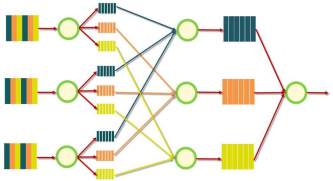


Map-Reduce and Adwords Problem

Miłosz Kadziński

Institute of Computing Science
Poznan University of Technology, Poland

www.cs.put.poznan.pl/mkadzinski/wpi

[1] Wykład będzie dotyczył dwóch niepowiązanych ze sobą zagadnień. W pierwszej, dłuższej części skupimy się na bardzo specyficznym sposobie przetwarzanie danych o dużych rozmiarach. Wyróżnia się w nim dwie fazy: map, która służy do systematycznej analizy dla mniejszych wolumenów danych oraz Reduce, dedykowana do agregacji częściowych wyników takiej analizy i udzielenia całościowej odpowiedzi dla rozważanego problemu. Przedstawimy motywację dla przetwarzania danych w ten sposób, powiemy do czego MapReduce się nadaje i kiedy powinno się go używać. Ta część wykładu będzie miała też charakter małej powtórki, bo przejdziemy przez bardzo wiele zagadnień dotyczących wyszukiwania i przetwarzania informacji, które były poruszane w poprzednich tygodniach. W drugiej części zarysujemy charakterystykę problemu Adwords. Omówimy przykładowe algorytmy, które służą do wyboru reklamodawców, których reklamy zostaną wyświetlone dla danej sekwencji zapytań.

Big Data (1)

- Big data is **a fact of the world** (real-world systems must grapple with it)
- More data translates into **more effective algorithms**

- **Google** was processing 100TB of *data* a day in 2004 and 20 PB a day in 2008
- **eBay**'s warehouses in 2009 contained about 10 PB of *user data* in 2009

- **Large Hadron Collider** in Geneve produces severeal PB of data a year
- **European Bioinformatics Insitute** hosts a repository of sequence data (PB)
- **Synoptic Survey Telescope** in Chile produces 0.5PB of images each month

BIG DATA

- Gathering, analyzing, monitoring, filtering, searching and organizing **web content**
- Website operators record **user activity** (what users look at or click on, how much time they spend)

Bill Gates (Seattle, 1981)
"640K ought to be enough for anybody"

[2] Żyjemy w erze dużych danych. Akceptacja tej sytuacji wymaga rozwoju dedykowanych systemów, które byłyby w stanie zmierzyć się z przetwarzaniem dużych wolumenów. Potrzebę ich rozwoju wyrażają różne grupy. Po pierwsze są to największe światowe koncerny takie jak Google czy eBay, które w erze dużych danych żyją od kilkunastu lat, a zdolność ich przetwarzania zdecydowała o sukcesie, który te firmy odniosły. Po drugie, świat nauki wymaga coraz efektywniejszych algorytmów przetwarzające dane o dużych rozmiarach. Najlepszym przykładem jest tu Wielki Zderzacz Hadronów, który umożliwia realizację badań, służących lepszemu poznaniu cząstek elementarnych. Można jednak dodać przykłady Europejskiego Instytutu Bioinformatyki albo teleskopów badawczych, żeby uświadomić sobie, że repozytoria z sekwencjami DNA albo dane obrazowe też mają ogromne rozmiary i wymagają dedykowanego przetwarzania. Po trzecie, my jako zwykli użytkownicy albo dostawcy usług mniejszej skali też chcielibyśmy mieć możliwość przetwarzania dużych danych. Potrzeba taka występuje ilekroć chcemy przetworzyć choć małą część zawartości sieci lub przeanalizować aktywność użytkowników utrzymywanego przez nas serwisu. Dziś, a właściwie to od bardzo bardzo wielu lat, pewne jest, że słynne słowa Billa Gatesa z 1981r. dotyczące wystarczalności 640K dla każdego użytkownika, nie okazały się prorocze.

Big Data (2)

- Processing large amount of text by NLP researchers and IR community
- Three components: data, representation of data and algorithms
- **Data matters the most** (why not sophistacated algorithms and deep features applied to lots of data?)

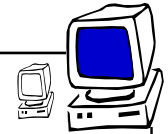
Why large data? "*Because they're there"*

Collect more data to have better algorithms for solving real-world problems

- Correct use of "than" and "then" in English (more data lead to better accuracy)
- Answering short, fact-based quetions: "Who shot Abraham Lincoln?" Answer: John Booth using pattern-matching instead of returning a list of docs
- Language model: probability distribution characterizing the likelihood of observing a particular sequence of words (speech recognition and machine translation)

[3] Coraz więcej osób żyje w przekonaniu, że większa ilość danych prowadzi do większej efektywności algorytmów. Co więcej, przez wielu dane są postrzegane jako istotniejsze niż algorytmy. Te ostatnie powinny być proste, a wysiłek powinien być nastawiony na zbieranie danych. Jest to szczególnie ewidentne w dziedzinie przetwarzania języka naturalnego oraz wyszukiwania i przetwarzania informacji. Podajmy trzy proste przykłady, które wpisuje się w tę tezę. Pierwszy dotyczy sprawdzenia poprawności językowej tekstu (jak dla porównania than vs. then). Drugi wiąże się z coraz popularniejszym zagadnieniem udzielania bezpośredniej odpowiedzi na zapytanie użytkownika zamiast wskazania dokumentów, gdzie mógłby on sobie tę odpowiedź znaleźć sam. Trzeci dotyczy modelowania języka, gdzie kluczowa jest estymacja prawdopodobieństw występowania poszczególnych sekwencji słów. Jest jasne, że zgromadzenie większej liczby rzeczywistych przykładów w każdym z powyższych przypadków pozwala na lepsze działanie algorytmu, który takie dane by eksploatował.

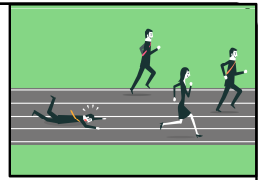
Big Ideas Behind Processing Big Data (1)



Scale "out", not "up"

- A large number of **commodity low-end servers** is **preferable**
- Purchasing symmetric multi-processing machines with a large number of processors and a large amount of shared memory is not cost effective
- The low-end server market overlaps with the high-volume desktop market

Assume failures are common



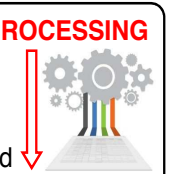
- At warehouse scale, failures are commonplace
- Example: 10,000-server cluster with a mean-time between failures of 1000 days for each server, would experience roughly 10 failures a day
- The need for coping with failures without impacting the quality of service (various mechanisms, e.g., automatic restarts on different cluster nodes)

[4] Na kolejnych trzech slajdach omówimy podstawowe założenia wiążące się z przetwarzaniem dużych danych, którym hołdować będziemy podczas tego wykładu. Po pierwsze, nie chcielibyśmy używać drogiego sprzętu, tj. dedykowanych serwerów obliczeniowych z ogromną liczbą procesorów i wielką pamięcią. Dużo bardziej opłacalne jest skorzystanie ze sprzętu, którym dysponujemy w domu, w biurze, na uczelni, itd. Może jest on tani, kiepskiej jakości, ale jeśli taki sprzęt, być może w większej liczbie, jest dostępny, to po prostu z niego korzystajmy. Po drugie, musimy pogodzić się z tym, że błędy w przetwarzaniu danych są nieuniknione i powszechne. Nawet gdy korzystalibyśmy z wielkiego, 10000-maszynowego klastra obliczeniowego, w ramach którego każdy serwer psułby się raz na trzy lata, to okazałoby się, że musimy obsługiwać ok. 10 błędów dziennie. Wniosek jest taki, że z błędami tak czy siak trzeba sobie radzić.

Big Ideas Behind Processing Big Data (2)

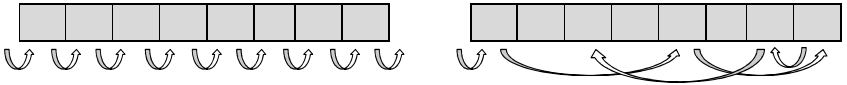
Move processing to the data

- In high-performance computing, ”processing" and ”storage” nodes linked together by a high-capacity interconnect
- **Efficient to move the processing around** rather than data
- Desired architecture: processors and storage (disk) are co-located (run code on the processor attached to the block of data we need)



Process data sequentially and avoid random access

- Big datasets are too large to fit in memory and must be held on disk
- **Avoid random data access, and instead process data sequentially**
- *Example*: a database containing 10^{10} 100-byte records: updating 1% of them will take about a month; reading and rewriting all records - under a work day

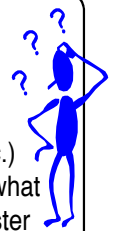


[5] Po trzecie, do niedawna pokutowało przeświadczenie, że w systemach dużej skali lepiej rozdzielać jest wierzchołki przetwarzające i przechowujące dane. Dziś większość osób zgadza się co do tego, że efektywniej jest przetwarzać dane tam, gdzie są przechowywane, co w praktyce oznacza, że kod powinien być uruchomiony na procesorze powiązanym z blokiem danych, które chcemy przeanalizować. Kolokwialne mówi się o przesunięciu przetwarzania do danych, a nie danych do przetwarzania. Po czwarte, ważne jest, by dane przetwarzać w sposób sekwencyjny, unikając losowego dostępu. Ogromne wolumeny danych nie mieszczą się w pamięci i muszą być przechowywane na dysku. Najlepszy przykład, który uzmysławia szybkość przetwarzania sekwencyjnego dotyczy bazy danych z 10 miliardami rekordów. Jeśli chcieć by uaktualnić tylko 1% z nich, to dokonując dostępu tylko do nich mogłoby to zająć miesiąc. Jeśli zaś przepisać by całą bazę od nowa, już uaktualnioną, to można by to zrobić w ciągu dnia, a nie miesiąca.

Big Ideas Behind Processing Big Data (3)

Hide system-level details from the application developer

- Distributed software: manage several threads, processes, or machines (locking of data, data starvation, etc.)
- Code runs concurrently in unpredictable orders, accessing data in unpredictable patterns (race conditions, deadlocks, etc.)
- **Solution**: isolate the developer from system-level details, and separate what computations are to be performed and how they are carried out on a cluster

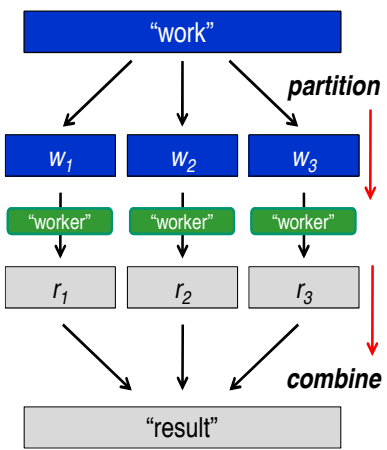


Seamless scalability

- For data-intensive processing, **scalable algorithms are highly desirable**
- **Complex tasks** often cannot be allocated in a linear fashion (”nine women cannot have a baby in one month”), but sometimes they **can be chopped**
- **Solution**: algorithm remains fixed and it is the responsibility of the execution framework to execute the algorithm while approaching the ideal scaling

[6] Po piąte, szczegóły systemowe powinny być ukryte przed użytkownikami. Rozproszenie obliczeń wymaga wiedzy o wątkach, procesach, blokadach, głodzeniu, zakleszczeniach, itd. Jest to wiedza zaawansowana i świetnie by było, gdyby nie była potrzebna użytkownikowi, który ma dane do przetworzenia, a obsługa rozproszenia obliczeń była zagwarantowana przez framework obliczeniowy. Wreszcie, to środowisko, w którym algorytm jest wykonywany, powinno zapewnić skalowalność obliczeń. W wielu zastosowaniach taka skalowalność może być osiągnięta poprzez odpowiedni podział danych do przetworzenia przez poszczególne jednostki obliczeniowe.

Divide and Conquer



How to ...

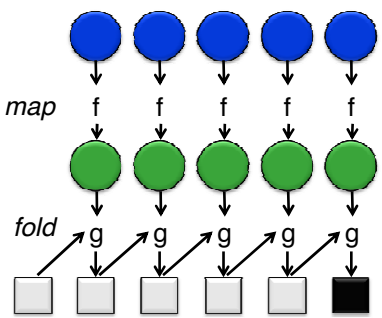
- break up a large problem into smaller tasks?
- assign tasks to workers?
- ensure that the workers get the data they need?
- coordinate synchronization among workers?
- share partial results?
- deal with software errors and hardware faults?

[7] Podstawowa zasada, który kryje się za omawianym sposobem przetwarzania danych to dziel i rządź/zwyciężaj. Złożony problem dzieli się na mniejsze podproblemy tego samego typu tak długo aż staną się wystarczająco proste do bezpośredniego rozwiązania. Z kolei rozwiązania tych podproblemów scala się, uzyskując całkowite rozwiązanie problemu. Takie podejście do problemu wymaga udzielenia odpowiedzi na wiele pytań. Jak podzielić problem na mniejsze zadania? Jak te zadania przydzielić do jednostek przetwarzających? Jak zapewnić im dostęp do danych? Jak skoordynować wykonanie algorytmów dla podproblemów i udostępnić ich wyniki? Jak obsłużyć błędy, które mogą pojawić się na różnych etapach?

Map-Fold: Origins of Map-Reduce

Functional programming

- Higher-order functions that can accept other functions as arguments
- Two common built-in higher order functions are *map* and *fold* (**Lisp**)



map (transformation of a dataset)

- takes as an argument a function *f* (that takes a single argument)
- applies *f* to all elements in a list

fold (aggregation operator)

- takes as arguments a function *g* (taking two args) and an initial value
- g is applied to the initial value and the first item in the list; then g is applied to the intermediate result and the next item, etc.

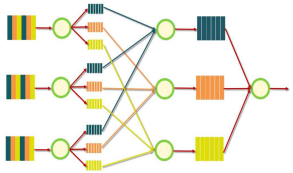
can be parallelized

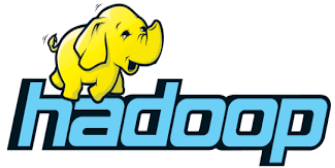
```
MAP([1,2,4,10], function(x) {return x*x;}) > [1,4,16,100]
FOLD([1,4,16,100], 0, function(x, y) {return x+y;}) > 121
```


[8] MapReduce wywodzi się z programowania funkcyjnego, gdzie wykorzystywane są funkcje wyższego poziomu, akceptujące inne funkcje jako argumenty. Przykładowo, w języku Lips wykorzystywano funkcje map i fold. Zadaniem tej pierwszej jest transformacja zbioru danych. Przyjmuje ona jako argument tablicę elementów do przetworzenia oraz jednoargumentową funkcję. Wykonanie map polega na zastosowaniu tej funkcji to każdego elementu z listy. Przykładowo (patrz dół slajdu) dla tablicy 4-elementowej i funkcji podnoszącej argument do kwadratu, wynikiem działania map będzie tablica, gdzie każdy z wejściowych argumentów jest podniesiony do drugiej potęgi. Z kolei funkcja fold odpowiada za agregację. Przyjmuje ona trzy argumenty: tablicę elementów do agregacji, element początkowy oraz funkcję dwuargumentową. To ostatnia stosowana jest najpierw do elementu początkowego i pierwszego elementu z listy, następnie do wyniku częściowego i drugiego elementu z listy, potem do wyniku częściowego i trzeciego elementu z listy, itd. Przykładowo, dla tablicy zwróconej przez map, zerowego elementu początkowego i funkcji agregacji sumującej dwa elementy, najpierw 0 zostanie dodane do 1, potem 1 do 4, potem 5 do 16, potem 21 do 100, a ostatecznym wynikiem będzie 121. Kluczowa obserwacja w kontekście MapReduce jest taka, że zarówno operacje map, jak i fold, można zrównoleglić ze względu na ich specyfikę: niezależną transformację elementów zbioru danych oraz agregację wyników częściowych.

What is Map-Reduce?

- **Programming model**, i.e., a generic ”recipe" for processing large datasets that consists of two stages: a user-specified computations applied over all input records in a dataset and aggregating the intermediate results
- **Execution framework** coordinating the map and reduce phases of processing for large-scale data on clusters of commodity machines
- **Software implementation** of the programming model and the execution framework, e.g., Google's proprietary implementation vs. the open-source Hadoop implementation in Java



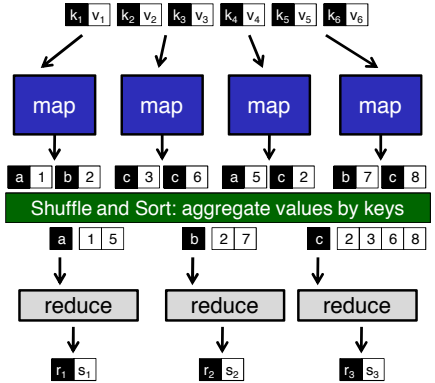


[9] Hasło MapReduce można rozumieć na różne sposoby. Po pierwsze jest to model programowania, który oferuje receptę na dwufazowy sposób przetwarzania dużych danych. Po drugie to środowisko, które koordynuje sposób wykonania faz map i reduce na klastrze obliczeniowym. Po trzecie to konkretna implementacja modeli i środowiska. Oryginalna platforma oparta na tym pomyśle została opracowana przez Google, a jedną ze słynniejszych propozycji open-sourcowych był Hadoop.

Simplified View of Map-Reduce

Map: $(k_1, v_1) \rightarrow [(k_2, v_2)]$

- Applied to every input key-value to generate an arbitrary number of intermediate key-value pairs



- Distributed "group by" operation on intermediate keys
- intermediate data arrive at each reducer in order, sorted by the key

Reduce: $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$

- Applied to all values associated with the same intermediate key to generate output key-value pairs

[10] Sposób przetwarzania oferowany przez MapReduce opiera się na pracy z parami (klucz, wartość). Rozpoczniemy od omówienia uproszczonego podejścia MapReduce. Zadaniem Mapperów jest realizacja funkcji map, która przetwarza każdą parę (klucz wejściowy, wartość wejściowa), generując określoną liczbę kluczy i wartości pośrednich. Takich Mapperów jest wiele i każdy z nich odpowiada ze przetworzenie określonej puli kluczy i wartości wejściowych. Pomiędzy fazami map i reduce następuje operacja grupowania wyników częściowych po kluczach częściowych po to, by Reducer realizujący funkcję reduce miał gwarancję, że wszystkie wyniki pośrednie skojarzone z tym są kluczem pośrednim do niego dotrą. Funkcja reduce jest więc stosowana do wszystkich wartości skojarzonych z tym samym kluczem pośrednim, prowadząc do powstania wyniku, czyli pary (klucz wyjściowy, wartość wyjściowa). Dla wygody podczas wykładu będzie mówili o Mapperze, Reducerze, Combinerze czy Partitionerze, nie szukając na siły ich bardziej opisowych polskich odpowiedników.

Map-Reduce: “Hello World!"

Word count

- Counts the number of occurrences of every word in a text collection
- May be the first step in building a unigram language model (i.e., probability distribution over words in a collection)

```
1: class Mapper
2:     method Map(docid a; doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t; count 1)

1: class Reducer
2:     method Reduce(term t; counts [c1; c2; ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1; c2; ...] do
5:             sum ← sum + c
6:         Emit(term t; count sum)
```


[11] Omówienie wybranych przykładów zaczniemy od swoistego Hello world! dla MapReduce. Jest nim realizacja zadania zliczania liczby wystąpień poszczególnych słów w kolekcji dokumentów, czyli word count. Jak wiecie z poprzednich wykładów, jest to przydatne zarówno do tworzenia reprezentacji dokumentów w wybranej przestrzeni, jak i modelowania języka, gdzie konieczne jest np. wykorzystanie prawdopodobieństw wystąpień poszczególnych słów w danej kolekcji. Zadaniem tego slajdu jest też przedstawienie pseudokodu, który będziemy wykorzystywać w czasie tego wykładu. W klasie Mappera i Reduce mamy tu tylko po jednej funkcji, odpowiednio Map i Reduce. Map przetwarza dokument, który ma identyfikator (klucz) i wartość (zawartość dokumentu). Zadaniem funkcji jest przeglądanie zawartości dokumentu i dla każdego napotkanego termu interpretowanego jako klucz pośredni emisja jedynki jako wartości pośredniej. Z kolei funkcja Reduce ma gwarancję, że dla danego klucza pośredniego dotrą do niego wszystkie wartości z nim skojarzone. Dla naszego przykładu będą to jedynki wyemitowane ilekroć dany term został napotkany w jakimś dokumencie przez jakiegokolwiek Mappera. Zadaniem Mappera jest ich dodanie oraz emisja pary wyjściowej, a więc termu, dla którego zliczona jest liczba jego wystąpień we wszystkich dokumentach. Warto zwrócić uwagę, że zadanie to świetnie nadaje się do realizacji w duchu MapReduce, bo przecież dokumenty można przetwarzać od siebie niezależnie, co stwarza pole do działania dla różnych Mapperów, a wyników końcowych też jest dużo, bo przecież liczba wystąpień każdego termu jest osobnym wynikiem, co z kolei daje pole do działania dla różnych Reducerów.

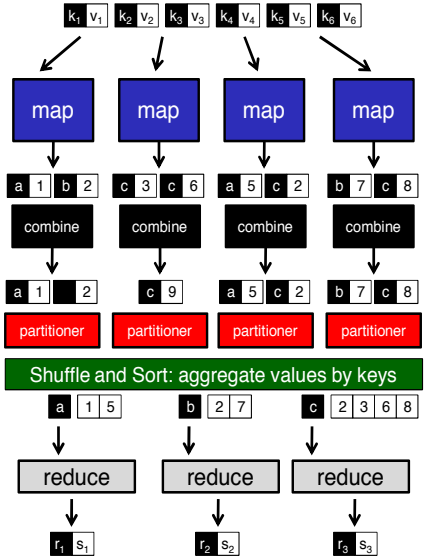
Big Ideas Behind Map-Reduce

- MapReduce is **separating** the **what** of distributed processing from the **how**
- MapReduce program, called **job**, consists of **code** for mappers and reducers (as well as combiners and partitioners) packaged together with **configuration parameters** (e.g., where the input lies and where the output should be stored)
- The developer submits the job to the submission node of a cluster (in Hadoop - jobtracker) and **execution framework** ("runtime") **takes care of everything else**

- **Scheduling**: maintain a sort of a task queue and track the progress of running tasks
- **Data/code co-location**: feed data to the code (sometimes stream data over the network)
- **Synchronization**: mechanisms by which multiple concurrently running processes join up – accomplished by a barrier between the map and reduce phases
- **Error and fault handling**: all the tasks above in an environment where errors and faults are the norm (hardware, software, data)

[12] Wróćmy do pomysłów przedstawionych na początkowych slajdach oraz pytań zadanych przy okazji omawiania podejścia dziel i rządź. MapReduce oddziela to, co ma zostać zrobione od tego, w jaki sposób ma to być zrealizowane. Program MapReduce, nazywany jobem, składa się z kodów Mappera i Reducera (oraz innych obiektów, które poznamy wkrótce) spakowanych razem z parametrami konfiguracyjnymi, które mówią np. gdzie znajdują się dane do przetworzenia, a gdzie mają być zapisane wyniki. Zadaniem programisty jest dostarczenia takiego programu do określonego wierzchołka w klastrze obliczeniowym i na tym jego rola się kończy. Realizacja rozproszonego przetwarzania danych jest koordynowana w całości przez framework MapReduce. W szczególności zapewnia on podział i szeregowanie zadań, dostępność danych, śledzenie i synchronizację wykonania zadań oraz obsługę błędów wszystkich rodzajów.

Map-Reduce: Processing (Key,Value) Pairs



Combiners allow for **local aggregation** before the shuffle and sort phase

- Mini-reducers on the output of the mappers
- Operate in isolation and do not have access to intermediate output from other mappers
- Reduce network traffic

Partitioners divide the intermediate key space and assign intermediate pairs to reducers

- Basic idea: compute the hash value of the key and take the mod with the number of reducers
- The division can be non-balanced in terms of work, but is balanced in terms of keys

[13] Mapper i Reducer reprezentują dwie kluczowe klasy. Wprowadźmy teraz dwie dodatkowe klasy, odgrywające istotną rolę w polityce zarządzania parami kluczy i wartości pośrednich. Są nimi Combiner i Partitioner. Zadaniem tego pierwszego jest realizacja lokalnej agregacji na wyjściu każdego Mappera. Co istotne, Combinery działają w izolacji i nie mają dostępu do wyników działania innych Mappera. Niemniej jednak ich działaniu pozwala na znaczące ograniczenie ruch z powodu redukcji liczby par kluczy i wartości pośrednich. Z kolei Partitioner dokonuje podziału przestrzeni kluczy pośrednich i ich przypisanie do poszczególnych Reducerów. Standardowy sposób tego rozdziału polega na wykorzystaniu funkcji haszującej, co pozwala na m.w. równy podział pod względem kluczy pośrednich, które przetwarza każdy Reducer. Nie wiemy jednak, jak jest ilość pracy skojarzona z każdym kluczem pośrednim, więc ostatecznie to zrównoważenie pracy między reducerami może nie być idealne.

Local Aggregation

```
1: class Mapper
2: method Map(docid a; doc d)
3:     H ← new AssociativeArray
4:     for all term t ∈ doc d do
5:         H{t} ← H{t} + 1
6:     for all term t ∈ H do
7:         Emit(term t; count H{t})
```

```
1: class Mapper
2: method Initialize
3:     H ← new AssociativeArray
4: method Map(docid a; doc d)
5:     for all term t ∈ doc d do
6:         H{t} ← H{t} + 1
7: method Close
8:     for all term t ∈ H do
9:         Emit(term t; count H{t})
```

- An associative array (Map in Java) is introduced to tally up term counts within a single doc
- Instead of emitting a key-value pair for each term in the doc, emit a key-value pair for each unique term in the document

- Preserve state across multiple calls of the Map method; continue to accumulate partial term counts in the associative array across multiple docs
- Combiner's functionality directly inside the mapper (in-mapper combining)
- Control over when local aggregation occurs; more efficient than using actual combiners
- State preserved across multiple input instances; memory to store intermediate results

[14] W duchu agregacji lokalnej wspomnianej przy okazji Combinera, wróćmy do przykładu ze zliczaniem słów. Omówiony na slajdzie przykład uzmysławia, że mimo, iż MapReducer bardzo wiele za nas zrobi, to wciąż jako programiści mamy spore pole do popisy i musimy podjąć istotne decyzje co do sposobu przetwarzania danych. Przedstawiono dwie wersje pseudokodu, w którym agregacja lokalna realizowana jest w Mapperze z wykorzystaniem tablicy asocjacyjnej. W wariancie po lewej stronie tablica taka jest tworzona wewnątrz funkcji map, a więc dla każdego dokumentu z osobna. Zamiast emisji jedynek dla każdego napotkanego termu, emitujemy zagregowaną liczbę jego wystąpień w dokumencie. Po prawej stronie, taka tablica inicjalizowana jest raz dla danego Mappera. Agreguje więc ona wyniki lokalne ze wszystkich dokumentów przetwarzanych przez danego Mappera. Wymaga to zachowania stanu takiej tablicy pomiędzy różnymi wywołaniami funkcji map. Sposób ten nazywa się kombinacją wewnątrz Mappera (in-mapper combining), co pozwala programiście na większą kontrolę nad tym, kiedy agregacja jest realizowana, ale jednocześnie pociąga za sobą większe wymagania pamięciowe.

Algorithmic Correctness with Local Aggregation (1)

- Care must be taken in applying **combiners**, which **offer optional optimizations**
- The correctness of the algorithm cannot depend on computations performed by the combiner or depend on them even being run at all
- Keys: user ids and values: session length; aim: compute the mean session length on a per-user basis

identity mapper

```
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 [r1; r2; ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r1; r2; ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         ravg ← sum/cnt
9:         Emit(string t; integer ravg)
```

reducer cannot be used as a combiner

Mean(1, 2, 3, 4, 5) <> Mean (Mean(1, 2), Mean(3, 4, 5))

[15] Jeśli chodzi o wykorzystanie właściwego Combinera, to musi być to robione z rozwagą i troską o poprawność algorytmu. W pierwszej kolejności wprowadźmy jednak kolejny przykład zastosowania MapReduce, do którego świetnie się on nadaje. Przykład ten znacie choćby z systemów rekomendacyjnych, gdzie jednym z elementów realizacji predykcji oceny dla danego użytkownika było skorzystanie ze średniej jego ocen i średniej ocen innych użytkowników. Na podobnej zasadzie istotnym problemem jest obliczenie średniej długości sesji danego użytkownika lub centroidu dla grupy użytkowników lub dokumentów. Nasz Mapper zawiera tylko prostą funkcję map, której zadaniem jest emisja tego, co widzi lub przetwarza. W tym wypadku jest to klucz w postaci tekstowej (np. id użytkownika) oraz wartość liczbowa (np. długoś jego sesji). Taki Mapper nazywa się tożsamościowym (identity mapper). Reducer ma gwarancję uzyskania wszystkich liczb skojarzonych z tym samym kluczem. Realizując funkcję reduce, osobno liczy wartości licznika i mianownika, a potem dzieli je, by uzyskać średnią. Bardzo często jako Reducer może być wykorzystywany jako Combiner bez żadnych adaptacji. W tym wypadku tak nie jest, bo średnia z całego zbioru liczb, nie jest zawsze równa średniej ze średniej podzbiorów liczb (patrz przykład na dole slajdu). Spróbujmy więc wprowadzić dedykowanego Combinera. Pamiętać należy przy tym, że Combiner jest elementem optymalizacji wykonania algorytmu. Nie ma gwarancji jego wykonania, a poprawność algorytmu nie może zależeć od tego, co on robi i czy w ogóle będzie wykonany.

Algorithmic Correctness with Local Aggregation (2)

- Combiner partially aggregates results by computing the numeric components
- Combiners must have the same input and output key-value type, which also must be the same as the mapper output type and the reducer input type
- No guarantee on how many times combiners are called (it could be zero)

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

```
1: class Combiner
2:     method Combine(string t; integers [r1; r2; ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r1; r2; ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t; pair (sum,cnt))
```

```
1: class Reducer
2:     method Reduce(string t; pairs [(s1,c1); (s2,c2); ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s,c) ∈ pairs [(s1,c1); (s2,c2); ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         ravg ← sum/cnt
9:         Emit(string t; integer ravg)
```


[16] W pierwszej propozycji Combiner będzie agregował wyniki częściowe potrzebne do obliczenia średniej. Niech będzie to licznik i mianownik. Ten pierwszy jest sumą wartości dla tego samego identyfikatora, a ten drugi liczbą tych liczb (a więc sumą jedynek). Problem z tą propozycją jest taki, że nie ma gwarancji wykonania Combinera, więc jeśli by go wykreślić, to wszystko powinno być poprawne. Tymczasem nasz Mapper emituje wartości w postaci pojedynczych liczb, czego spodziewa się na swoim wejściu Combiner. Ten ostatni emituje wartości pośrednie w postaci pary (licznik, mianownik), czego z kolei spodziewa się na swoim wejściu Reducer. Błąd techniczny polega na tym, że jeśli wykasować Combinera (który przecież może się nie wykonać ani razu), to Mapper operuje na wartościach w postaci pojedynczych liczb, a Reducer spodziewa się par. Kod ten jest więc poprawny.

Algorithmic Correctness with Local Aggregation (3)

- With no combiners, the mappers would send pair directly to the reducers
- With combiners, the algorithm would remain correct (the combiners aggrgate partial sums and counts to pass along to the reducers)

```
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 [(s1,c1); (s2,c2), ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s,c) ∈ pairs [(s1,c1); (s2,c2); ...] 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 [(s1,c1); (s2,c2); ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s,c) ∈ pairs [(s1,c1); (s2,c2); ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         ravg ← sum/cnt
9:         Emit(string t; integer ravg)
```


[17] Rozwiązanie problemu zaprezentowanego na poprzednim slajdzie jest bardzo proste. Wystarczy bowiem zmodyfikować funkcję map Mappera tak, by emitowała wartości w postaci par, tj. każdą liczbę skojarzała z jedynką, oznaczającą, że liczba ta pojawiła się raz. Dzięki temu Mapper dostarczyłby prawidłowego formatu danych zarówno dla Combinera (jeśli ten byłby wykonany), jak i Reducera (jeśli optymalizacja lokalna byłaby pominięta).

Pairs and Stripes: much is up to you...

Construct complex keys and values so that data necessary for a computation are naturally brought together by the framework

```
1: class Mapper
2:     method Map(docid a; doc d)
3:         for all term w ∈ doc d do
4:             for all term u ∈ Neighbors(w) do
5:                 Emit(pair (w,u); count 1)

1: class Reducer
2:     method Reduce(pair p; counts [c1; c2; ...])
3:         s ← 0
4:         for all count c ∈ counts [c1; c2; ...] do
5:             s ← s + c
6:         Emit(pair p; count s)
```

```
1: class Mapper
2:     method Map(docid a; doc d)
3:         for all term w ∈ doc d do
4:             H ← new AssociativeArray
5:             for all term u ∈ Neighbors(w) do
6:                 H{u} ← H{u} + 1
7:             Emit(term w; stripe H)

1: class Reducer
2:     method Reduce(term w; stripes [H1; H2; ...])
3:         Hf ← new AssociativeArray
4:         for all stripe H ∈ stripes [H1; H2; ...] do
5:             Sum(Hf, H)
6:         Emit(term w; stripe Hf)
```

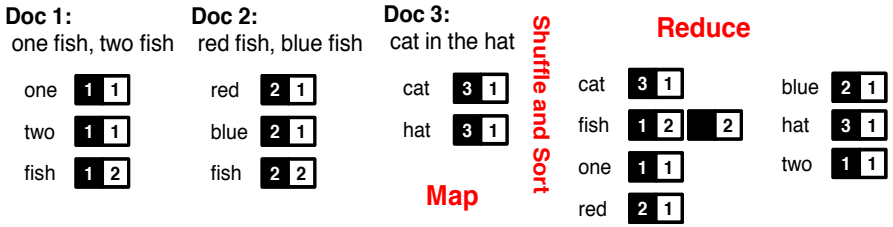
Pairs — Build **word co-occurence matrix** (query extensions, corpus linguistics, statistial NLP) — **Stripes**

- Immense number of key-value pairs
- Simple values
- Fewer opportunities for partial aggregation
- In-mapper combining requires sorting partial counts

- More compact representation and less sorting to perform
- Values more complex
- Combiners have more opportunities to perform aggregation
- Tha associative table may grow to be quite large

[18] Inny przykład, do którego MapReduce świetnie się nadaje, a jednocześnie dowodzi, że wciąż wiele zależy od programisty dotyczy obliczeń dla macierzy współwystępowania termów. Jest to przydatne choćby w przypadku realizacji podstawowych technik rozszerzania zapytań, jak i modelowanie języka. Dwa alternatywne sposoby rozwiązania tego problemu opierają się na wykorzystaniu albo złożonych kluczy (po lewej) albo złożonych wartości (po prawej). Wykorzystanie kluczy w postaci par pozwala na uproszczenie formatu wartości, które są pojedynczymi liczbami, ale zwiększa liczbę kluczy, dając tym samym mniejszą szansę dla agregacji lokalnej. Z kolei wykorzystanie kluczy w postaci pojedynczych termów zmniejsza ich liczbę i zwiększa szansę agregacji wyników. Wymaga jednak większej złożoności wartości, które w tym wypadku są przechowywanej w tablicy, której każdy element skojarzony jest z innym termem. Tablice takie mogą osiągać bardzo duże rozmiary, co pociąga za sobą problemy z pamięcią. Oczywiście nie ma uniwersalnego wskazania, który z tych dwóch sposobów jest lepszy. W dużej mierze wybór zależy od rozmiaru przetwarzanej kolekcji dokumentów.

Inverted Indexing (1)



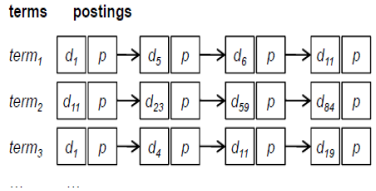
- Scalability is critical
- Must be relatively fast, but need not be real time
- Fundamentally a batch operation
- Incremental updates may or may not be important
- For the web, crawling is a challenge in itself

Perfect for MapReduce!

[19] Skupmy się teraz na indeksowaniu, a dokładniej na tworzeniu indeksu odwrotnego (ang. inverted index). Można go postrzegać jako książkę telefoniczną, która dla danego termu mówi, gdzie, tj. w których dokumentach, można go odnaleźć. W przykładzie na slajdzie obok identyfikatora dokumentu w indeksie przechowywana jest też liczba wystąpień termu w tym dokumencie. Zadanie Mappera polega więc na przetwarzaniu dokumentu, a Reducera na agregacji, czyli sklejaniu wyników dotyczących tego samego termu. Do realizacji tego typu zadań MapReduce ponownie nadaje się świetnie z kilku powodów. Ważna jest tu skalowalność, bo liczba dokumentów i termów do przetworzenia jest wysoka. Istotna jest też efektywność, ale zadanie nie mus być realizowanie w czasie rzeczywistym. W istocie często wykonywane jest partiami, a jako użytkownicy godzimy się z tym, że jest ono kosztowne i jego realizacja wiąże się z opóźnieniami. Gdy w sieci pojawi się jakiś dokument zajmuje przecież trochę czasu zanim będzie on zindeksowany i zacznie pojawiać się w wynikach wyszukiwania.

Inverted Indexing (2)

```
1: class Mapper
2:     method Map(docid a; doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1
6:         for all term t ∈ H do
7:             Emit(term t; posting <a,H{t}>)
```




```
1: class Reducer
2:     method Reduce(term t; postings [<a1,f1>; <a2,f2>; ...])
3:         P ← new List
4:         for all posting <a,f> ∈ postings [<a1,f1>; <a2,f2>; ...] do
5:             Append(P, H)
6:         Sort(P)
7:         Emit(term w; postings P)
```


[20] W zastosowaniu do utworzenie indeksu odwrotnego, Mapper zajmuje się przetwarzaniem dokumentu. Wykorzystuje tablicę asocjacyjną, by dla danego termu zliczyć i przechować liczbę jego wystąpień w tym dokumencie. Jako klucz emitowany jest term, a jako wartości zarówno identyfikator dokumentu, jak i liczba wystąpień termu w jego ramach. Warto tu podkreślić, że Mapper mógłby też realizować szereg innych operacji, które znacie z pierwszych wykładów, takich jak tokenizacja, normalizacji, stemming lub lematyzacja. Reducer ma gwarancję, że dostanie informacje o wystąpieniach danego termu we wszystkich dokumentach. Operacja przez niego realizowana jest prosta. Polega ona na sklejeniu wartości w ramach listy oraz jej posortowaniu (np. po identyfikatorach dokumentów).

Retrieval

Retrieval

- Look up postings lists corresponding to query terms
- Traverse postings for each query term
- Store partial query-document scores in *accumulators*
- Select top *k* results to return (only relatively few results)
- Must have sub-second response time

MapReduce is fundamentally **batch-oriented**

- Optimized for throughput, not latency
- Startup of mappers and reducers is expensive

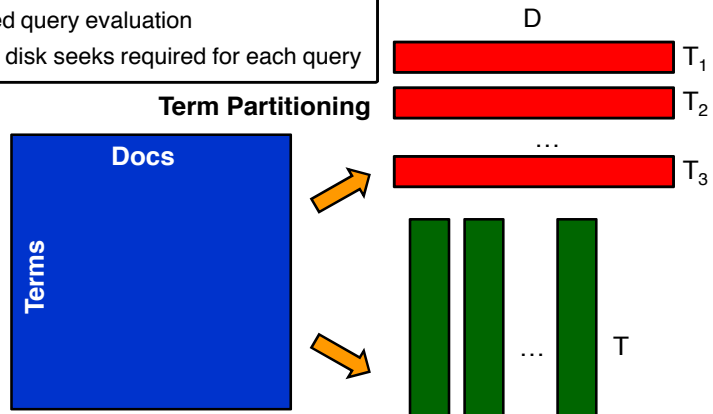
MapReduce is not suitable for real-time queries

[21] Przykłady podane do tej pory wskazywały na adekwatność wykorzystania MapReduce w ich kontekście. Podajmy teraz zastosowanie, w którym taka rekomendacja nie ma miejsca. MapReduce nie powinno stosować się w wyszukiwaniu, a więc obsłudze zapytań użytkownika w czasie rzeczywistym. Wymaga to przeglądania indeksów tylko w poszukiwaniu termów wykorzystanych w zapytaniu, zachowania częściowych wyników, zwrócenie tylko kilku wyników (dokumentów) - bardzo niewielu w kontekście całej sieci oraz bardzo szybkiej odpowiedzi. Tymczasem MapReduce jest zorientowany na przetwarzania całościowe, optymalizację przepustowości i wydajności, a nie realizację dostępu losowego, zachowywanie stanów czy minimalizację opóźnienia. Z tym ostatnim kłóci się choćby wysoki koszt uruchomienia Mapperów i Reducerów.

Term vs. Document Partitioning

- Pipelined query evaluation
- Smaller disk seeks required for each query

Term Partitioning



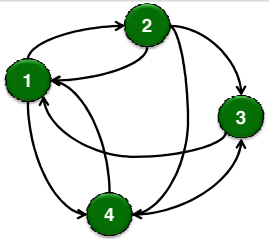
Document Partitioning

- Requires a query broker (forward, merge, return)
- Query processed by every partition server
- Shorter query latencies

Better (see Google)

[22] Obsługa zapytań jest jednak dobrym przyczynkiem do tego, by wspomnieć o tym, jak przechowywane są informacje o dokumentach, których nie można zmieścić na jednym serwerze. Możliwości podziału takiej dużej macierzy termy na dokumenty są dwie: albo wierszami albo kolumnami. W tym pierwszym przypadku na pojedynczym serwerze przechowywana jest kompletna informacja o danym termie, a w tym drugim zupełna informacja o dokumencie. Choć obsługa zapytania wymaga tu kontaktu z każdym serwerem, to w przypadku błędu któregokolwiek z nich jesteśmy w stanie wciąż obsłużyć zapytanie, nawet jeśli wyniki będą pomijały jakiś mały podzbiór dokumentów. Strategia z podziałem względem dokumentów (document partitioning) jest w praktyce wykorzystywana przez Google.

PageRank (1)

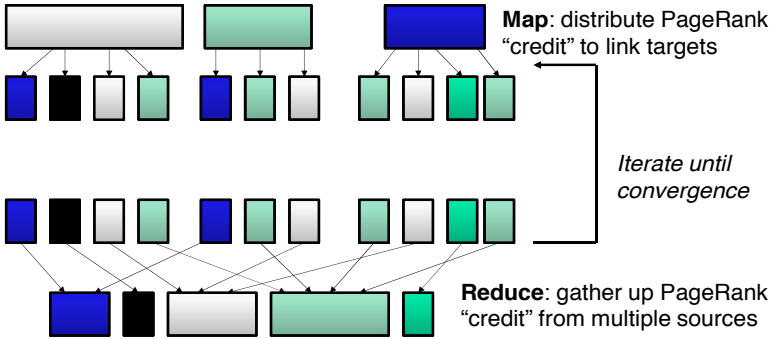


Adjacency matrix

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

Adjacency list

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

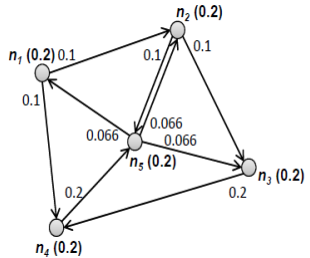


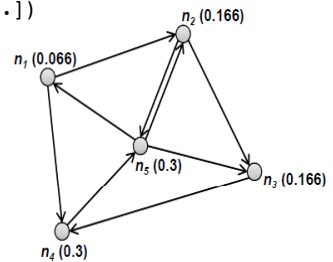
[23] Wróćmy do przykładów zastosowań, do których MapReduce nadaje się świetnie. Należą do nich różnego typu algorytmy grafowe, które zostaną przypomniane poprzez przywołanie zasady działania PageRank. Graf przedstawiony na slajdzie jest wewnętrznie reprezentowany w postaci listy sąsiedztwa, tj. dla każdego wierzchołka dysponujemy listą jego następników. PageRank można wyjaśnić z punktu widzenia działania każdego wierzchołka jako nadawcy mocy oraz jej odbiorcy. W pierwszej fazie, realizowanej przez funkcję map, każdy wierzchołek dzieli swój aktualny PageRank po równo do swoich następników. Z kolei w fazie reduce, każdy wierzchołek agreguje dedykowane dla niego cząstki mocy, po prostu dodając to, co pochodzi od różnych nadawców. Taki sposób postępowania musi być powtórzony kilka razy, bo wartości PageRank w metodzie iteracyjnej są wiarygodne dopiero po kilku iteracjach.

PageRank (2)

```
1: class Mapper
2:       method Map(nodeid n; node N)
3:           p ← N.PageRank/|N.AdjacencyList|
4:           Emit(nid n; N)
5:           for all nodeid m ∈ N.AdjacencyList do
6:               Emit(nodeid m; p)

1: class Reducer
2:       method Reduce(nodeid m; counts [p1; p2; ...])
3:           M ← empty set
4:           for all p ∈ counts [p1; p2; ...] do
5:               if IsNode(p) then
6:                   M ← p
7:               else
8:                   s ← s + p
9:           M.PageRank ← s
10:          Emit(nodeid m; node M)
```





[24] Przeanalizujmy pseudokod zastosowania MapReduce do obliczenia wartości PageRank stron w grafie. Mapper działa dla danego wierzchołka, dzieląc jego moc przez liczbę sąsiadów. Emituje samego siebie, by możliwe było odtworzenie struktury sieci przez Reducera i w następnej iteracji, a także identyfikator każdego sąsiada z dedykowaną dla niego częścią mocy. Przykładowo, w sieci na slajdzie jest 5 wierzchołków, a każdy z nich ma na początku równą moc 0.2. Wierzchołek w lewym górnym rogu wysyła po 0.1 do każdego ze swoich dwóch sąsiadów. Reducer przetwarza wartości dwojakiego typu skojarzone z tym samym identyfikatorem wierzchołka. Jeśli wartość odpowiada informacji o wierzchołku (gdzie przechowywana jest m.in. informacja o sąsiadach), to jest ona odtwarzana. Jeśli jednak jest wartością numeryczną, to sumujemy ją, a ostateczny wynik zapisujemy jako aktualną wartość PageRank dla wierzchołka. Przykładowo, do wierzchołka środkowego trafiają dwa kawałki mocy: 0.1 z prawego górnego i 0.2 z lewego dolnego, co daje 0.3 w aktualnej iteracji. W związku z zasadą działania PageRank wynik poprzedniej iteracji staje się wejściem dla iteracji następnej. Nie zmienia to faktu, że jest to typ zastosowania, w którym MapReduce sprawdza się doskonale.

When is Map-Reduce (less) appropriate?

When it is appropriate?

- ☺ **Lots of input data**
  (e.g., *compute statistics over large amounts of text*)
  take advantage of distributed storage, data locality, …
- ☺ **Lots of intermediate data**
  (e.g., *postings*)
  take advantage of sorting/ shuffling, fault tolerance
- ☺ **Lots of output data**
  (e.g., *web crawls*)
  avoid contention for shared resources
- ☺ Relatively **little synchronization** is necessary

When it is less appropriate?

- ☹ **Data fits in memory**
- ☹ Large amounts of **shared data** is necessary
- ☹ **Fine-grained synchronization** is needed
- ☹ Individual operations are **processor-intensive**

[25] Na zakończenie pierwszej części omówmy cechy problemów, do których rozwiązania MapReduce jest dedykowany. Po pierwsze, ilość danych do przetworzenia musi być duża, co wymaga przetwarzania rozproszonego. Gdy mieszczą się one w pamięci, wykorzystanie MapReduce nie jest konieczne. Po drugie, problem powinien być dekomponowalny na mniejsze, które nie wymagają współdzielenia danych, a dla których wyniki końcowe można raportować oddzielnie. Po trzecie, realizowane operacje nie powinny być złożone. Po czwarte, liczba wyników częściowych powinna być wysoka, co daje potencjał dla sortowania, grupowania i obsługi błędów, ale nie wymaga skomplikowanej synchronizacji dla potrzeb całościowej agregacji.

History of Web Advertising

Banner ads (1995-2001)

- Initial form of web advertising
- Popular websites charged X$ for every 1000 “impressions” of ad
  - Called “CPM” rate (cost per mille/thousand)
  - Modeled similar to TV or magazine ads
- Untargeted to demographically targeted
- Low clickthrough rate (low return on investment (ROI) for advertisers)

[26] W drugiej części skupimy się na problemie Adwords, ale zanim to niego dojdziemy małe wprowadzenie do wykorzystania reklam w Internecie. W tym zakresie wszystko zaczęło się w połowie lat 90. ubiegłego wieku. Początkowo wykorzystywano bannery reklamowe, których użycie jest podobne do reklam telewizyjnych. Nie były one w żaden sposób ukierunkowane na konkretnego użytkownika, a opłata reklamodawcy następowała albo za 1000 wyświetleń albo, w późniejszym okresie, za 1000 kliknięć. Skuteczność takich reklam nie była jednak wysoka.

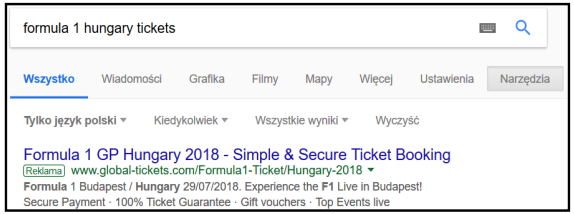
Performance-based Advertising

Introduced by Overture (GoTo) around 2000

- Advertisers "**bid**" **on** search **keywords**
- When someone searches for that keyword, **the highest bidder's ad is shown**
- Advertiser is charged only if the ad is clicked on

Similar model adopted by Google with some changes around 2002 ("**Adwords**")

- **It works!**
- Multi-billion-dollar industry
- **Interesting problems**
  - What ads to show?
  - Which search terms should I bid on /how much to bid?



[27] Przełom, choć wtedy jeszcze nie zdawano sobie z tego sprawy, nastąpił w 2000r. Wtedy firma Overture opracował wyszukiwarkę GoTo. Wyszukiwarka ta analizowała zapytania użytkownika i automatycznie generowała płatne ogłoszenia skojarzone z wyrazami kluczowymi użytymi w zapytaniu. Wyniki wyszukiwania były podobne do rezultatów w innych wyszukiwarkach, z tą różnicą, że na początku listy znajdowały się odsyłacze do produktów lub firm, które opłaciły ogłoszenie. Reklamodawca musiał zapłacić tylko, gdy jego reklama była kliknięta. Pomimo że GoTo nie zyskało wielkiej popularności, to firma Overture została ostatecznie wykupiona przez Yahoo, a podobny model szybko zaczęło wykorzystywać w swojej wyszukiwarce Google. Dziś wiemy, że takie podejście sprawdza się doskonale, a biznes ten generuje miliardy dolarów zysku. W ramach wstępu do tej tematyki chcemy skupić się na tym, jakie reklamy wybierać do wyświetlenia.

Adwords Problem

- A stream of queries arrives at the search engine
  $q_1, q_2, q_3, \ldots$
- Several advertisers bid on each query
- When query q_i arrives, search engine must pick a subset of advertisers whose ads are shown
- **Goal**: maximize search engine's revenues
- An online algorithm is needed!

"You get to see the input one piece at a time, and need to make irrevocable decisions along the way"

[28] Założenia problemu Adwords są następujące. Do obsłużenia jest strumień zapytań, które pojawiają się w wyszukiwarce. Wielu reklamodawców obstawia słowa kluczowe, które potencjalnie pojawiają się w zapytaniach. Gdy pojawia się jakieś zapytanie, wyszukiwarka musi wybrać podzbiór reklamodawców, których reklamy zostaną pokazane. Celem, co bardzo istotne, jest maksymalizacja zysku wyszukiwarki, a nie reklamodawców. Potrzebny jest więc algorytm, który działałby w trybie online. Oznacza to, że po każdym zapytaniu algorytm musi zdecydować, co zrobić, a jego decyzje są nieodwołalne. W szczególności, nie może poczekać do końca dnia, aby przeanalizować całą sekwencję zapytań i zoptymalizować swoje działanie i zyski na większej sekwencji.

[29] Rozpoczniemy od omówienia zasad działania uproszczonego algorytmu Balance, który został opracowany na Uniwersytecie Stanford. Opiera się on na nierealistycznych założeniach, które potem zostaną poluźnione. Budżet każdego reklamodawcy jest równy, oferty na słowa kluczowe mają charakter binarny (albo 0 albo 1), a dla danego zapytania wyświetla się tylko jedną reklamę. Algorytm działa w sposób zachłanny, dla każdego zapytania wybierając tego reklamodawcę, który ma największy niewydany budżet. W przypadku remisu, zwycięzca zostanie wybrany arbitralnie (my przyjmiemy, że premiowany będzie niższy identyfikator reklamodawcy).

Simplified BALANCE

Simplified setting:
- Each advertiser has the same budget G
- One advertiser per query
- Assume all bids are 0 or 1

The greedy algorithm
Arbitrarily pick an eligible advertiser for each keyword

Simple BALANCE
- For each query, pick the advertiser with the largest unspent budget
- Break ties arbitrarily

Example: two advertisers A (bids on query x) and B (bids on x and y) with budget of $4

query stream	x	x	x	x	y	y	y	y
worst greedy	B	B	B	B	-	-	-	-

query stream	x	x	x	x	y	y	y	y
BALANCE	A	B	A	B	B	B	-	-

competitive ratio = 4/8

optimal	A	A	A	A	B	B	B	B

competitive ratio = 6/8

- For input I, suppose greedy produces matching M_{greedy} while an optimal matching is M_{opt}

$$\textbf{competitive ratio} = \min_{\text{all possible inputs I}} (|M_{greedy}|/|M_{opt}|)$$

- In the general case, the worst competitive ratio of BALANCE is $1-1/e \approx 0.63$
- No online algorithm has a better competitive ratio

Rozważmy przykład, w który jest dwóch reklamodawców z budżetem w wysokości 4 dolarów: A obstawia słowo kluczowe x, natomiast B zarówno x, jak i y (oferty w wysokości 1 dolara). W rozważanym przykładzie najpierw pojawia się sekwencja czterech zapytań ze słowem kluczowym x, a potem czterech zapytań ze słowem kluczowym y. W przypadku wykorzystania najbardziej prymitywnego zachłannego algorytmu, w najgorszym razie dla każdego zapytania z x, wyświetlilibyśmy reklamę B, a dla y nie byłoby już zainteresowanego reklamodawcy z niezerowym budżetem. Twórca wyszukiwarki zarobiłby więc 4 dolary na 8 maksymalnie możliwych. Dałoby się bowiem zarobić 8, jeśli dla 4x wyświetlić zawsze A, a dla 4y wyświetlić B. Stosunek tego, co zwraca algorytm, do tego, jaki najlepszy wynik mógłby zwrócić, nazywa się competitive ratio. W tym wypadku, wynosiłoby ono $4/8$. Z kolei BALANCE, dla pierwszego zapytania z x, wykryłby remis między A i B (po 4 niewydane dolary) i arbitralnie rozstrzygnął go na korzyść A. Przy drugim zapytaniu, zainteresowani są A i B, ale to B ma większy niewydany budżet, więc jego reklama jest wyświetlona, itd. Przy siódmym zapytaniu (z y) zainteresowany jest B, ale nie ma już pieniędzy, itd. Ostatecznie zarobilibyśmy 6 dolarów na 8 możliwych. Współczynnik competitive ratio można uogólnić do wszystkich możliwych instancji i ocenić algorytm względem najgorszego możliwego zysku, który oferuje. W przypadku BALANCE takie competetive ratio wynosi 1-1/e (e jest liczbą Eulera), co stanowi ok. 63% zysku, który moglibyśmy uzyskać analizując sekwencję zapytań post factum. Jest to świetny wynik i żaden algorytm, działający w trybie online, nie oferuje lepszego.

Generalized BALANCE

- Each advertiser has a limited, but **arbitrary budget (and arbitrary bids)**
- Search engine guarantees that the advertiser will not be charged more than their daily budget

- Simple BALANCE can be terrible
  Consider two advertisers A and B
  A: bid $x_1 = 1$, budget $b_1 = 110$ B: bid $x_2 = 10$, budget $b_2 = 100$

- Allocate query q to bidder i with largest value of $\psi_i(q)$

amount spent so far = m_i

bid = x_i **fraction of budget left over $f_i = 1-m_i/b_i$**

$\psi_i(q) = x_i(1-e^{-f_i})$ budget = b_i

- Same competitive ratio (1-1/e)

[30] Co jednak w wypadku, gdy pozbyć się nierealistycznych założeń odnośnie takiego samego budżetu oraz takiej samej wysokości ofert? BALANCE w wersji prostej nie zadziałałby. Wystarczy przeanalizować przykład, w którym budżet A wynosi 110, a B - 100; z kolei oferta A to 1, a B to 10. Algorytm długo preferowałby A z większym niewydanym budżetem, ale jego zysk byłby mizerny (1+1+1+...). Musimy więc wykorzystać uogólnioną wersję algorytmu BALANCE. Wykorzystywana jest w nim funkcja, która dla zapytania q zwraca wyniki dla każdego z reklamodawców oznaczonego przez i. Im większa jej wartość, tym bardziej preferowany reklamodawca. Wartość tej funkcji zależy od wysokości oferty tego reklamodawcy oraz funkcji, która rośnie asymptotycznie wraz z procentem niewydanego budżetu (mi/bi to procent wydany; 1-mi/bi to procent niewydany). Ostatecznie, im wyższa twoja oferta oraz im wyższy procent niewydanego budżetu, tym większe twoje szanse na wyświetlenie reklamy. Co ciekawe, działając przy dowolnych wysokościach budżetów i ofert, algorytm oferuje takie samo competitive ratio jak jego wersja uproszczona.

Simplified Google Adwords

- Each ad has a different likelihood of being clicked
  - Advertiser A bids x=$2, click probability = 0.1
  - Advertiser B bids x=$1, click probability = 0.5
  - Clickthrough rate (CTR) measured historically
- Simple solution (initial Google Adwords algorithm)
  - Instead of raw bids, use the “expected revenue per click”
  - $\psi_i(q) = x_{i,q} \cdot CTR_i$ (for A: $2 \cdot 0.1 = 0.2 <$ for B: $1 \cdot 0.5 = 0.5$)

[31] Na koniec wspomnijmy o wersji algorytmu Adwords wykorzystywanej na samym początku przez Google. Choć dziś trudno w to uwierzyć, to ok. 2003r. Google brało pod uwagę iloczyn wysokości oferty oraz miary clickthrough rate, która odzwierciedla skuteczność reklamy (ile procentowo razy została kliknięta w stosunku do liczby wyświetleń). Miara taka ujmuje więc oczekiwany zysk z wyświetlenia reklamy. Oczywiście Google szybko zorientowało się, że na problemie Adwords można zarobić więcej. Pierwsze zmiany, które wprowadzili, dotyczyły określenia szerszego tematu w funkcji oceny, wzięciu pod uwagę całej historii konta, jakości strony, do której prowadziła reklama, a także współczynnika lokalizacji, odzwierciedlającego skuteczność reklamy w różnych częściach świata. Dziś działanie Adwords jest na tyle skomplikowane, że nie bez powodu istnieją firmy, które specjalizują się w optymalizacji kampanii reklamowych prowadzonych w Internecie.

Summary (1)

Four advertisers A, B, C, and D with a daily budget of $2 bid for the following keywords ($1 each): A: w, x; B: x, z; C: x, y; D: y, z.

I) Use a simplified version of BALANCE to select the ads for the following queary stream (in case of a tie use the following order for breaking it A > B > C > D):

query stream	x	y	w	z	z	w	y	x
BALANCE	A	C	A	?	?	?	?	?
OPTIMAL								

II) What would be the optimal offline allocation of ads for the above query stream? How does it compare with the worst case evaluation for BALANCE?

III) Compute a competetive ratio for the above given data.

[32] I)
II)
III)

Summary (2)

Three advertisers A, B, and C compete for the same keyword with the following bids: $1, $2 and $5, respectively. Their respective CTRs (click through rates are 0.5, 0.1 and 0.2).

I) Use simple Google Adwords algorithm to rank the advertisers. Whose ad would be selected?

II) How B needs to increase her bid so that to be ranked first?

[33] I)
II)