

Plan

- Hadoop - wprowadzenie
  - HDFS
    - Charakterystyka
    - Architektura
    - Inne funkcje
  - MapReduce
    - Wprowadzenie
    - Przykład Word Count
    - Narzędzia
  - Inne narzędzia do zarządzania w chmurze (Pig, HBase, Hive)

## Hadoop - wprowadzenie



- Potrzeba do przetwarzania ogromnych zbiorów danych na dużych klastrach komputerów
- Bardzo droga budowa niezawodności w każdej aplikacji
- Codzienne awarie węzłów
  - Awarie są spodziewane, aniżeli wyjątkowe
  - Liczba węzłów w klastrze nie jest stała
- Potrzebna wspólna infrastruktura
  - Wydajne, niezawodne, łatwe w użyciu
  - Otwarto-źródłowe, Licencja Apache

18

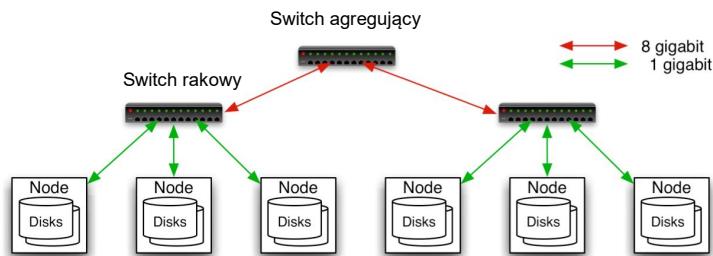
## Rozwiązania Hadoop



- Amazon/A9
- Facebook
- Google
- New York Times
- Veoh
- Yahoo!
- .... wiele innych

19

## Powszechny sprzęt



- Typowo w 2-warstwowej architekturze
  - Węzły to ogólniedostępne PCs
  - 30-40 węzłów/szafę
  - Uplink z szafy to 3-4 Gigabit/s
  - Wewnętrzna przeływnośc w szafie 1 Gb/s

## Hadoop Distributed File System (HDFS)

21

## Zalety HDFS



- Bardzo duży rozproszony system plików
  - 10K węzłów, 100 millionów plików, 10PB
- Działa na ogólnodostępnym sprzęcie
  - Pliki są replikowane aby obsłużyć przypadki awarii
  - Wykrywa awarie i jest w stanie sobie z nimi radzić
- Zoptymalizowany do przetwarzania wsadowego
  - Lokalizacje danych odsłonięte tak, że obliczenia mogą się odbywać w miejscu przechowywania danych
  - Zapewnia bardzo wysoką zagregowaną przepustowość



22

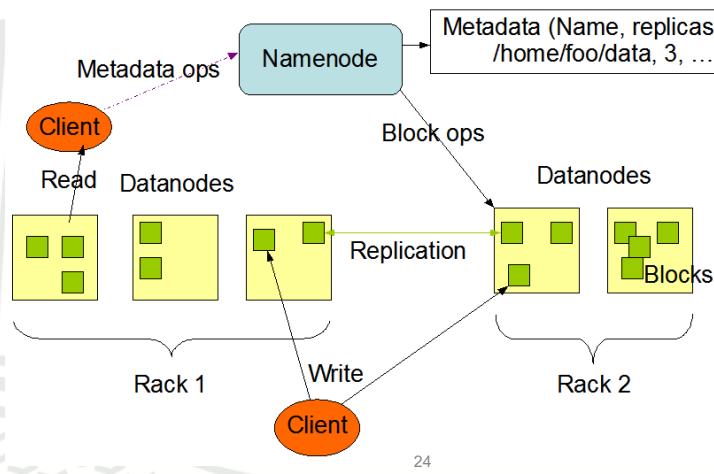
## Rozproszony system plików



- Pojedyncza przestrzeń nazw dla całego klastra
- Spójność danych
  - Model dostępu write-once-read-many
  - Klient może tylko dopisywać do istniejącego pliku
- Pliki są podzielone na bloki
  - Typowy rozmiar bloku to 64MB
  - Każdy blok jest powielany na wielu węzłach danych
- Inteligentny klient
  - Klient może odnaleźć lokalizację bloku
  - Klient uzyskuje dostęp do danych z węzła danych

23

## Architektura HDFS



24

## Węzeł główny (NameNode)



- Zarządzalna przestrzeń nazw systemu plików
  - Mapuje nazwę pliku do zestawu bloków
  - Mapuje bloki do węzłów danych (DataNode) w miejscu ich fizycznego składowania
- Zarządzanie konfiguracją klastra
- Silnik replikacji dla bloków

25

## Węzeł główny Metadane



- Metadane w pamięci
  - Bieżące metadane są w pamięci głównej
  - Brak zapotrzebowania stronicowania metadanych
- Typy metadanych
  - Lista plików
  - Lista bloków dla każdego pliku
  - Lista węzłów z danymi dla każdego bloku
  - Atrybuty plików, np. czas utworzenia, wsp. replikacji
- Log transakcyjny
  - Zapisuje tworzenie i usuwanie plików itp.

26

## Węzeł zarządzający danymi (DataNode)



- Serwer alokacji bloków
  - Przechowuje dane w lokalnym systemie plików (np. ext3)
  - Przechowuje metadane bloku (np. CRC)
  - Zwraca dane i metadane do klientów
- Raport bloków
  - Okresowo wysyła raport wszystkich istniejących bloków do węzła głównego (NameNode)
- Ułatwia przetwarzanie potokowe danych
  - Przekazuje dane do innych określonych węzłów danych

27

## Umiejscowienie bloków



- Bieżąca strategia
  - Jedna replika na węźle lokalnym
  - Druga replika w szafie zdalnej
  - Trzecia replika w tej samej zdalnej szafie
  - Dodatkowe repliki są umiejscawiane losowo
- Klienci odczytują z najbliższej repliki

28

## Tętno (Heartbeats)



- Węzły z danymi (DataNodes) wysyłają potwierdzenie do węzła głównego (NameNode)
  - raz na 3 sekundy
- Węzeł główny (NameNode) używa potwierdzeń do wykrycia awarii węzłów z danymi (DataNode)

29

## Silnik Replikacji



- Węzeł główny (NameNode) wykrywa awarie węzłów z danymi
  - Wybiera nowe węzły z danymi do ponownej replikacji
  - Balansuje użycie dysku
  - Równoważy ruch komunikacyjny do węzłów danych

30

## Awaria węzła głównego (NameNode)



- Pojedynczy punkt awarii
- Logi z transakcjami składowane w różnych katalogach
  - Katalog w lokalnym systemie plików
  - Katalog w zdalnym systemie plików (NFS/CIFS)

31

## Przetwarzanie potokowe danych



- Klient otrzymuje listę węzłów danych, na których można umieścić repliki bloku
- Klient zapisuje do pierwszego węzła DataNode
- Pierwszy DataNode przesyła dane do następnego węzła w potoku
- Gdy wszystkie repliki są napisane, klient przechodzi do następnego bloku zapisu w pliku

32

## Wyrównywacz (Rebalancer)



- Cel: procent zapełnienia dysków DataNodes powinien być podobny
  - Zwykle uruchamiany, gdy nowe węzły danych są dodawane.
  - Klaster jest w trybie online, kiedy Rebalancer jest aktywny
  - Rebalancer jest dławiony, aby uniknąć przeciążenia sieci
  - Narzędzia wiersza poleceń

33



## Zapasowy węzeł główny (NameNode)

- Kopiuje FSIImage i log transakcji z węzła głównego do katalogu tymczasowego
- Łączy FSIImage i log w nową strukturę FSIImage w katalogu tymczasowym
- Aktualizuje nowy oraz FSIImage na węźle głównym
  - Log transakcyjny węzła głównego jest czyszczony

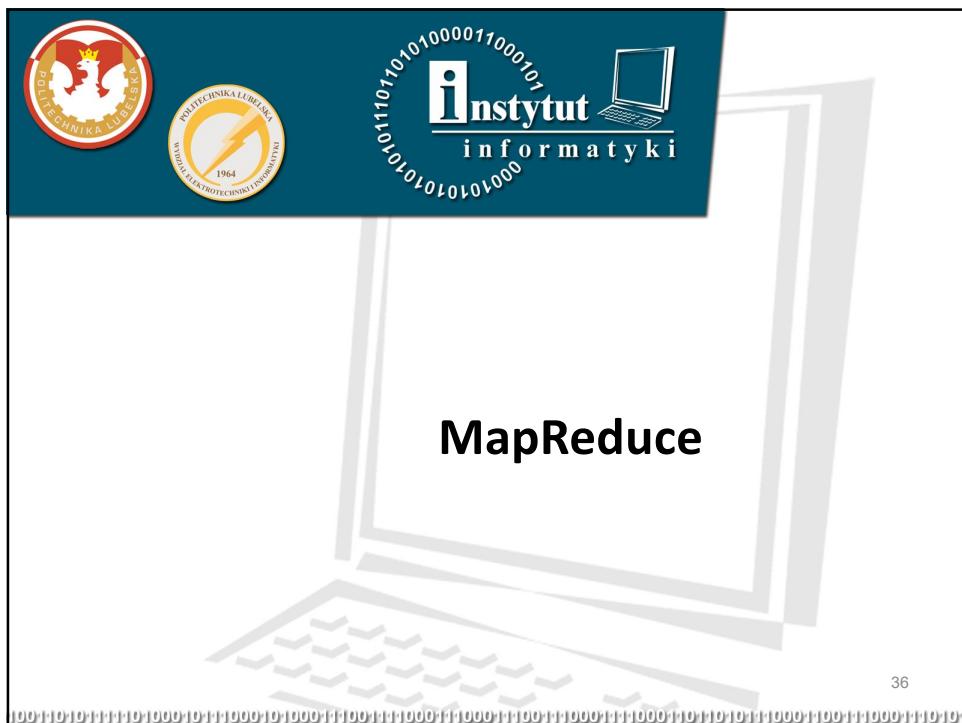
34



## Interfejs użytkownika

- Wiersz poleceń HDFS dla użytkownika:
  - hadoop dfs -mkdir /foodir
  - hadoop dfs -cat /foodir/plik.txt
  - hadoop dfs -rm /foodir/plik.txt
- Wiersz poleceń HDFS dla administratora
  - hadoop dfsadmin -report
  - hadoop dfsadmin -decommission  
datanodename
- Interfejs Web
  - <http://host:port/dfshealth.jsp>

35

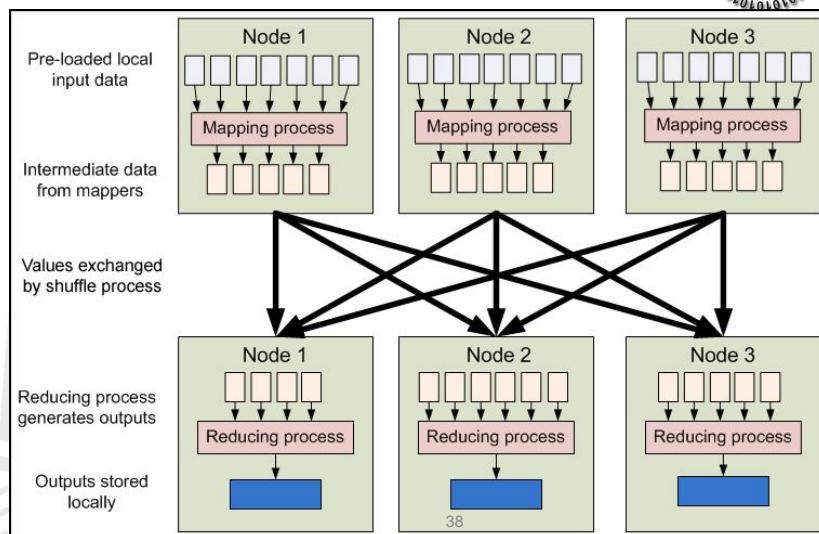


36

The slide features the logo of the Institute of Informatics at the Lublin Polytechnic. The logo is circular, containing the text "Instytut informatyki" and "POLITECHNIKA LUBELSKA". A computer monitor icon is positioned next to the text. Above the logo, there is a red circular emblem with a white bird and the text "POLITECHNIKA LUBELSKA". Below the main logo, there is a yellow circular emblem with a lightning bolt and the text "POLITECHNIKA LUBELSKA" and "WYDZIAŁ ELEKTROTECHNICZNO-INFORMATYCZNY 1964". The background of the slide shows a large computer monitor displaying the question "Co to MapReduce?". The monitor is set against a background of binary code (0s and 1s) and is surrounded by several smaller, overlapping monitors.

37

## MapReduce – przepływ danych



źródło: <http://developer.yahoo.com>

## MapReduce - Właściwości



- Podzielone na mniejsze zadania Map i Reduce
  - Udoskonalone równoważenie obciążenia
  - Szybszy powrót do stanu normalnego z awarii
- Automatyczne ponowne wykonanie w przypadku awarii
  - W klastrze, niektóre węzły są zawsze wolne lub ułomne
  - Framework ponownie wykonana nieudane zadania
- Optymalizacja lokalizacji
  - Przy BigData, przepustowość danych jest problemem
  - MapReduce + HDFS jest bardzo skuteczne
  - MapReduce pyta HDFS jaka jest lokalizacja danych wejściowych
  - Zadania Map planowane blisko wejścia, jeśli to możliwe

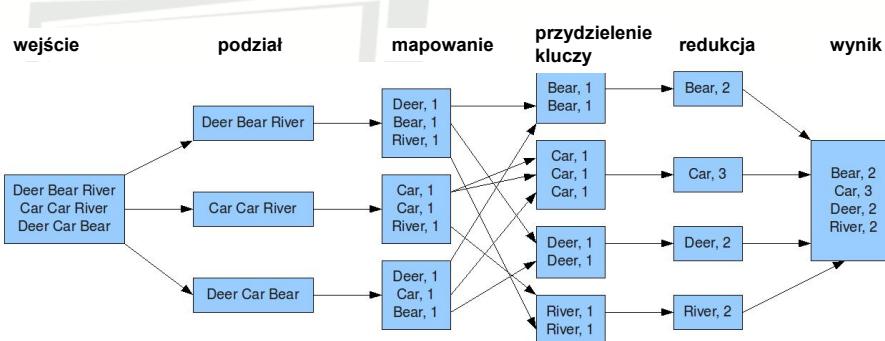
## Przykład - zliczanie słów



- Mapper
  - Wejście: wartości: wiersze tekstu wejściowego
  - Wyjście: klucz: słowo, wartość: 1
- Reducer
  - Wejście: klucz: słowo, wartość: liczba wystąpień
  - Wyjście: klucz: słowo, wartość: suma
- Uruchomienie programu
  - Definiuje zadanie
  - Wysyła zadanie do chmury

40

## Zliczanie słów – przepływ danych



41

## Zliczanie słów - Mapper



```
public static class Map extends MapReduceBase implements  
    Mapper<LongWritable,Text,Text,IntWritable> {  
    private static final IntWritable one = new IntWritable(1);  
    private Text word = new Text();  
  
    public static void map(LongWritable key, Text value,  
        OutputCollector<Text,IntWritable> output, Reporter reporter) throws  
        IOException {  
        String line = value.toString();  
        StringTokenizer tokenizer = new StringTokenizer(line);  
        while(tokenizer.hasNext()) {  
            word.set(tokenizer.nextToken());  
            output.collect(word,one);  
        }  
    }  
}
```

42

## Zliczanie słów - Reducer



```
public static class Reduce extends MapReduceBase implements  
    Reducer<Text,IntWritable,Text,IntWritable> {  
    public static void map(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));  
    }  
}
```

43

## Przykład zliczanie słów



- Zadania są kontrolowane przez konfiguracje JobConfs
- Konfiguracje JobConfs są mapami nazw atrybutów do wartości ciągów znaków
- Framework definiuje atrybuty do kontrolowania sposobu wykonania zadania
  - conf.set("mapred.job.name", "MyApp");
- Aplikacje mogą dodać dowolne wartości do JobConf
  - conf.set("my.string", "foo");
  - conf.set("my.integer", 12);
- JobConf jest dostępny dla wszystkich zadań

44

## Wszystko razem



- Opracować program dla uruchomienia aplikacji
- Program uruchomieniowy definiuje:
  - Użycie Mapera i Reducera
  - Wyjściowy klucz i typ wartości (typy wejściowe są wywnioskować z wejścia (InputFormat))
  - Lokalizacje dla wejścia i wyjścia
- Program uruchomieniowy następnie wysyła zadanie i zwykle czeka na jego zakończenie

45

## Wszystko razem



```
JobConf conf = new JobConf(WordCount.class);
conf.setJobName("wordcount");

conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Map.class);
conf.setCombinerClass(Reduce.class);
conf.setReducer(Reduce.class);

conf.setInputFormat(TextInputFormat.class);
Conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);
```

46

## Formaty wejścia i wyjścia



- Mechanizm MapReduce może zaspecyfikować jak mają być czytane dane wejściowe poprzez podanie w interfejsie InputFormat
- Mechanizm MapReduce może zaspecyfikować jak mają być zapisywane dane wyjściowe poprzez podanie w interfejsie OutputFormat
- Domyslnie interfejsy wskazują na TextInputFormat oraz TextOutputFormat, które przetwarzają wierszowe dane tekstowe
- Innym popularnym wyborem jest użycie formatów SequenceFileInputFormat oraz SequenceFileOutputFormat dla danych binarnych
- Mechanizmy te są zorientowane na pliki ale nie ma konieczności używania plików.

47

## Interfejs InputFormat



- org.apache.hadoop.mapred
  - Interface InputFormat<K,V>
- Zależne podinterfejsy:
  - ComposableInputFormat<K,V>
- Wszystkie klasy implementujące:
  - CombineTextInputFormat, CombineSequenceFileInputFormat, CompositeInputFormat, DBInputFormat, FileInputFormat, FixedLengthInputFormat, KeyValueTextInputFormat, MultiFileInputFormat, NLineInputFormat, Parser.Node, SequenceFileAsBinaryInputFormat, SequenceFileAsTextInputFormat, SequenceFileInputFilter, SequenceFileInputFormat, TextInputFormat

48

## Interfejs InputFormat



- Szkielet aplikacji Map-Reduce wykorzystuje **InputFormat** aby:
  1. Zwaliować specyfikację wejściową zadania.
  2. Podzielić pliki wejściowe na logiczne części **InputSplits**, które są przypisane do indywidualnych maperów.
  3. Zapewnić realizację implementacji **RecordReader**, aby zebrać rekordy wejściowe z **InputSplit** do przetworzenia przez **Mapper**.

49



## Interfejs InputFormat

- Rozmiar bloku danych dla klasy FileSystem jest traktowany jako góra granica dla każdej porcji wejściowej. Dolna wartość wejściowego bloku danych może być zdefiniowana przez **mapreduce.input.fileinputformat.split.minsize**
- Jasno widać, że podział danych wejściowych ze względu na rozmiar nie jest wystarczający, ponieważ granice zakresu rekordów mają być przestrzegane.
- W takich przypadkach należy zastosować klasę **RecordReader**, na której leży odpowiedzialność za przestrzeganie granic zakresu rekordów i przedstawianie zorientowanego na rekordy widoku **InputSplit** dla danego zadania.

50



## Interfejs InputFormat

Modyfikator i typ

[RecordReader<K,V>](#)

[InputSplit\[\]](#)

Metoda

[getRecordReader\(InputSplit split,](#)  
[JobConf job, Reporter reporter\)](#)  
Pobiera RecordReader dla danego  
[InputSplit.](#)

[getSplits\(JobConf job,](#)  
[int numSplits\)](#)  
Logicznie dzieli zestaw plików  
wejściowych do pracy.

51

## Mapowania i redukcje



- Mapowania
  - Zwykle tyle ile wynosi liczba przetwarzanych bloków HDFS, jest to wartość domyślna
  - lub - liczbę map można określić jako parametr
  - Liczba map może być również kontrolowana poprzez specyfikację minimalnego rozmiaru podziału.
  - Aktualne rozmiary wejść do mapera są liczone według:
    - $\max(\min(\text{block\_size}, \text{data}/\#\text{maps}), \text{min\_split\_size})$
- Redukcje
  - Dopóki ilość przetwarzanych danych jest mała
    - $0.95 * \text{num\_nodes} * \text{mapred.tasktracker.tasks.maximum}$

52

## Użyteczne narzędzia



- Partitioners - partycjonery
- Combiners - sumatory
- Compression - kompresja
- Counters - liczniki
- Speculation – spekulatywność
- Zero Reduces – redukcja zer
- Distributed File Cache – rozproszony bufor plików
- Tool – klasa Tool (narzędzie)

53

## Partitioners - partycjonery

- Partycjonery to aplikacje, które definiują jak klucze są przypisane do reduktorów
- Domyslnie partycjonowanie rozprzestrzenia klucze równomiernie, ale losowo
  - `Uses key.hashCode() % num_reduces`
- Niestandardowe partycjonowanie jest często konieczne, na przykład w celu określenia całkowitej wartości wyjścia
  - powinno implementować interfejs Partycjonera
  - być ustawiane przez  
`conf.setPartitionerClass (MyPart.class)`
  - aby uzyskać całkowity porządek, powinno spróbować klucze wyjściowe mapowania i wybrać wartości tak aby podzielić klucze w przybliżeniu na równe porcje i użyć ich w partycjonerze

54

## Combiners - Sumatory

- Kiedy proces mapowania daje wiele powtarzalnych kluczy to:
  - dobrze jest przeprowadzić lokalną agregację uzależnioną od mapy
  - wykonać proces przez określonego Sumatora
  - celem jest zmniejszenie ilości danych przejściowych
  - sumatory mają ten sam interfejs co Reduktory i są często tą samą klasą
  - sumatory nie mogą dawać wyników cząstkowych ponieważ są wielokrotnie pośrednio uruchamiane
  - W przykładzie WordCount:
    - `conf.setCombinerClass (Reduce.class) ;`

55

## Kompresja wyjścia



- Kompresja wyjścia i danych pośrednich często przynieść ogromne zyski wydajności
  - może być zaspecyfikowana przez plik konfiguracyjny lub ustawiona w programie
  - Należy ustawić parametr `mapred.output.compress=true` aby skompresować wyjście zadania
  - Należy ustawić `mapred.compress.map.output=true` aby skompresować wyjście mapera

56

## Kompresja wyjścia



- Typy kompresji  
**mapred(.map)??.output.compression.type**
  - „block” - grupa kluczy i wartości jest kompresowana razem
  - „record” – każda wartość jest kompresowana indywidualnie
  - Zazwyczaj najlepsza jest kompresja bloków
- Kompresja – typy algorytmów kompresujących  
**mapred(.map)??.output.compression.codec**
  - domyślnie (zlib) – wolniejsze, ale lepsza kompresja
  - LZO - szybsze, gorsza kompresja

57

## Counters - liczniki



- Często aplikacje MapReduce mają policzalne zdarzenia
- Na przykład środowisko zlicza rekordy wejściowe i wyjściowe Mapera i Sumatora
- Aby zdefiniować liczniki użytkownika:
  - `static enum Counter {EVENT1, EVENT2};`
  - `reporter.incrCounter(Counter.EVENT1, 1);`
- Definicja przyjaznych nazw w pliku  
**MyClass\_Counter.properties**

```
CounterGroupName=MyCounters
EVENT1.name=Event 1
EVENT2.name=Event 2
```

58

## Spekulatywne wykonanie



- Zdolność mikroprocesorów przetwarzających instrukcje potokowo do wykonywania instrukcji znajdujących się za skokiem warunkowym, co do którego jeszcze nie wiadomo, czy nastąpi.
- Ostatecznie wyniki wyliczone z wyprzedzeniem zostaną albo uwzględnione, albo odrzucone, zależnie od tego, czy skok się wykona.

59

## Spekulatywne wykonanie



- Mechanizm redukcji opóźnień jest heurystyczny.
- Procesor próbuje „zgadnąć” przebieg wykonania programu pobiera i wykonuje kolejne rozkazy.
- Procesor nie wie, czy „zgadywanie” dało właściwy efekt aż do czasu wykonania instrukcji skoku.

60

## Wykonanie spekulatywne - MapReduce



- Szkielet programowy MapReduce może uruchomić wiele instancji wolnych zadań
  - Są użyte wyniki otrzymane od instancji najwcześniej zwracającej dane
  - Kontrola odbywa się przez zmienną **mapred.speculative.execution**
  - Może dramatycznie zwiększyć opóźnienia w wykonaniu się zadań chmurowych

61

## Redukcja zer



- Często istnieje potrzeba uruchomienia filtru dla danych wejściowych
  - Tam gdzie nie wymagane jest sortowanie lub tasowanie wymagane przez zadanie
  - Ustawienie liczby reduktorów równej 0
  - Wyniki wyjściowe wędrują od razu do wyjścia OutputFormat na dysku

62

## Rozproszony bufor plików



- Czasami zachodzi potrzeba wykonać kopię danych na lokalnym komputerze
  - Pobranie 1GB danych dla każdego Mapera jest kosztowne
- W takim przypadku należy zdefiniować listę plików do pobrania w pliku konfiguracyjnym JobConf
- Pliki są pobierane raz na każdym komputerze
- Należy dodać do programu uruchomieniowego:
  - `DistributedCache.addCacheFile(new URI("hdfs://adres:8020/plik"), conf);`
- Do zadania dodać:
  - `Path[] files = DistributedCache.getLocalCacheFiles(conf);`

63

## Klasa Tool



- Wspiera obsługę podstawowych opcji wiersza poleceń

- **-conf plik1** – załadowuje konfiguracje z pliku o nazwie plik1
  - **-D prop=value** - definiuje pojedynczy parametr i przypisuje mu wartość

- Struktura klasy:

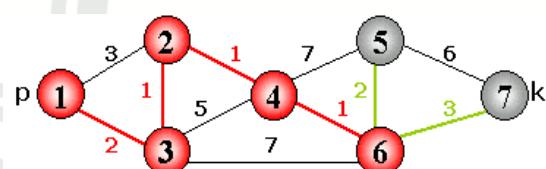
```
public class MyApp extends Configured implements Tool {  
    public static void main(String[] args) throws Exception {  
        System.exit(ToolRunner.run(new Configuration(),  
            new MyApp(), args));  
    }  
    public int run(String[] args) throws Exception {  
        .... getConf() ....  
    }  
}
```

64

## Znalezienie najkrótszej ścieżki



- Popularnym zadaniem jest znalezienie drogi od węzła startowego do jednego lub więcej węzłów docelowych
- Na pojedynczej maszynie wykorzystany do rozwiązania jest algorytm Dijkstry
- Czy można użyć przeszukiwania wszerz (breadth-first search) aby znaleźć najkrótszą ścieżkę z wykorzystaniem MapReduce?



65



## Przeszukiwanie wszerz



- Przechodzenie grafu rozpoczyna się od zadanego wierzchołka  $s$  i polega na odwiedzeniu wszystkich osiągalnych z niego wierzchołków.
- Wynikiem działania algorytmu jest drzewo przeszukiwania wszerz o korzeniu w  $s$ , zawierające wszystkie wierzchołki osiągalne z  $s$ .
- Do każdego z tych wierzchołków prowadzi dokładnie jedna ścieżka z  $s$ , która jest jednocześnie najkrótszą ścieżką w grafie wejściowym.

66

## Znalezienie najkrótszej ścieżki: rozwiązańie intuicyjne



- Możemy zdefiniować rozwiązanie tego problemu indukcyjnie
  - DystansDo(węzełStarowy) = 0
  - Dla wszystkich  $n$  węzłów bezpośrednio dostępnych z węzłaStartowego  $DystansDo(n) = 1$
  - Dla wszystkich  $n$  węzłów dostępnych z innego zestawu węzłów  $S$  mamy:
    - $DystansDo(n) = 1 + \min(DystansDo(m), m \in S)$

67

## Od intuicji do algorytmu



- Zadanie mapowania otrzymuje numer węzła  $n$  jako klucz, a  $(D, \text{punkty-do})$  jako jego wartości
  - $D$  to dystans do węzła z punktu startowego
  - $\text{punkty-do}$  jest to lista węzłów osiągalnych z  $n$
- $\forall p \in \text{punkty-do}, \text{emituj } (p, D+1)$   
(dla każdego  $p$  należącego do  $\text{punkty-do}$ )
- Zadania redukcji gromadzą możliwe odległości dla danego  $p$  i wybierają najmniejszą

68

## Rozwiązanie



- Zadanie MapReduce może osiągnąć granicę grafu w jednym przeskoku
- Aby przeprowadzić całe przeszukiwanie wszerz komponent nie będący w MapReduce przekazuje wyjście tego kroku do zadania MapReduce do kolejnej iteracji
  - Problem: gdzie podziała się lista  $\text{punkty-do}$ ?
  - rozwiązań: mapper emituje również  $(n, \text{punkty-do})$

69

## Zakończenie wykonania



- Algorytm ten staruje z jednego węzła grafu
- Kolejne iteracje zawierają wiele więcej węzłów grafu jako dane wejściowe
- Czy algorytm się kiedyś kończy?
  - Tak, ostatecznie trasy między węzłami nie będą odkrywane i nie zostanie znaleziona lepsza odległość. Gdy odległość jest taka sama, możemy przestać.
  - Mapper powinien emitować  $(n, D)$  aby się upewnić, że bieżący dystans jest przekazywany do reducera

70