MapReduce @ Adatb haladóknak

MapReduce paradigma a CAP-tétel kontextusában

Balassi Márton
balassi.marton@gmail.com

Adatbázisok haladóknak 2012.

2012. október 30.

Miről lesz szó?

Elosztott adatfeldolgozásról általában

Motiváció, kontraszt a „hagyományossal”

Brewer-sejtés, CAP-tétel

MapReduce paradigma

Elosztott fájlrendszer (DFS)

Map és Reduce

Hadoop

Példák

Kérdések, válaszok

Elosztott adatfeldolgozás: motiváció

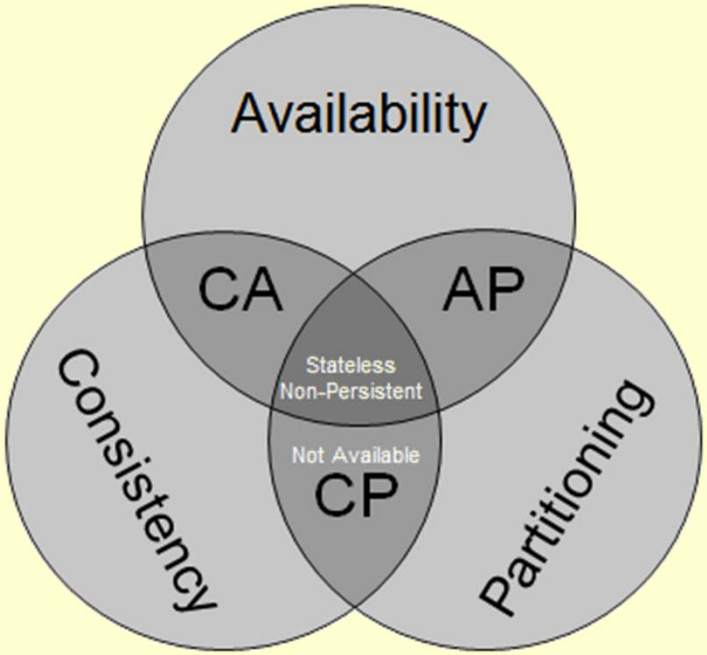
- Mi történik, ha fel kell dolgoznunk az internet adatmennyiségét?
  - Legyen $5*10^{10}$ dokumentum, átlagosan 20 Kbyte tartalommal: Pbyte nagyságrend
  - Az I/O kapacitás a legerősebb vasak esetében sem haladja meg a Gbyte/sec nagyságrendet: ~1 évig tart a beolvasás
  - Megoldás: Scale up helyett Scale out, azaz inkább több commodity vasat hasznosítunk

Más rendszerek kontextusában

- SQL (*structured* query language)
- Scale out/up
- Key/Values Pairs
- Offline batch processing
- DFS

Brewer-sejtés: elméleti határ

- Építhetünk-e egyszerre konzisztens, nagyrendelkezésre állású és partíciótűrésű rendszert?



CAP-tétel

- Az ördög a részletekben rejlik…

Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

Seth Gilbert* Nancy Lynch*

Abstract

When designing distributed web services, there are three properties that are commonly desired: consistency, availability, and partition tolerance. It is impossible to achieve all three. In this note, we prove this conjecture in the asynchronous network model, and then discuss solutions to this dilemma in the partially synchronous model.

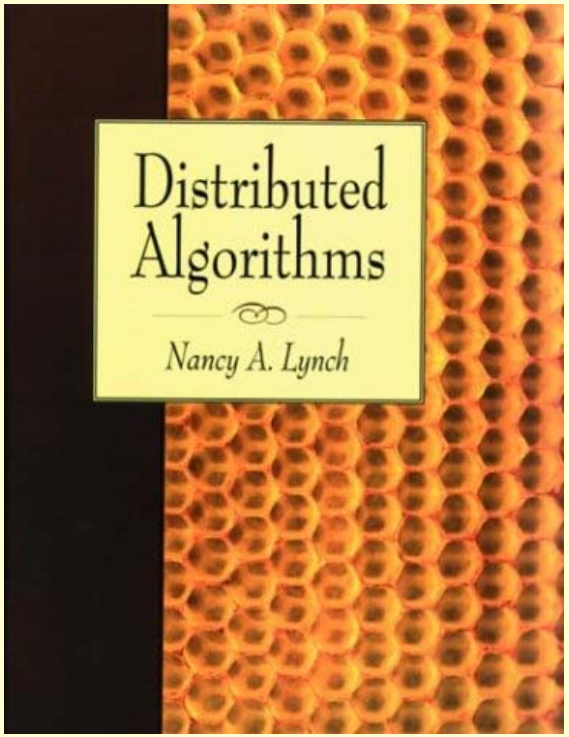
1 Introduction

At PODC 2000, Brewer[1], in an invited talk [2], made the following conjecture: it is impossible for a web service to provide the following three guarantees:

- Consistency
- Availability
- Partition-tolerance

All three of these properties are desirable – and expected – from real-world web services. In this note, we will first discuss what Brewer meant by the conjecture; next we will formalize these concepts and prove the conjecture;

*Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.
[1]Eric Brewer is a professor at the University of California, Berkeley, and the co-founder and Chief Scientist of Inktomi.



Konzisztencia (Consistency)

- Létezik egy teljes rendezés az összes rendszerbeli operáció között, amely biztosítja, hogy úgy fussanak le, mintha ezt egyetlen példányban tennék
- Azaz:
  - Megköveteljük az elosztott közös tártól, hogy úgy működjön, mintha egyetlen node-on futna, egyesével kiszolgálva az operációkat
  - Szemléletesen annyit tesz, hogyha kliens_A 1-est, majd 2-est ír egy X helyre, akkor kliens_B nem olvashatja később ezt az 1-est, mint a 2-est.

Rendelkezésre állás (Availability)

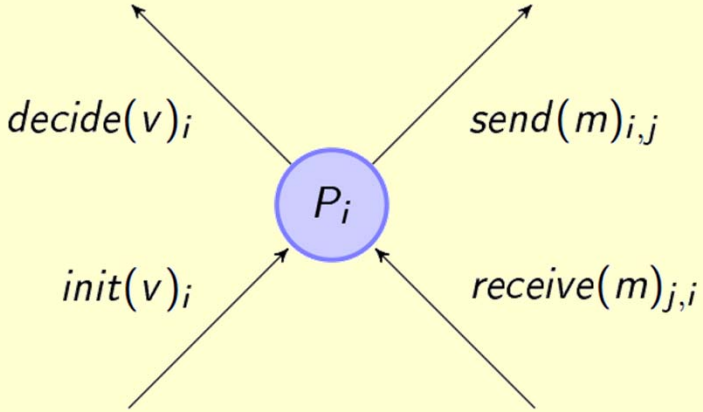
- Minden működő node-nak válaszolnia kell az általa a kliensektől kapott kérésekre.
- De a modell része, hogy a válasz megérkezése végtelenül sok időbe telhet…
- Azaz:
  - Végül minden operáció terminál.
  - A visszaadott érték helyességét viszont semmi nem garantálja.

Partíciótűrés (Partition tolerance)

- A rendszer képessége, hogy működőképes maradjon partíciók jelenlétében, akkor is, ha partíciókat törlünk vagy adunk hozzá.
- Maga a cikk „elfelejti” definiálni.
- De mi is az a „partíció”?
  - Tekintjük a node-ok diszjunkt halmazokba, azaz „komponensekbe” való felosztását.
  - Azt mondjuk, hogy különböző komponensek közötti csomagok elvesznek.
  - Gyakorlatilag ez pontosan annyi, hogy csomagok a node-ok között „ elveszhetnek”.

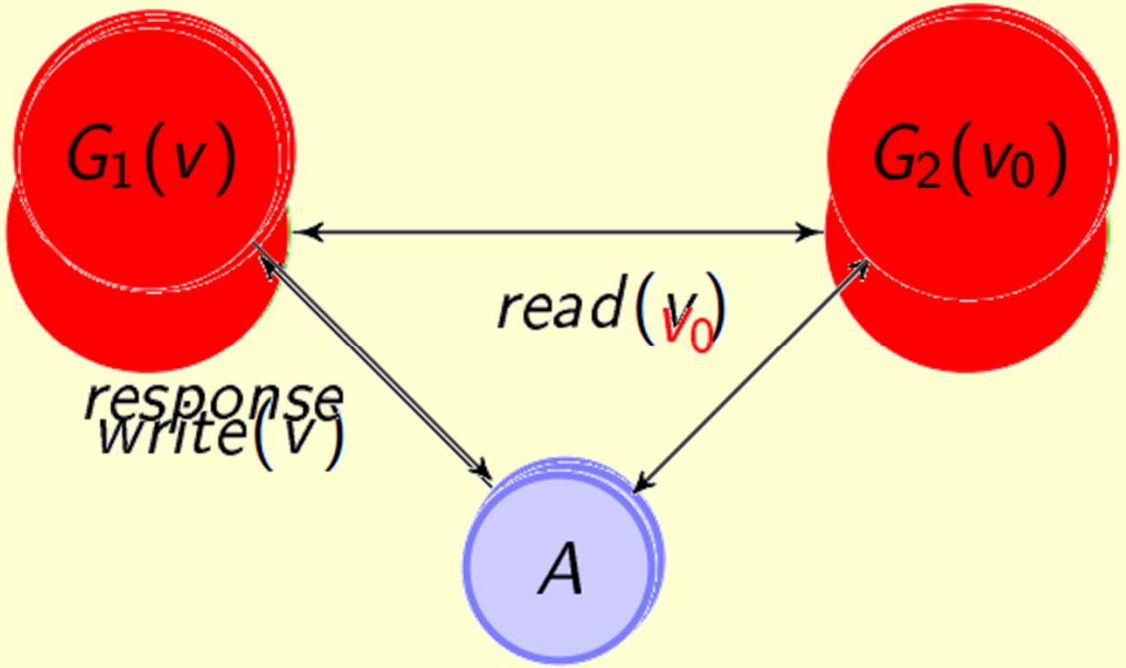
Lynch aszinkron modellje

- I/O auomata
- Fairness
- Kompozíció



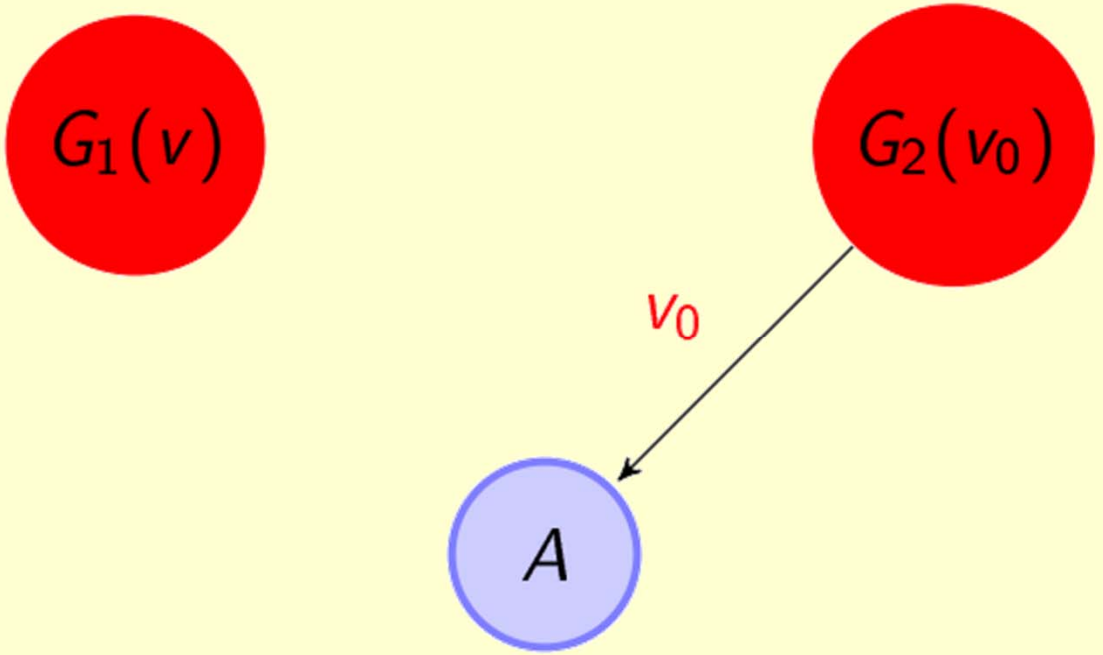
CAP-tétel és bizonyítása

- Az aszinkron modellben nem konstruálható olyan read/write adatobjektum, amely garantálja a rendelkezésre állás és a konzisztencia feltételeit minden fair esetben.



CAP-tétel és bizonyítása

- Az aszinkron modellben nem konstruálható olyan read/write adatobjektum, amely garantálja a rendelkezésre állás és a konzisztencia feltételeit minden fair esetben.



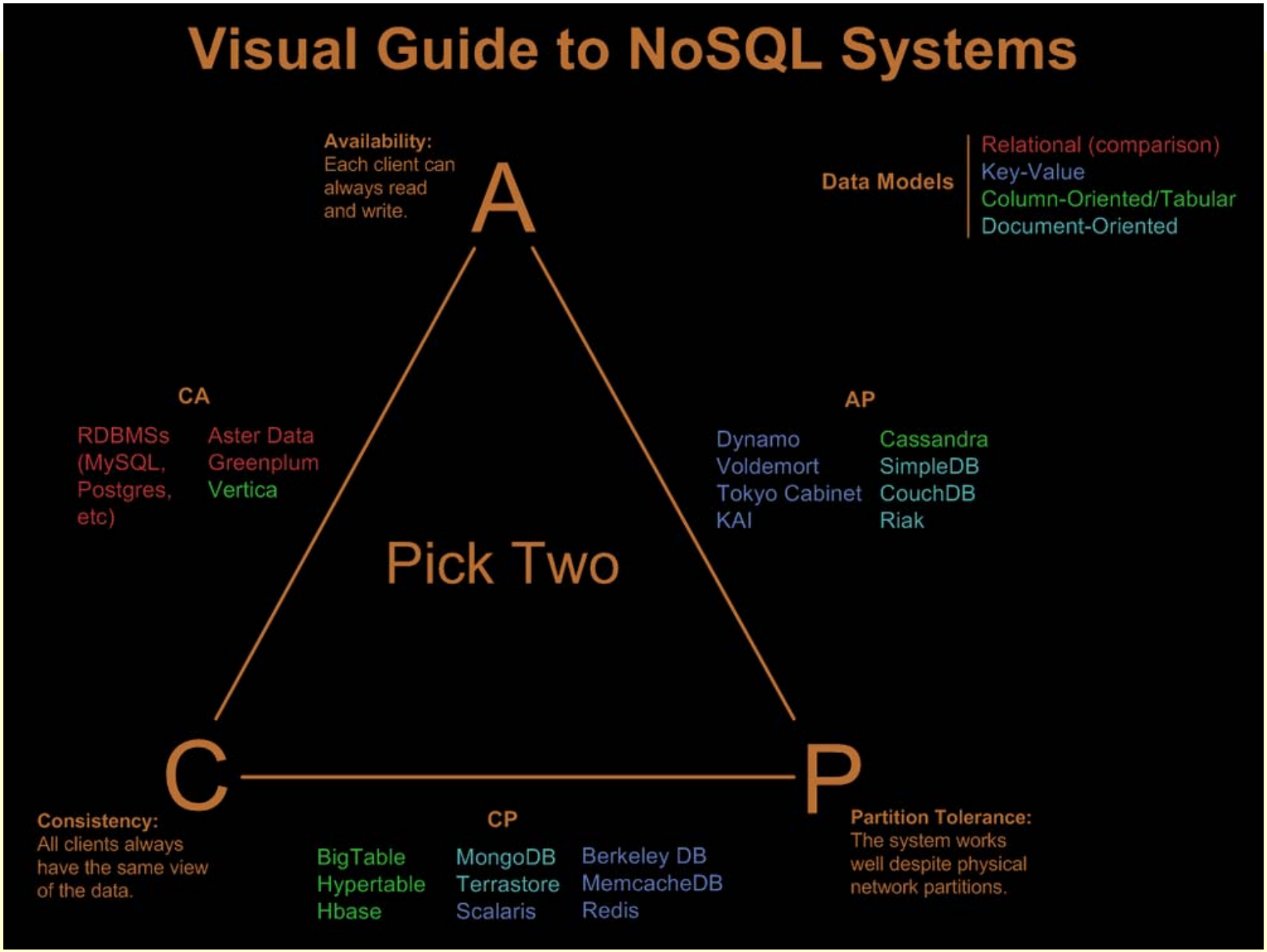
CAP-tétel és bizonyítása

- Az aszinkron modellben nem konstruálható olyan read/write adatobjektum, amely garantálja a rendelkezésre állás és a konzisztencia feltételeit minden fair esetben.

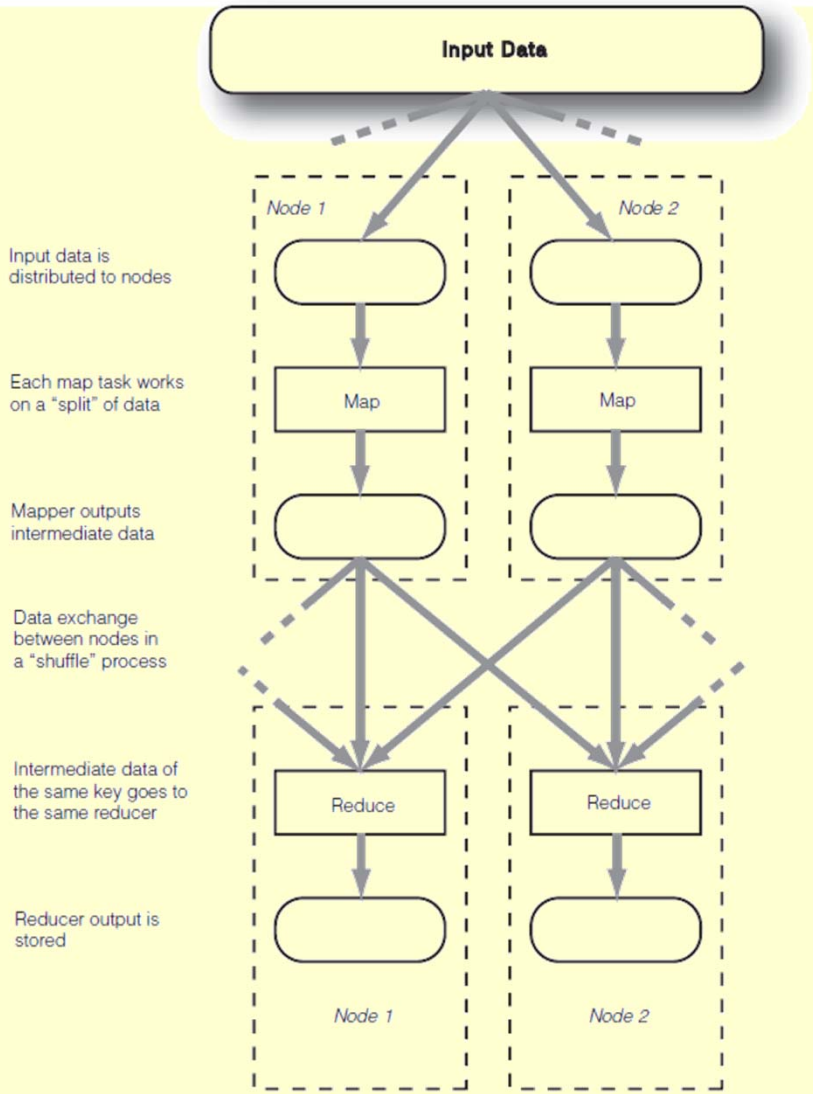
Leegyszerűsítve

$$P \Rightarrow \neg(A \wedge C)$$

CAP-tétel és NoSQL eszközök



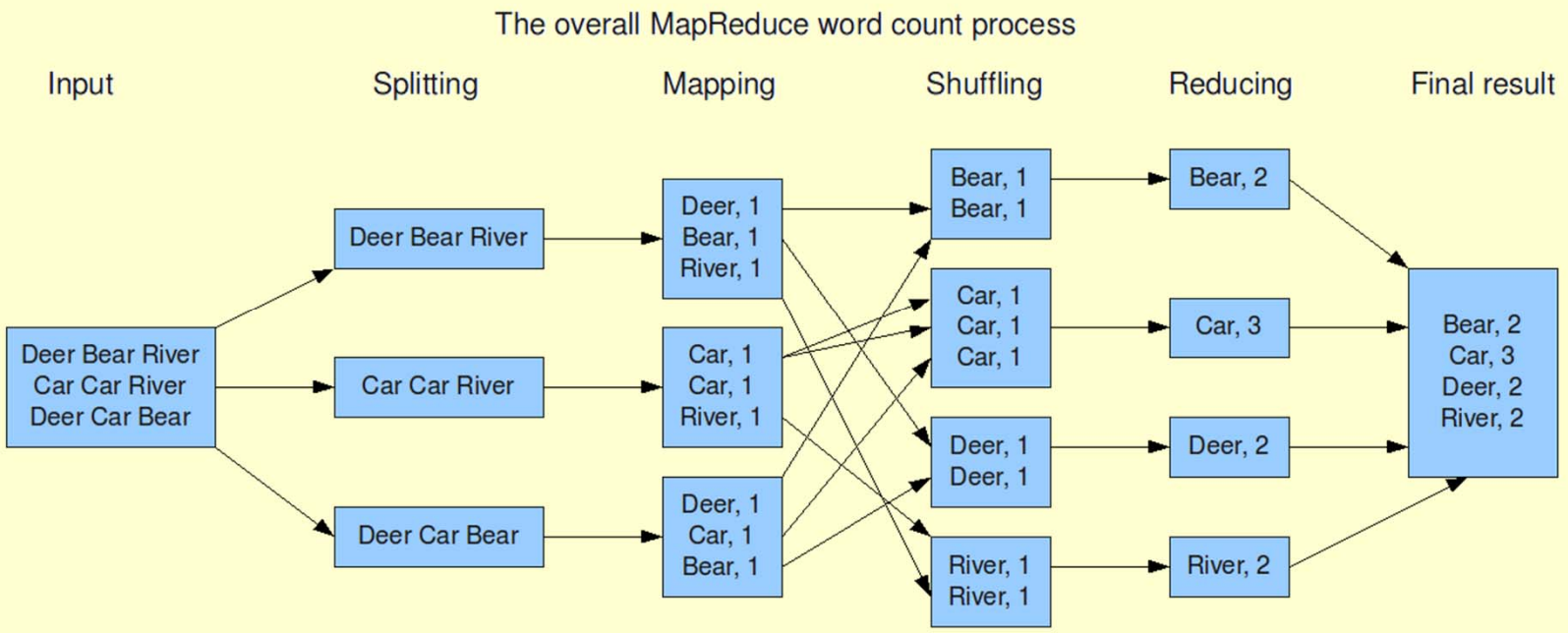
MapReduce paradigma



WordCount példa

	Input	Output
map	<k1, v1>	list(<k2, v2>)
reduce	<k2, list(v2)>	list(<k3, v3>)

WordCount példa



WordCount példa (Hadoop)

```
public static class MapClass extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {

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

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}
```

WordCount példa (Hadoop)

```
public static class Reduce extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

MeanNaive példa

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

MeanCombine példa

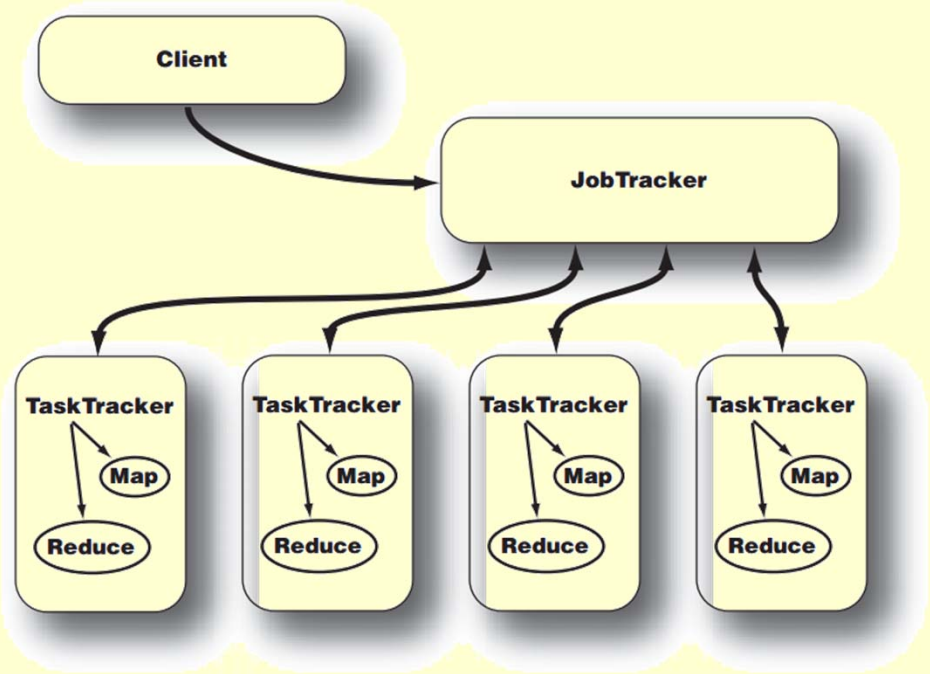
```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, pair (r, 1))

1: class Combiner
2:     method Combine(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         Emit(string t, pair (sum, cnt))

1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Hadoop folyamatok

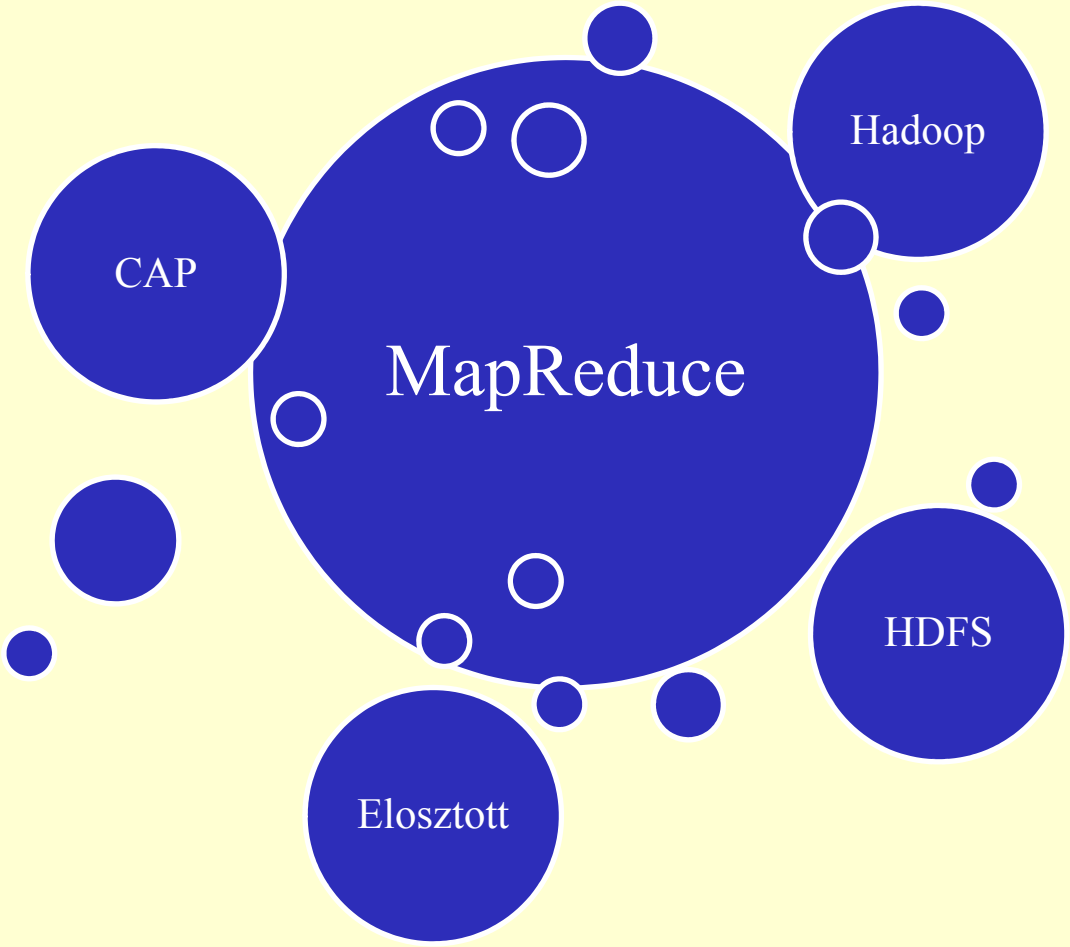
- NameNode
- (Secondary NameNode)
- DataNode
- JobTracker
- TaskTracker



A gyakorlatban…

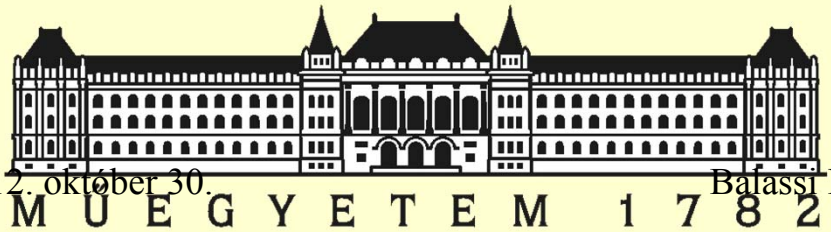
- A bit of hands on Hadoop
- WordCount
- HDFS kezelés
- Webes UI
- Szimulációk kezelése

Összefoglalás helyett



Köszönöm a figyelmet!

Balassi Márton
marton@db.bme.hu



Adatbázisok haladóknak 2012.

2012. október 30.