagy fájlok kezelése:
    - Akár terabájtos nagyságú fájlok kezelésére is szükség lehet.

HDFS – architektúra

- **Tervezési célok**
  - Feldolgozási teljesítmény:
    - Ahol csak lehetséges, ki kell használni az adatok és a feldolgozást végző csomópontok hálózaton belüli elhelyezkedését.
  - Hordozhatóság:
    - Heterogén hardver és szoftver platformok támogatása, könnyű bővíthetőség és skálázhatóság.

HDFS – architektúra

- **Tervezési alapkoncepciók**
  - Nagy áteresztőképesség előtérbe helyezése az alacsony késleltetéssel szemben:
    - A fájlrendszer szervezése a szokásos gyors véletlenszerű hozzáférés helyett stream hozzáférésre van optimalizálva.
  - Egyszerű konkurencia modell:
    - Egy stream író – több, párhuzamos stream olvasó
    - Zárolásmentes írás és olvasás
    - De: ezek miatt nincs teljes POSIX-kompatibilitás

HDFS – architektúra

- **Tervezési alapkoncepciók**
  - Feldolgozás az adat mellett:
    - A feldolgozást végző programot általában kisebb költséggel lehet mozgatni, mint az feldolgozandó nagy mennyiségű adatot.
  - A fájlok szervezésekor a minél nagyobb teljesítmény elérése érdekében figyelembe kell venni az általános klaszter-topológiát:
    - Egy nagy klaszter jellemzően több rack-ből áll.

HDFS – architektúra

- **Tervezési alapkoncepciók**
  - A rack-ek közötti kommunikációhoz általában legalább két switch-en át kell menni a forgalomnak → A rack-ek közötti késleltetés nagyobb, mint a rack-en belüli.
  - Az aggregált rack-en belüli sávszélesség sokkal nagyobb, mint a rack-ek közötti sávszélesség.
- A fájlrendszert rack-en belüli párhuzamos hozzáférésre kell optimalizálni, ahol lehetséges.

HDFS – logikai szervezés

- HDFS a mester-dolgozó szervezési mintát implementálja:
  - Egy mester csomópont: NameNode
    - HDFS metaadatainak tárolása a helyi fájlrendszeren, a fájlrendszer menedzselése
  - Több dolgozó csomópont: DataNode-ok
    - HDFS adatainak tárolása a helyi fájlrendszeren.
    - A kliens alkalmazások a DataNode-okon futnak és nem különálló gépeken.
    - Minden csomópont lehetőség szerint a helyi fájlrendszerén tárolt blokkokon dolgozik.

HDFS – fájlszervezés

- **NameNode**
  - Kezeli a fájlrendszer névterét: hierarchikus névtér (fájl illetve könyvtár fa).
  - Kezeli a fájlokhoz és könyvtárakhoz tartozó metaadatokat (pl. a blokkok fizikai elhelyezkedése, stb.).
  - Koordinálja az adatok DataNode-okon történő elhelyezését.

HDFS – fájlszervezés

- **NameNode**
  - Limitált hozzáférés-vezérlést biztosít: a fájlrendszerhez történő hozzáférés szabályozása kevésbé kritikus, ha a kliensek alapvetően megbízhatóak.
  - Egy NameNode jelenléte nélkülözhetetlen a HDFS működéséhez, ennek kiesése elérhetetlenné teszi a teljes fájlrendszert.

HDFS – fájlszervezés

- **NameNode**
  - Ezek periódikusan egy-egy perzisztens fájlba kerülnek mentésre, amelyek egy szerkesztési napló segítségével vannak naprakészen tartva.
  - A NameNode kiesésének kiküszöbölésére stand-by NameNode másolatokat lehet alkalmazni, de ehhez külső eszköz szükséges (pl. ZooKeeper); a HDFS jelenleg nem biztosít ilyet.

HDFS – fájlszervezés

- **NameNode**
  - A fájlok nagy méretű blokkokban kerülnek tárolásra, az egész klaszteren elosztva:
    - Általában 64 vagy 128 MB egy blokk mérete (fájlonként konfigurálható)
    - Minden blokk a DataNode-ok helyi fájlrendszerén, fájlként kerül tárolásra

HDFS – fájlszervezés

- **NameNode**
  - Minden blokk *n*-szeresen replikált
    - a replikációs faktor fájlonként konfigurálható (alapesetben *n* = *3*)
    - a replikák is a teljes klaszteren elosztva helyezkednek el
    - a replikáció célja:
      - nagyobb rendelkezésre-állás elérése
      - teljesítmény növelése
    - a replikációs stratégia kritikus mindkét cél esetében

HDFS – fájlszervezés

- **DataNode**
  - A fájlok tárolását végzi egy dolgozó csomóponton.
  - Két fájl reprezentál egy-egy blokk-replikát egy DataNode-on:
    - az adatot magát tartalmazó fájl
    - az ehhez a fájlhoz tartozó ellenőrzőösszegeket és verzió azonosítót tartalmazó fájl

HDFS – fájlszervezés

- **DataNode**
  - Egy DataNode a NameNode-hoz történő csatlakozás során egy kézfogási hitelesítést végez. Ennek során:
    - ellenőrzi a névtér azonosítóját és a HDFS verziót
    - újonnan csatlakozó DataNode-ok esetén megkapja azt a névtér azonosítót, amihez tartozni fog.

HDFS – fájlszervezés

- **DataNode**
  - Minden DataNode esetén szükséges azok regisztrációja a NameNode-on:
    - Ekkor egy tároló azonosító (storage ID) kerül hozzárendelése a DataNode-hoz, ami egy egyedi belső azonosító és nem változik.

HDFS – fájlszervezés

- **DataNode – NameNode kommunikáció:**
  - Blokk riport: a tárolt blokk-replikák azonosítására szolgál
    - Tartalma: blokk azonosító, verzió azonosító, és a blokk hossza
    - Először a regisztráció során kerül elküldésre, majd adott periodicitással (alapesetben óránként)

HDFS – fájlszervezés

- **DataNode – NameNode kommunikáció:**
  - Szívverés: az elérhetőség jelzésére szolgáló üzenet
    - Az alapértelmezett időintervallum 3 másodperc.
    - Egy DataNode kiesettnek tekintendő, ha egy adott ideig nem küld szívverés üzenetet (alapesetben 10 percig).
    - Ezek a rendszeres üzenetek egyben a skálázhatóságot is biztosítják.

HDFS – fájlszervezés

- **DataNode – NameNode kommunikáció:**
  - Egy szívverés üzenet a tárhelyről és a terheléseloszlásról tartalmaz információkat:
    - a teljes tárkapacitás
    - a felhasznált tárhely mérete
    - a folyamatban lévő írási és olvasási műveletek száma
  - A NameNode ezek alapján utasításokat küld a DataNode-nak válaszként.

HDFS – I/O műveletek és blokk kezelés

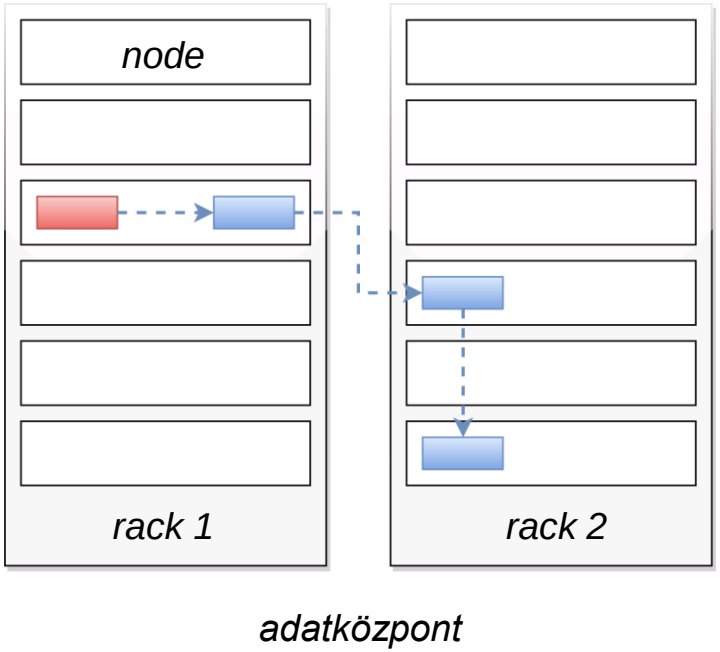
- **Fájl írás és olvasás**
  - Egy alkalmazás egy fájl létrehozásával és az abba történő írással vihet be adatot a HDFS tárba.
  - Minden fájl esetén a hozzáfűzés, olvasás és törlés műveletek engedélyezettek programozottan.

HDFS – I/O műveletek és blokk kezelés

- **Blokk kezelés**
  - A blokkok elhelyezését és a replikák kezelését a NameNode végzi a DataNode-ok által küldött blokk riportok és a szívverés üzenetek alapján.
  - Amikor egy új blokkra van szükség, a NameNode egy új blokk azonosítót hoz létre és eldönti, hogy mely DataNode-ok fogják tárolni a blokk replikáit.

HDFS – I/O műveletek és blokk kezelés

- Az adatot a DataNode-ok pipeline-jellegűen kapják meg, azaz az első kiválasztott DataNode továbbítja a csomagokat a másodiknak, majd az a harmadiknak, stb.
- Az írt adat nem áll rendelkezésre az olvasók számára, ameddig a blokk létrehozása le nem zárul minden DataNode-on.
- A folyamat végét acknowledgement üzenetekkel jelzik a DataNode-ok a NameNode felé.



HDFS – I/O műveletek és blokk kezelés

- **Replikációs stratégiák**
  - Az alapértelmezett stratégia folyamatosan biztosítja, hogy:
    - egy DataNode nem tartalmaz egynél több blokk replikát egyik blokk esetében sem;
    - legalább egy replika ugyan azon a csomóponton van, amelyik a blokkot írja;
    - Ha lehetséges, egy rack sem tartalmazza ugyan annak a blokknak kettőnél több replikáját;

HDFS – I/O műveletek és blokk kezelés

- **Replikációs stratégiák**
  - ha lehetséges, egy replika legyen ugyan azon a rack-en, amiben az a csomópont van, amelyik írja a blokkot;
  - a blokkok egyenletesen kerüljenek eloszlásra a teljes klaszteren.
- Ezeken kívül további stratégiák is definiálhatóak.

HDFS – I/O műveletek és blokk kezelés

- **Kiegyensúlyozás**
  - A fenti stratégiai célok és új DataNode-ok hozzáadása kiegyensúlyozatlanná teheti a klasztert:
    - az új blokkok egyformán kerülnek elosztásra a régi és az új DataNode-okon, de a régi blokkok megmaradnak a korábbi helyükön,
    - az újabb DataNode-okra így sokkal kisebb terhelés jut kezdetben,

HDFS – I/O műveletek és blokk kezelés

- **Kiegyensúlyozás**
  - a feldolgozás folyamán ezek a csomópontok nagyobb hálózati forgalmat generálhatnak, mivel más csomópontokról kell adatot olvasniuk.
  - A HDFS nem végez automatikus kiegyensúlyozást.
  - Megoldás: *balancer* segédprogram (a Hadoop része), amely klaszter szintű kiegyensúlyozást tud végrehajtani.

Az Apache Hadoop MapReduce

- Az **Apache Hadoop MapReduce** egy keretrendszer, ami a *MapReduce* programozási paradigmát valósítja meg.
- Jellemzői:
  - Nagy teljesítmény: terabájtos vagy nagyobb nagyságrendű adatok olcsó hardverekből épített, akár több ezer csomópontot tartalmazó klasztereken történő feldolgozására lett tervezve.
  - Hibatűrés: az egyes taszkok külön-külön is újraindíthatóak.

Az Apache Hadoop MapReduce

- Legnagyobb előnye, hogy automatikus feladatelosztást és párhuzamosítást tesz lehetővé, így egyszerű absztrakciót biztosít a fejlesztők számára.
- Minden csomópont a rajta tárolt adatokon dolgozik, együttműködve az alkalmazott fájlrendszerrel, ami gyorsabb feldolgozást tesz lehetővé.

MapReduce – a map és a reduce

- **Funkcionális programozási paradigma**
  - A programozási feladat egy függvény kiértékelése.
  - Az eredményhez vezető út nem feltétlenül ismert, a program végrehajtásához csak az eredmény pontos definíciója szükséges.
  - Pl.: Lisp, Haskell, Erlang

MapReduce – a map és a reduce

- **Funkcionális programozási paradigma**
  - A MapReduce modell a funkcionális nyelvekben gyakran használt *map* és *reduce* műveletekből merített.
  - Minden MapReduce job két fő fázisból áll:
    - *map:* a feladat szétbontása részproblémákra
    - *reduce:* a részeredményekből előállítja a feladat megoldását

MapReduce – a map és a reduce

- **Funkcionális programozási paradigma**
  - Egy tisztán funkcionális környezetben egy lista eleme, amelyet a *map* előállít nem „látja” a művelet hatását a lista többi elemén.
  - Lisp példa:

```
(map square `(1 2 3 4))
  (1 4 9 16)
(reduce + `(1 4 9 16))
  (+ 16 (+ 9 (+ 4 1) ) )
  30
```

square: unáris operátor

+ : bináris operátor

MapReduce – a map és a reduce

- **Funkcionális programozási paradigma**
  - Ha az *f* függvény alkalmazása a lista elemeire kommutatív, akkor a végrehajtás sorrendjét tetszőlegesen megváltoztathatjuk, illetve a függvény alkalmazását párhuzamosan is végrehajthatjuk.
  - A MapReduce programozási modell ezt az implicit tulajdonságot használja ki.

MapReduce – a map és a reduce

- **Funkcionális programozási paradigma**
  - Számos problémát meg lehet hatékonyan fogalmazni a *map* és a *reduce* absztrakciókat használva.
  - A problémák ilyen módon történő felosztása egyszerű feladatkiosztást tesz lehetővé a klaszteren.
  - Egyszerűbbé teszi a futási hibák kezelését.

Az Apache Hadoop MapReduce

- **A Hadoop MapReduce működése**
- A MapReduce-ban a rendszer felhasználói által elvégezni kívánt feladatok ***job***-okként jelennek meg.
- Egy job a következő elemekből áll össze:
  - a feldolgozandó bemenő adatokból,
  - egy MapReduce programból, és
  - a konfigurációs információkból.

Az Apache Hadoop MapReduce

- **A Hadoop MapReduce működése**
- A Hadoop egy job bemenő adatait rögzített méretű, független ***darab**okra* ***(input split)*** bontja.
- Minden egyes darabhoz a Hadoop létrehoz egy önálló ***map taszk**ot*, amely a felhasználó által definiált map függvényt hajtja végre a darabok ***rekord**jain*.
  - A map taszkok mindig a feldolgozandó adathoz legközelebbi node-on kerülnek végrehajtásra.

Az Apache Hadoop MapReduce

- **A Hadoop MapReduce működése**
- A map taszkok a futtató node-ok saját helyi fájlrendszerére írják a kimeneteiket.
  - Mivel a map taszkok kimenetei csak köztes kimenetek, ezért a job lefutása után már nincs szükség rájuk, így ezek elosztott, replikált tárolása felesleges lenne.
- A MapReduce a map taszkok kimeneteit ezután kulcsok szerint rendezi, majd a ***reduce taszk****okat* futtató node-ok helyi fájlrendszerére másolja.

Az Apache Hadoop MapReduce

- **A Hadoop MapReduce működése**
- A ***reduce taszkok*** hajtják végre a felhasználó által definiált reduce függvényt, és állítják elő a job végleges kimeneteit.
- Ezek kimenetei már a klaszter elosztott fájlrendszerén kerülnek tárolásra.
- A reduce taszkok számát nem az adat, hanem a feladat és a klaszter kapacitása alapján a felhasználónak kell megadni.

Az Apache Hadoop MapReduce

- **A Hadoop MapReduce működése**
- Ha egynél több reduce taszk fut majd, a map taszkok ***partícionálják*** a kimeneteiket, egy-egy saját partíciót létrehozva azoknak.
- Egy partíción belül számos kulcs is lehet, de az egy kulcshoz tartozó rekordok mind ugyan abba a partícióba kerülnek.
- A partícionáló függvényt is a felhasználónak kell definiálnia.

Az Apache Hadoop MapReduce

- **A Hadoop MapReduce működése**
- Számos esetben a klaszter sávszélessége jelenti a szűk keresztmetszet egy job futtatásakor, így fontos minimalizálni a map és reduce taszkok közötti adatátvitelt.
- Egy job-hoz megadható egy ***combine*** függvény, amely a map taszkok kimenetein fut, és amely kimenete lesz majd a reduce taszkok bemenete.

Az Apache Hadoop MapReduce

- **A Hadoop MapReduce működése**
- A ***combine*** egy speciális reduce függvényként is tekinthető, amely csak az adott node által generált kimeneteken működik.
- A combine függvény nem helyettesíti a reduce használatát, de bizonyos esetekben a reduce használható combine-ként is.

Az Apache Hadoop MapReduce

- **A Hadoop MapReduce működése**
- Mivel a combine egy optimalizációs lehetőség, ezért nem minden esetben fut le.
  - Pl.: ha nincs kellő számú rekord az adott node-on előállt kimenetben, hogy megérje futtatni.
- Mivel nem garantált, hogy a combine le fog-e futni, és ha igen, hányszor, ezért a map-ek és a combine által adott kimenetek formátumának meg kell egyeznie.

Apache Hadoop – feladatütemezés

- **A JobTracker és a TaskTracker**
  - A Hadoop Core a feladatok ütemezéséhez a mester-dolgozó szervezési mintát implementálja:
    - Egy mester csomópont: *JobTracker*

      A job-ok kezelését és ütemezését végzi.
    - Több dolgozó csomópont: *TaskTracker-ek*

      A job-ok futtatását és monitorozást végzi a dolgozó csomópontokon.

Apache Hadoop – feladatütemezés

- **JobTracker**
  - A MapReduce taszkok kiosztása a klaszter csomópontjaira.
    - Az ütemezés során minden taszk lehetőség szerint olyan csomópontra kerül, amely a lehető legközelebb van ahhoz, amelyen a feldolgozáshoz szükséges blokkok tárolása történik.
    - Ideális esetben ugyan arra a csomópontra kerül, ahol az adat tárolódik, vagy legalább ugyan arra a rack-re.

Apache Hadoop – feladatütemezés

- **JobTracker**
  - A TaskTracker-ek jelzik a JobTracker számára, ha egy kiosztott taszk hibával leáll.
  - A JobTracker ütemezési stratégiája:
    - Ha egy taszk leáll, akkor egy másik csomópontra újraütemezi azt.
    - Ha egy taszk több csomóponton is hibát ad, akkor megjelöli azt.
    - A legtöbbször hibát jelző csomópontokat listázza és nem ütemez azokra feladatokat.

Apache Hadoop – feladatütemezés

- **JobTracker**
  - Amikor befejeződik egy taszk, a JobTracker frissíti a munka státuszát.
    - A kliensek ezt a státuszt tudják lekérdezni.
  - Fontos: a JobTracker egy single-point-of-failure, azaz ennek hibája esetén a teljes MapReduce szolgáltatás leáll, és megszakad minden job végrehajtása.

Apache Hadoop – feladatütemezés

- **TaskTracker**
  - A JobTracker által kiosztott taszkok fogadását és végrehajtását végzi.
  - Minden TaskTracker egy bizonyos számú (csomópontonként konfigurálható) szabad slot-ot biztosít, ami az általa fogadható taszkok számát jelöli.

Apache Hadoop – feladatütemezés

- **TaskTracker**
  - Amikor a JobTracker egy munkát akar kiosztani, akkor először megpróbál szabad slot-okat találni azon a csomóponton, amin a feldolgozandó adat található.
  - Ha nincs ilyen csomópont, akkor az adathoz legközelebbi csomópontra ütemezi a feldolgozást, ahol szabad slot található.

Apache Hadoop – feladatütemezés

- **TaskTracker**
  - Minden egyes taszkhoz egy saját JVM folyamatot indít, ezzel biztosítva, hogy az egyes taszkok hibája esetén annak ne legyen hatása a többi futó taszkra. (Taszk izoláció.)
  - A taszkok futása során monitorozza azok működését, elkapja a kimeneteket és a kilépési kódokat.
  - Amikor egy taszk befejeződik (sikeresen vagy hibával), akkor azt jelzi a JobTracker felé.

Apache Hadoop – feladatütemezés

- **TaskTracker**
  - A TaskTracker szívverés üzeneteket is küld a JobTracker számára (alapesetben percenként), hogy jelezze az elérhetőségét és működőképességét.
  - Ezek az üzenetek tartalmazzák az aktuálisan elérhető slot-ok számát is, ami lehetővé teszi a klaszteren futó taszkok figyelését és feladatok ütemezését.

Apache Hadoop – feladatütemezés

- **Spekulatív végrehajtás**
  - Egy job végrehajtásának idejét a leglassabb taszk végrehajtásának ideje határozza meg.
    - Ennek oka lehet hardver vagy szoftver hiba is, de ez nem kerül diagnosztizálásra.
  - A MapReduce megpróbálja meghatározni, hogy melyek a lassú (lemaradó - straggler) taszkok és ezeknek egy másolatát is elindítja.

Apache Hadoop – feladatütemezés

- **Spekulatív végrehajtás**
  - Ha a lassú végrehajtás a futtatási környezet miatt állt elő, akkor a másolatként indított taszk nagy valószínűséggel hamarabb fog befejeződni, mint az eredeti, így az egész job is hamarabb tud lefutni.
  - A spekulatív végrehajtás nem növeli a megbízhatóságot, csupán a futtatás sebességének optimalizálására szolgál.

Apache Hadoop – feladatütemezés

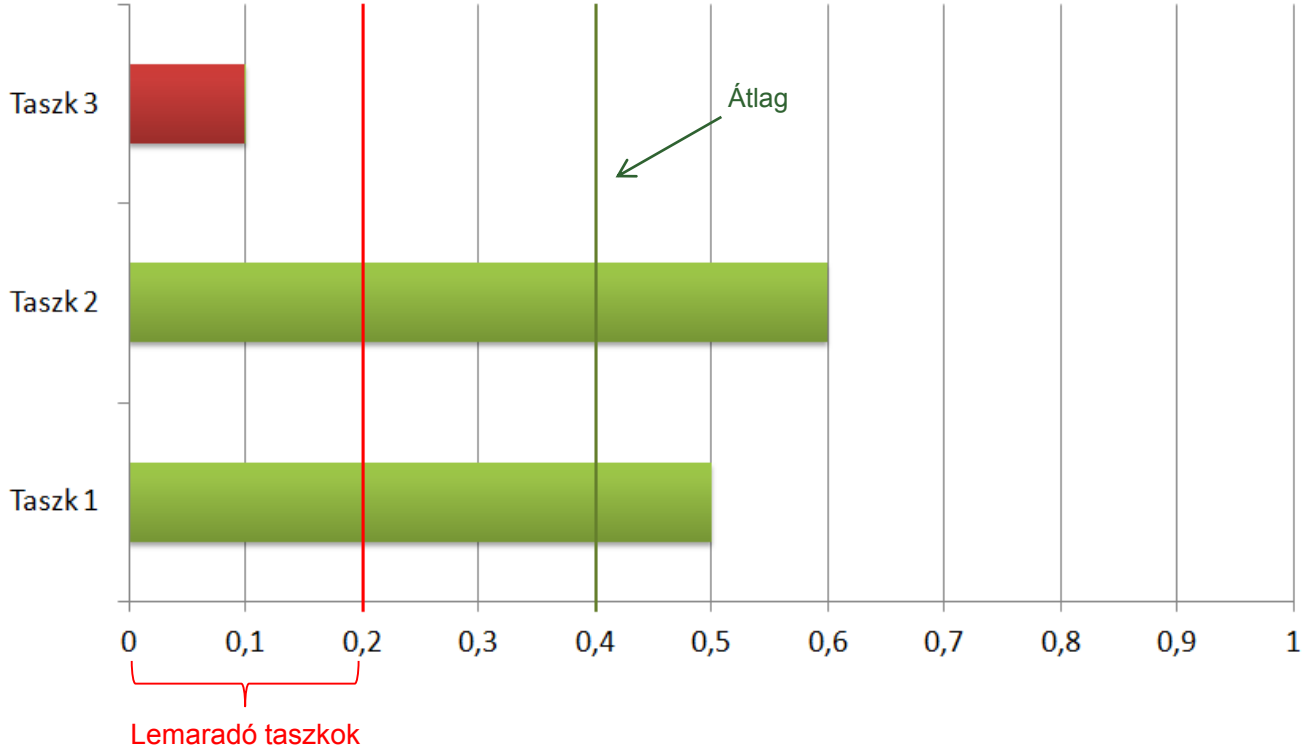
- **Spekulatív végrehajtás**
  - Minden lemaradó taszkhoz csak egy másolat indítása engedélyezett, az erőforrások hatékony kihasználása érdekében.
  - A lemaradó és a másolat taszkok közül akármelyik is fejeződik be hamarabb, annak kimenetei lesznek csak később felhasználva és a JobTracker a másikat azonnal leállítja.

Apache Hadoop – feladatütemezés

- **Lemaradó taszkok meghatározása**
  - A Hadoop minden taszk előrehaladását monitorozza, és ez alapján egy 0 és 1 közötti score értéket rendel azokhoz.
  - Ha ez a score érték kisebb, mint az átlag – 0,2, és az adott taszk legalább egy perce fut, akkor lemaradónak kell jelölni azt.

Apache Hadoop – feladatütemezés

- **Lemaradó taszkok meghatározása**



Apache Hadoop – feladatütemezés

- **Hibatűrés**
  - A MapReduce hibatűrésének egyik alapja a hibás taszkok monitorozása és újraindítása.
  - Abban az esetben, ha egy TaskTracker nem kommunikál a JobTracker-rel egy előre definiált ideig, a JobTracker feltételezi, hogy a TaskTracker elérhetetlen, pl. hardver hiba miatt leállt.

Apache Hadoop – feladatütemezés

- **Hibatűrés**
  - Ha egy job a *map* fázisban van, a JobTracker (JT) újra kiosztja az *összes* olyan taszkot, ami az elérhetetlen TaskTracker-en (TT) futott, egy másik TT-re.
  - Ha egy job a *reduce* fázisban van, akkor a JT kiosztja azokat a reduce taszkokat egy másik TT-re, amelyek folyamatban voltak az elérhetetlen TT-en.

A MapReduce adatfolyama

- Input fájlok/adatblokkok beolvasása
- Az input split-ekre bontása
- Mapping
- A köztes output kiírása a pufferbe
- Particionálás, rendezés, a puffer kiírása
- Kombinálás
- A köztes output kiírása a lemezre

Map taszk

- A partíciók darabjainak másolása
- Rendezett partíciók összeállítása
- Reducing
- A job outputjának kiírása

Reduce taszk

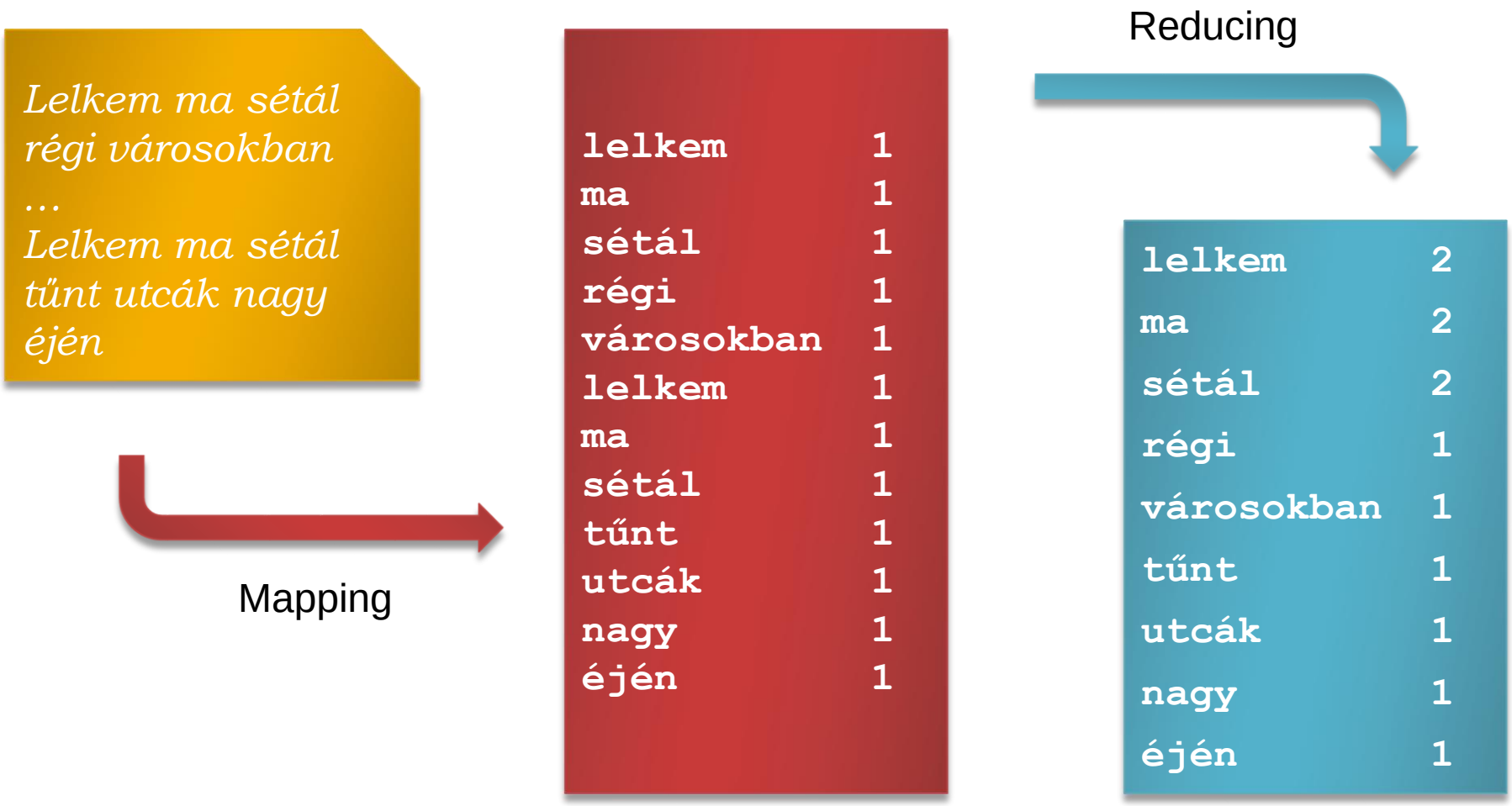
MapReduce példa – szószámláló

- **Feladat:**
  - *Számoljuk meg egy adott dokumentum-halmazban az egyes szavak előfordulásainak számát!*
  - Ez esetben a mapper megkapja a dokumentumhalmaz egy részét, majd leképezi azt <*kulcs, érték*> párokra, a függvény specifikációjának megfelelően.

MapReduce példa – szószámláló

- Ebben a példában a kimenet <*szó, "1"*> párok listája lesz.
- Fontos:
  - A mapper ez esetben sem végez aggregációt, az a reducer feladata lesz majd.
  - A map szerepe, hogy az adatot <*kulcs, érték*> párok listájára képezze.

MapReduce példa – szószámláló



MapReduce példa – Mapper osztály

```
public class MapperClass
    extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    @Override
    public void map(Object key, Text value, Context contex)
                    throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer st = new StringTokenizer(line);
        while (st.hasMoreTokens()) {
            word.set(st.nextToken());
            contex.write(word, one);
        }
    }
}
```

MapReduce példa – Reducer osztály

```
public class ReducerClass
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    private IntWritable totalSum = new IntWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context)
                      throws IOException, InterruptedException {
        int sum = 0;
        Iterator<IntWritable> it = values.iterator();
        while (it.hasNext()) {
            sum += it.next().get();
        }
        totalSum.set(sum);
        context.write(key, totalSum);
    }
}
```

MapReduce példa – Driver osztály

```
public class WordCount {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) { System.exit(-1); }
        Job job = Job.getInstance(new Configuration());
        job.setJobName("Szoszamlalo-pelda");
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(MapperClass.class);
        job.setReducerClass(ReducerClass.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setJarByClass(WordCount.class);
        job.submit();
    }
}
```

Ajánlott irodalom

- T. White: ***Hadoop: The Definitive Guide 4th ed.***, O'Reilly, 2015
- H. H. Liu: ***Hadoop Essentials: A Quantitative Approach***, CreateSpace IPP, 2012
- S. Perera: ***Instant MapReduce Patterns – Hadoop Essentials How-to***, Packt, 2013
- C. Lam: ***Hadoop in Action***, Manning, 2010