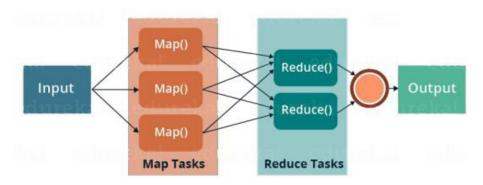
Map, filter, reduce pattern v NoSQL světě

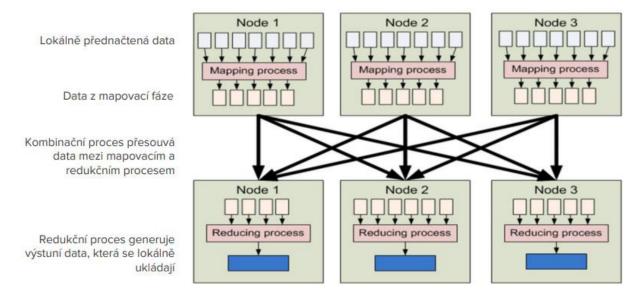
Úloha: Máme historická data o počasí mezi lety 2000 a 2015. Tato data obsahují průměrné teploty za jednotlivé dny a jednotlivé oblasti. Jsou uloženy v místech, kde se tato data sbírají. Chceme najít nejvyšší teplotu v každém roce z dat přes celou zeměkouli.

- problém kritické cesty zpoždění stroje při řešení podúlohy by nemělo zpodtit následující milník nebo celou úlohu
- spolehlivost jednotlivých částí systému co kdyby vypadla některá komponenta systému?
- rovnoměrné rozdělení úlohy jak rozdělit úlohy na menší bloky tak, aby komplexita a časová náročnost byla odpovídající výkonnosti stroje → žádný samostatný stroj by neměl být přetížen nebo nevyužíván
- jedna podčást úlohy se nespočítá pokud některý ze strojů nespočítá výstup, tak nebudeme schopni vypočítat finální výsledek → musí existovat mechanismus, který je schopný sestavit řešení i v této situaci
- agregace výsledku měl by ex. mech. pro agregaci výsledku, který generuje každý stroj do finálního výstupu
- → MapReduce framework řeší tyto výzvy
- nutnou podmínkou je řešitelnost úlohy principem Rozděl a panuj
- → relaxuji řešení tak, že se spokojím i se suboptimálním výsledkem

Map reduce framework

- programovací framework, který umožňuje řešit úlohy, ve kterých se paralelně počítají úlohy nad obrovským množstvím dat, které se nacházejí v různých lokalitách
- pro řešitele je plně transparentní fyzické uložení dat a jejich distribuovatelnost
- funguje v následující sekvenci:
 - o Input načtení a preprocessing dat
 - Map přemapování vstupních dat do key-value párů
 - o Shuffle kombinování (přeskládání) mezivýstupů pro reduce fázi
 - o Reduce (redukce) agregace dat do menších key-value párů
 - Output postprocessing a zápis dat





- výhody MapReduce:
 - paralelní zpracování úloha je rozdělena na základě principu Rozděl a panuj a je řešena paralelně
 - lokální zpracování místo přesouvání dat do centra (kde probíhá proces zpracování), tak přesouváme proces zpracování tam, kde jsou data

Mapovací fáze

- probíhá na zdrojových/lokálních strojích

record reader

- o parsuje vstupní data do jednotlivých záznamů (records)
- o nejčastěji do formy <key, value>, kde key = kontextová data, value = obsah záznamu
- o klíč později použit pro grupování, value zpracovávána reducerem
- map mapuje <key, value> páry na jiné <key, value> páry
- **combiner** optional; lokálně běžící reducer, který je proveden v map fázi (kvůli výkonu)
- partitioner vezme <key,value> páry a rozdělí je do skupin (každá pro jeden reducer)

Reduce fáze

- probíhá na cílových strojích

- shuffle and sort

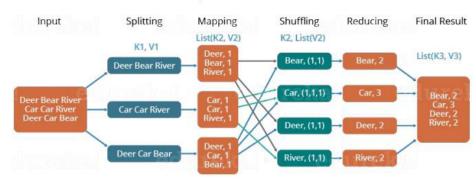
- o probíhá automaticky na pozadí
- programátor ji nemůže ovlivnit
- posbírá si připravené skupiny a přenese je na stroje, kde běží pro ně určené reducery
- **reducer** provádí redukční operaci nad připravnými daty
- **output format** "collector" co posbírá data z reduceru, zformátuje a zapíše n výstup (file,...)

MapReduce framework - příklad

Máme velké množství souborů uložených na různých strojích v různých lokalitách. Pro zjednodušení uvažujme jeden soubor *example.txt*, který má následující obsah: *Dear, Bear, River, Car, River, Deer, Car, Bear*

Cílem je spočítat výskyty jednotlivých slov a ty zapsat na výstup

The Overall MapReduce Word Count Process



- rozdělíme vstup na tři kusy (mohou být v různých lokalitách); rozkládá práci na více strojů
- každý kus projdeme a přepíseme do mapy KV párů; klíč = slovo, value = hodnota 1 (výskyt)
- data se přeskládají tak, aby každý ředucer zpracovával stejné slovo, pošlou se do reduceru
- po fázi mapování a míchání bude mít každý reducer jedinečný klíč a seznam hodnot odpovídajících tomuto klíči
- každý reducer počítá hodnoty, které jsou v tomto seznamu přítomny
 - o dostane seznam hodnot, např. pro Bear [1,1]
 - o počítá jejich počet v samotném seznamu > konečný výstup (Bear, 2)
- všechny výstupní páry key-value posbírány a zapsány do výstupního souboru

CAP Theorem

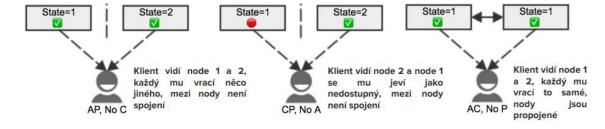
- tvrdí, že pro distribuované datové úložiště není možné poskytovat více jak dvě záruky z:
 - Consistency (konzistence)
 - všichni klienti dostávají stejná data
 - každé čtení vrátí buď výsledek posledního zápisu, nebo chybu

Availability (dostupnost)

- data lze dostat i v případě, že došlo k výpadku několika nodů
- na každý dotaz přijde (nechybová) odpověď

Partition tolerance (odolnost k přerušení)

 systém funguje dál i v případě, že dojde ke zdržení či ztrátě části zpráv v rámci sítě



Sumarizační patterny

Agregátor

Cíl: zgrupování záznamů dohromady dle klíčového atributu, spočítání agregační funkce nad takto zgrupovanými záznamy

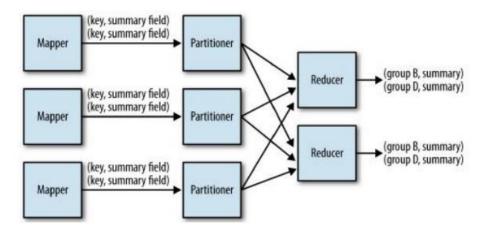
Příklady: minimum, maximum, průměr, medián,...

Motivace:

- většina dat je příliš jemná a v přílišném množství, než aby šlo vypozorovat něco zajímavého
- např. web loguje aktivity uživatele, výsledkem jsou TB dat
 - zgrupování aktivit dle hodiny a počtu záznamů → kdy je web aktivní
 - o zgrupování reklam ve spojení s aktivitou → efektivita reklam

Snažíme se o něco podobného jako je SQL:

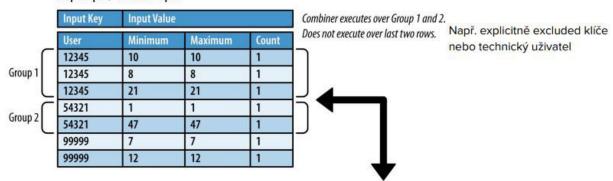
SELECT MIN(numericalcol1), MAX(numericalcol1), COUNT(*) FROM table GROUP BY groupcol2;



Mapovací fáze: klíče obsahují hodnoty atributů podle kterých chceme grupovat a číslo, které budeme agregovat Kombinační fáze: Sloučíme záznamy, které mají stejné klíče a ty pošleme do redukce

Redukční fáze: Reducer dostane skupinu a k ní seznam čísel které agreguje

Map Output / Combiner Input



Combiner Output / Reducer Input

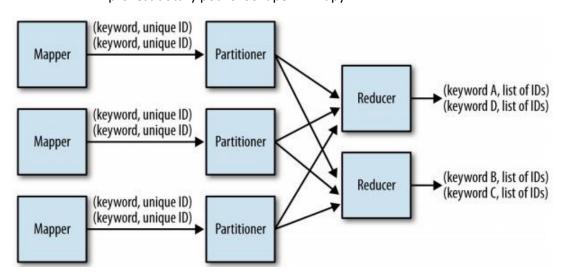
Output Key	Output Value		
	Minimum	Maximum	Count
12345	8	21	3
54321	1	47	2
99999	7	7	1
99999	12	12	1

Inverzní index

Cíl: vytvořit index k datům, abychom mohli provádět rychlejší hledání

Motivace:

- chceme vyhledávat na webu pomocí klíčových slov
- procházení všech stránek po zadání klíčového slova je nepoužitelné
- projdeme si všechny stránky dopředu a vytvoříme si mapu klíčových slov
 - seznam stránek, ve kterých se slovo vyskytuje
 - konstrukce indexu trvá dlouho (a zabírá místo), ale po vytvoření lze extrémně rychle provést dotazy pouze lookupem z mapy



Filtrační patterny

Filtr

Cíl: profiltrovat data tak, abychom dostali ta, která jsou pro nás zajímavá

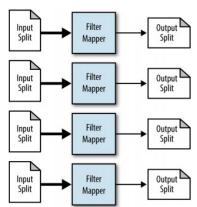
- prakticky se jedná o hromadnou aplikaci funkce f, vracející na záznam true/false

Motivace:

- prohledáváme záznamy o plánovaných eventech
- zajímají nás jen takové, které se týkají BigData (buď je v textu, nebo v tagu)
- po vybrání zajímavé podmnožiny dat pokračujeme s analýzou pouze v této podmnožině

Snažíme se o něco podobného jako je SQL:

SELECT * FROM table WHERE value <3;



- mapovací fáze: pokud nás záznam zajímá, pošleme jej dále na výstup (jinak zahazujeme)
- komb. / red. fáze: není
- příklady:
 - distribuovaný grep
 - tracking vlákna událostí
 - o předčištění dat

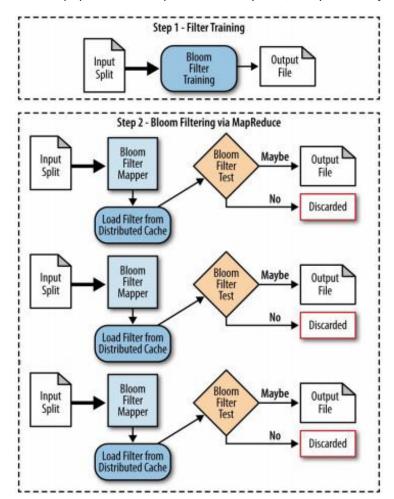
Bloom filtr

Cíl: filtrovat tak, abychom dostali data, ve kterých jsou zástupci z předem definované množiny

- ostatní se zahazují

Motivace:

- chceme vybrat pouze záznamy týkající se předem def. množiny uživatelů
- "natrénujeme" bloom filtr na námi definované uživatelé a nahrajeme jej do distribuované cache, která je dostupná z mapper jobů
- popř. chceme např. realizovat systém, který kontroluje chat na výskyt vulgárních slov



- mapovací fáze: z distr. cache se nahraje bloom filter
 - o v jednoduchém případě např. seznam slov a aplikace tohoto filtru na data
- komb. / red. fáze: není

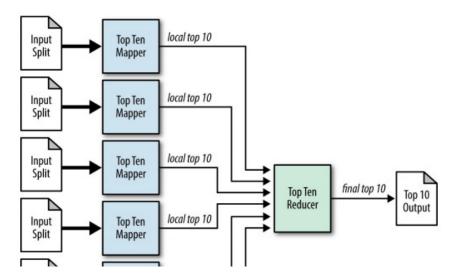
Top ten

Cíl: získat relativně malou skupinu K nejlepších záznamů dle nějaké ranking funkce z velkého mn. dat

Motivace:

- při statické analýze dat chceme najít okrajové hodnoty (outliers)
- lze-li definovat ranking funkci, která je schopná porovnat dva záznay, tak lze takto nalézt uspořádání, které funguje i přes veškeré záznamy
- nesnaží se setřídit všechny záznamy přes všechna data (extrémně náročné)
 - o ale vybrat menší množství dobře hodnocených záznamů

Snažíme se o něco podobného jako je SQL: SELECT * FROM table ORDER BY col4 DESC LIMIT 10;



- mapovací fáze: vybírá top ten lokální záznamy
- kombinační fáze: není
- **redukční fáze**: reducer setřídí záznamy a vybere finální top ten skupinu

Distinct

- profiltruje vstupní data tak, aby výstupem byly pouze unikátní záznamy
- příklad deduplikace; typy záznamů (ne jejich počty),...
- řešení:
 - map přemapujeme data do struktury, kde klíč = naše datové záznamy, které cheme mít unikátní; value = null
 - o reduce spojí záznamy se stejným klíčem do jednoho páru <key, null>
 - o output formatter zapíše na výstup pouze klíče

Join patterny – struktura do hierarchie

Cíl: propojení záznamů mezi sebou dle nějakého klíče

Motivace:

- chceme zmigrovat data z relační databáze do dokumentově orientované
- máme web s blogy na různé tématické okruhy, v rámci blogů se zadávají příspěvky a k nim vznikají revize

- data jsou uložena v relační databázi
- → při přechodu na stránku se její obsah poskládá z různých DB tabulek (náročné)
- → v rámci analýzy dat chceme korelovat délku příspěvku s délkou komentářů (komplikovaný join, pak extrakce dat)
 - bylo by jednodušší při grupování dat dle příspěvků tak, že jsou u nich komentáře a revize

Posts Post

036

Comment

Comment

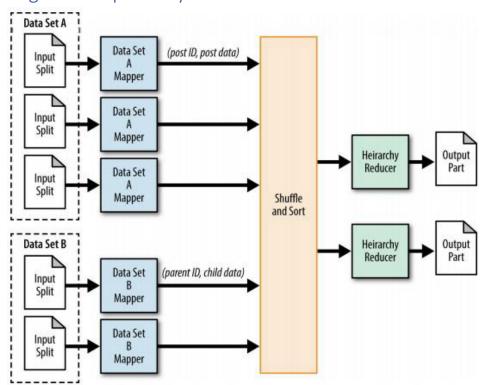
Post

Comment

Comment

Comment

Organizační patterny – struktura do hierarchie



- mapovací fáze:
 - výstupní klíč odpovídá tomu, jak chceme identifikovat kořen naši hierarchické struktury (např. postld, parentld=postld)
 - o typ dat lze identifikovat např. přidáním prefixu vlastním datům (P pro Post,...)
- kombinační fáze: nemá zde příliš význam
- redukční fáze:
 - reducer přebírá kolekci záznamů, které mají stejný klíč (tedy kolekci posts a comments se stejným ID)

_

- o při iteraci pak přebalí data do cílového formátu (např. obalí xml tagy)
- příklady:
 - o transformace dat pro MongoDB
 - o propojení dat přes klíč

Join patterny – kartézský součin

Cíl: propojení záznamů mezi sebou dle nějakého klíče

Motivace:

- potřebujeme udělat párovou analýzu přes všechny záznamy
- tedy např. potřebujeme spočítat vzájemnou korelaci pro všechny možné dvojice záznamů

Řešení:

- mám množiny (datovéí záznamy typů) A a B s různými hodnotami
- $A = \{1,2\}$
- $B = \{1,2,3\}$
- ...

