MapReduce

Janina Mincer-Daszkiewicz

Systemy rozproszone

MSUI, II rok

Materiały i rysunki zaczerpnięto z następujących źródeł

1. Jeffrey Dean,Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, firma Google, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004

   http://students.mimuw.edu.pl/SR-MSUI/03-MapReduce/mapreduce-osdi04.pdf

2. Google Code University,Distributed Systems

   http://code.google.com/edu/parallel/index.html

3. Christophe Bisciglia, Aaron Kimball, & Sierra Michels-Slettvet, MapReduce Theory and Implementation, Distributed Computing Seminar, Google, 2007

MapReduce z lotu ptaka ...

- Produkt firmy Google™
- Model programowania i implementująca go biblioteka w C++ do przetwarzania dużych zbiorów danych
- Programy napisane zgodnie z tym modelem są automatycznie zrównoleglane i wykonywane na klastrze zbudowanym ze standardowych komputerów
- Zdejmuje z programisty konieczność zajmowania się szczegółami wykonania programu w środowisku rozproszonym
- Wiele aplikacji wykonywanych rutynowo w Google wykorzystuje ten model

Cele projektowe

- W Google rośnie liczba obliczeń przetwarzających olbrzymie ilości danych (odwiedzane strony webowe, logi serwerów WWW itp.)
- Większość tych obliczeń jest prosta koncepcyjnie
- Ze względów wydajnościowych powinno się je wykonywać w środowisku rozproszonym, ale to znacząco komplikuje kod:
  - jak zrównoleglić wykonanie na wiele komputerów
  - jak rozproszyć dane wejściowe
  - jak radzić sobie z awariami maszyn
  - jak zebrać i przekazać użytkownikowi wyniki obliczeń
  - jak optymalnie wykorzystać przepustowość sieci

Cele projektowe – cd

- Inspiracją dla nowego modelu programowania były obecne w językach funkcyjnych operacje map i reduce
- Na każdym logicznym rekordzie danych wykonuje się map, otrzymuje się pośrednie wyniki w postaci par (klucz, wartość), następnie wykonuje się reduce na wszystkich wartościach o tym samym kluczu
- Programista pisze program w kategoriach operacji map i reduce, wykonaniem programu w środowisku rozproszonym zarządza dołączana do niego biblioteka
- Biblioteka zapewnia także odporność na błędy, ponawiając wykonanie obliczeń w przypadku awarii itp.

Paradygmat programowania funkcyjnego

- Operatory funkcyjne nigdy nie modyfikują oryginalnych struktur danych, zawsze tworzą nowe
- Oryginalne dane pozostają w niezmienionej postaci
- Porządek operacji nie wpływa na końcowy wynik
- Kod programu nie determinuje jawnie przepływu danych
- Jeśli porządek aplikowania funkcji f do elementów listy jest przemienny, to można dowolnie zmieniać kolejność lub zrównoleglać wykonanie

Model programowania

- map
  - dostarcza ją programista
  - pobiera rekord z wejścia i produkuje pośrednie wyniki w postaci par (klucz, wartość)
  - biblioteka grupuje wszystkie wartości pośrednie z tym samym kluczem i przekazuje do funkcji reduce
- reduce
  - dostarcza ją programista
  - pobiera zbiór wartości dla tego samego klucza i łączy je tworząc mniejszy zbiór
  - wartości pośrednie są dostarczane przez iterator

Przykład

- Zliczanie liczby wystąpień każdego słowa w dużym zbiorze dokumentów

```
map (String key, String value):
// key: nazwa dokumentu
// value: zawartość dokumentu
for each word w in value:
    EmitIntermediate(w, „1”);
```

- map emituje każde słowo z licznikiem częstości

Przykład

```
reduce (String key, Iterator values):
 // key: słowo
 // values: lista liczników
 for each v in values:
         result += ParseInt(v);
 Emit (AsString(result));
```

- reduce sumuje liczniki dla konkretnego słowa
- Dodatkowo programista pisze kod, który wypełnia specification object nazwami plików wejściowych i wyjściowych oraz dodatkowymi parametrami
- Wywołuje funkcję MapReduce przekazując jej ten obiekt

Więcej przykładów

- **Rozproszony grep**: map emituje wiersz, jeśli pasuje on do wzorca, reduce jest funkcją identycznościową
- **Licznik częstości odwołań do danego URL**: map przetwarza logi z odwołaniami do stron webowych i emituje (URL, 1), reduce dodaje wartości dla tego samego URL i emituje

  (URL, licznik)
- **Graf odwrotnych odsyłaczy**: map emituje pary (target, source) dla każdego odnośnika do URL target znalezionego w stronie source, reduce konkatenuje listy i emituje (target, lista(source))
- **Odwrotny indeks**: map parsuje każdy dokument i emituje ciąg (słowo, id dokumentu), reduce akceptuje wszystkie pary dla danego słowa i emituje

  (słowo, lista(id dokumentu))

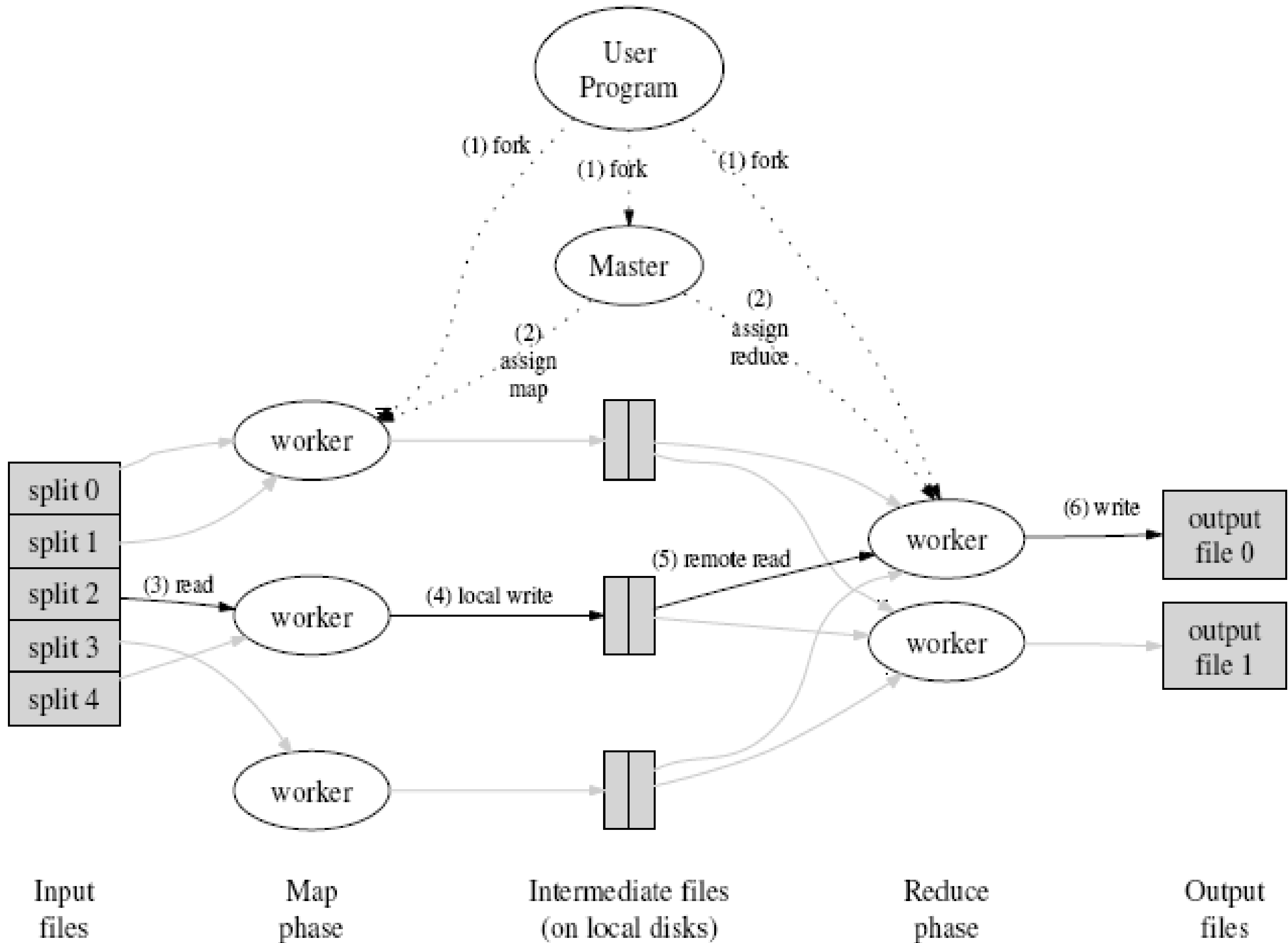
Implementacja

- Implementacja zależy od środowiska
- Tutaj opisano implementację dla środowiska używanego w Google: duże klastry standardowych PC połączonych Ethernetem:
  - maszyny to zwykle dwuprocesorowe x86 z Linuksem
  - 2-4 GB RAM
  - sieć zwykle 100 Mb/s lub 1 Gb/s
  - klaster zawiera setki lub tysiące maszyn, częste awarie
  - dyski IDE, rozproszony system plików Google FS
  - użytkownicy dostarczają zadania planiście, każde obejmuje wiele zadań, planista wybiera maszynę

Wykonanie

- Wywołania map są rozproszone między wieloma maszynami, dane wejściowe są dzielone automatycznie na M podzbiorów, każdy może być przetwarzany równolegle przez różne maszyny
- Wywołania reduce są rozproszone przez podział przestrzeni kluczy pośrednich na R części przy użyciu funkcji podziału, np. hash(key) mod R
- Liczba podziałów i funkcja podziału są dostarczane przez użytkownika

Implementacja cd



Implementacja – wywołanie MapReduce

- MapReduce dzieli pliki wejściowe na M części po 16-64 MB. Następnie uruchamia wiele kopii programu na klastrze
- Jedna kopia jest wyróżniona – to **zarządca**. Pozostałe to **pracownicy**, którzy dostają zadania od zarządcy. Jest M zadań map i R zadań reduce. Zarządca wybiera bezczynnych pracowników i przydziela każdemu zadanie map lub reduce.
- Pracownik wykonujący map czyta zawartość opowiadającej mu części, parsuje pary (klucz, wartość) i przekazuje je do funkcji map

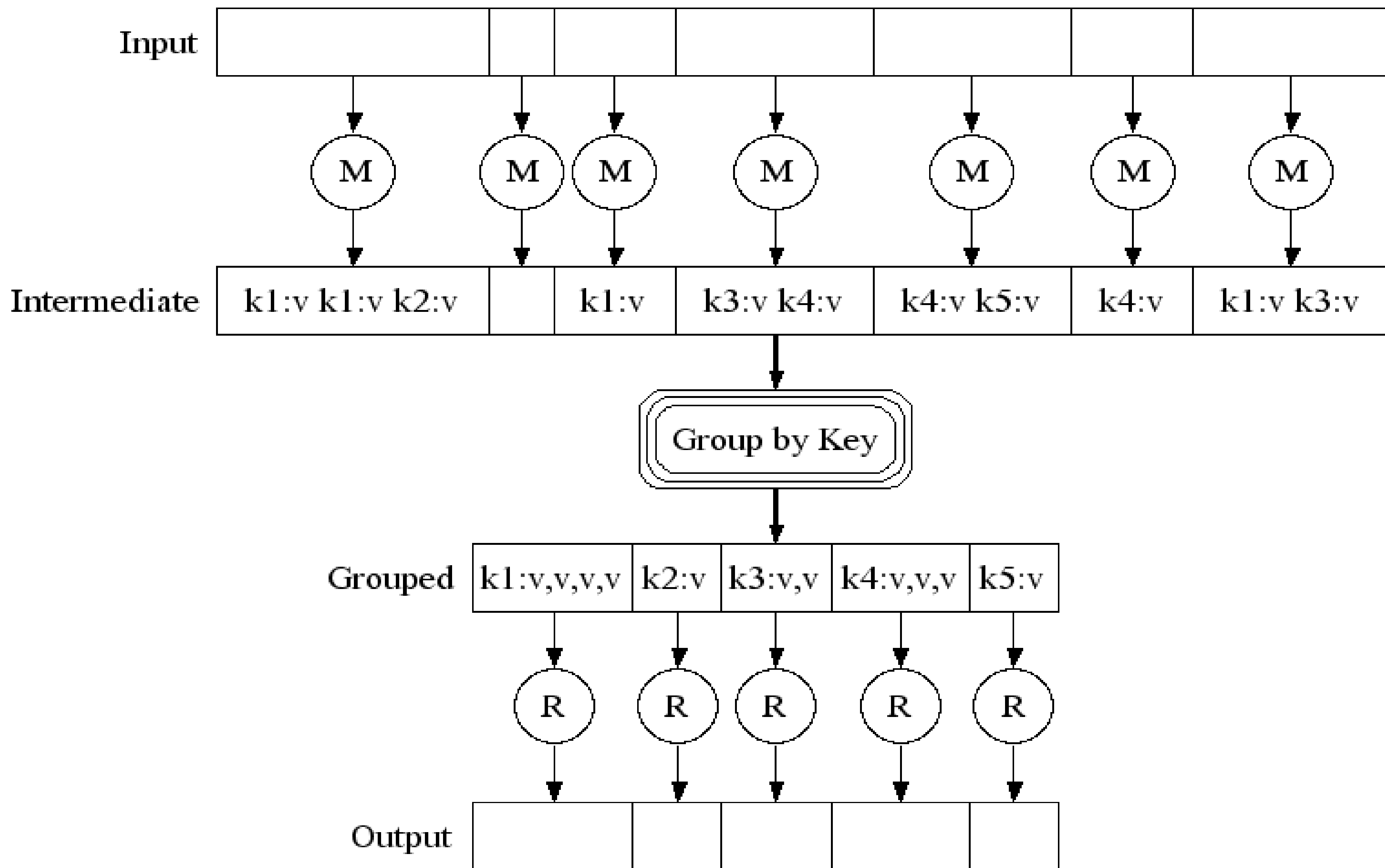
Implementacja – cd

- Pośrednie pary (klucz, wartość) produkowane przez funkcję map są buforowane w pamięci. Cyklicznie są zapisywane na lokalny dysk, w podziale na R części. Położenie tych par na lokalnym dysku jest przekazywane do zarządcy, które przekazuje je dalej do pracowników wykonujących reduce
- Pracownik wykonujący reduce używa RPC do czytania danych, sortuje je, grupując razem wystąpienia tego samego klucza, iteruje po posortowanych danych i dla każdej unikatowej wartości klucza przekazuje ją i zbiór odpowiadających jej wartości do funkcji reduce

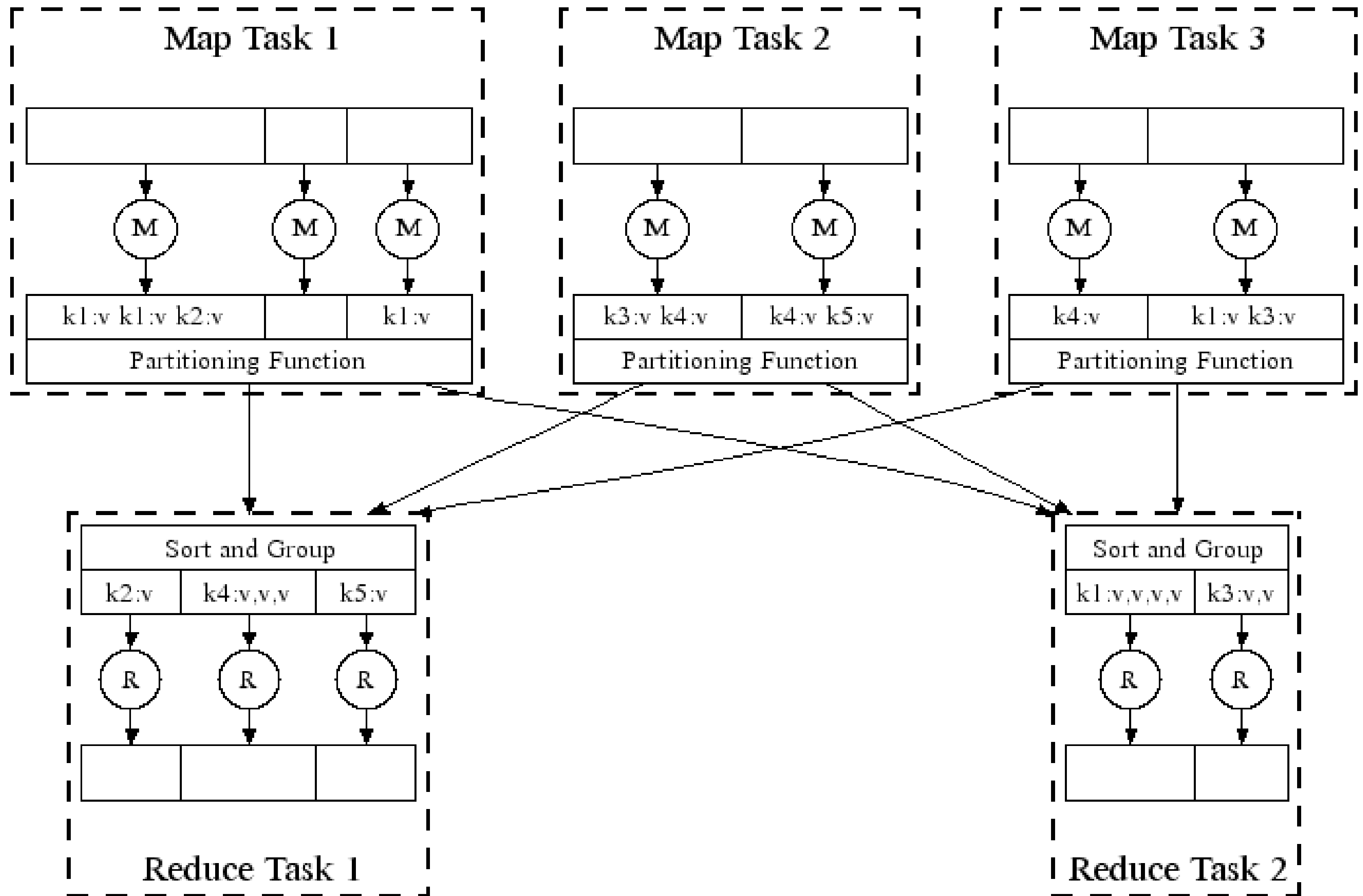
Implementacja – cd

- Wynik funkcji reduce jest dołączany na koniec odpowiedniego pliku wyjściowego
- Gdy wszystkie zadania map i reduce zakończą pracę, zarządca budzi proces użytkownika. W tym momencie następuje powrót z wywołania MapReduce
- Wynik jest dostępny w R plikach wyjściowych

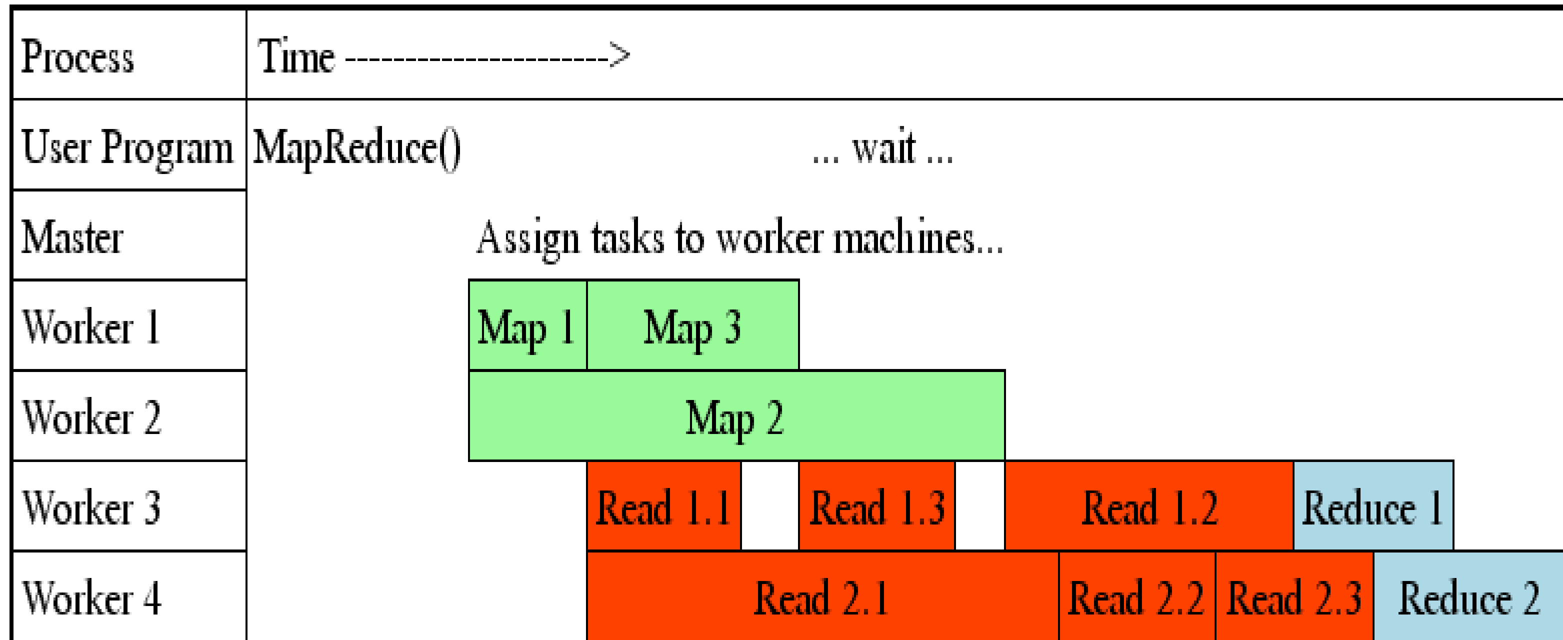
Wykonanie



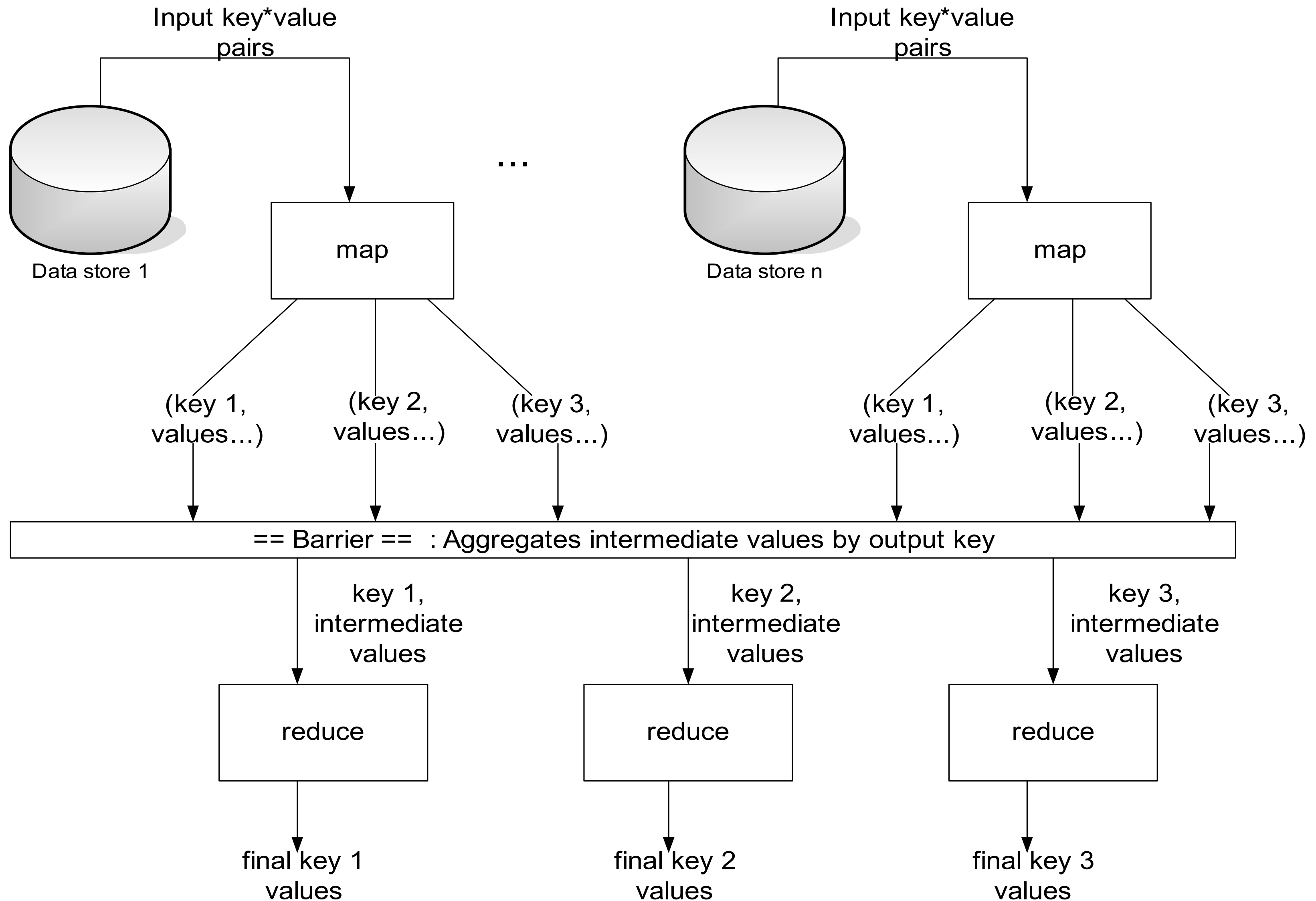
Wykonanie



Wykonanie



Wykonanie - cd



Równoległość

- Funkcje map wykonują się równolegle, tworząc różne wartości pośrednie z różnych wejściowych zbiorów danych
- Funkcje reduce także wykonują się równolegle, każda przetwarza różny klucz
- Wszystkie wartości są przetwarzane niezależnie
- Wąskie gardło: faza reduce nie może się zacząć nim nie zakończy się faza map

Struktury danych zarządcy

- Dla każdego zadania map i reduce zarządca przechowuje stan (idle, in-progress, completed) i tożsamość maszyny wykonującej zadanie
- Dla każdego zakończonego zadania map zarządca przechowuje położenie i rozmiary R wytworzonych plików pośrednich. Przekazuje tę informację dalej do pracowników wykonujących zadania reduce

Odporność na błędy

- Ponieważ zadania wykonują się na tysiącach standardowych komputerów, biblioteka musi zapewnić odporność na błędy
- Zarządca cyklicznie wysyła ping do pracowników
- Zakończone zadania map w sytuacji błędu są wyznaczane do ponownego wykonania, gdyż ich wynik jest zapisany na lokalnych dyskach, czyli jest niedostępny
- Zakończone zadania reduce nie muszą być wykonywane ponownie, gdyż ich wynik jest zapisany w globalnym systemie plików

Odporność na błędy – zarządca

- Zarządca cyklicznie mógłby zapisywać punkty kontrolne swoich struktur danych
- W przypadku awarii zarządcy można by wystartować nową kopię od ostatniego punktu kontrolnego
- W obecnej implementacji przerywa się wykonanie MapReduce
- Klient w razie potrzeby wznawia wykonanie MapReduce

Szerokość pasma sieci

- Szerokość pasma sieci jest kluczowym zasobem
- Dane wejściowe są przechowywane na lokalnych dyskach w systemie plików Google FS, który dzieli każdy plik na kawałki po 64 MB i każdy blok zapisuje w kilku kopiach na różnych maszynach
- Zarządca próbuje uruchomić zadanie map na maszynie zawierającej kopię danych wejściowych (lub przynajmniej w pobliżu)
- Dzięki temu większość danych wejściowych jest czytana z lokalnych plików i nie obciąża sieci

Dodatkowe zadania

- O całkowitym czasie wykonania operacji MapReduce w znacznym stopniu decyduje **maruder** – maszyna, która wykonuje nienaturalnie długo jedno z ostatnich zadań map lub reduce
- Maruderem może być maszyna ze złym dyskiem, obciążona maszyna lub maszyna niesprawna
- Kiedy wykonanie MapReduce zbliża się do końca, zarządca zleca dodatkowe wykonanie pozostałych, jeszcze nie zakończonych zadań. Zadanie oznacza się jako zakończone, gdy zakończy się wykonanie jednego ze zleconych wykonań

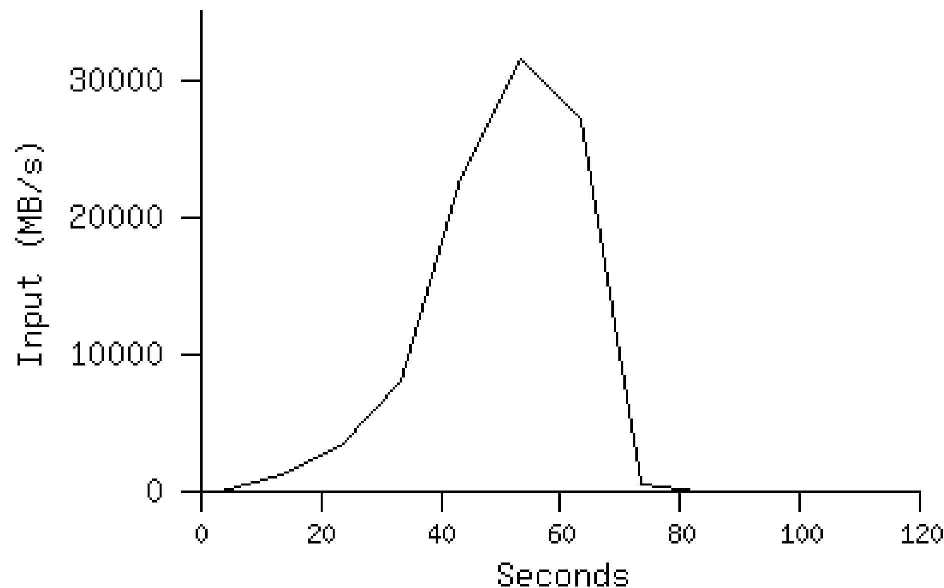
Udoskonalenia

- Funkcja podziału. Domyślna funkcja podziału ma postać hash(key) mod R. Czasami zasadne jest użycie innej funkcji, np. gdy chcemy, żeby kluczami były URL-e i chcemy, żeby wszystkie wyniki dla jednej maszyny trafiły do tego samego pliku wyjściowego. Dlatego użytkownik może dostarczyć własną funkcję podziału, np. hash(Hostname(urlkey)) mod R.
- Funkcja łącząca. Użytkownik może dostarczyć funkcję łączącą, która połączy wyniki (np. <the, 1>) przed przesłaniem ich siecią (combiner == mini-reducer)
- Pomijanie błędnych rekordów. Opcjonalnie MapReduce może pomijać rekordy, które powodują deterministyczne awarie.

Wydajność

- Wydajność zmierzono dla dwóch obliczeń:
  - szukanie wzorca w dużym zbiorze (10^{10} 100-bajtowych rekordów, czyli terabajty danych)
  - sortowanie dużego zbioru (tyle samo danych)
- Konfiguracja klastra
  - 1800 maszyn, 2 GHz Intel Xeon, 4 GB RAM, dwa dyski IDE po 160 GB, 1 Gb Ethernet
  - programy wykonywane w weekend, gdy maszyny były w większości bezczynne

Grep – tempo transmisji w czasie

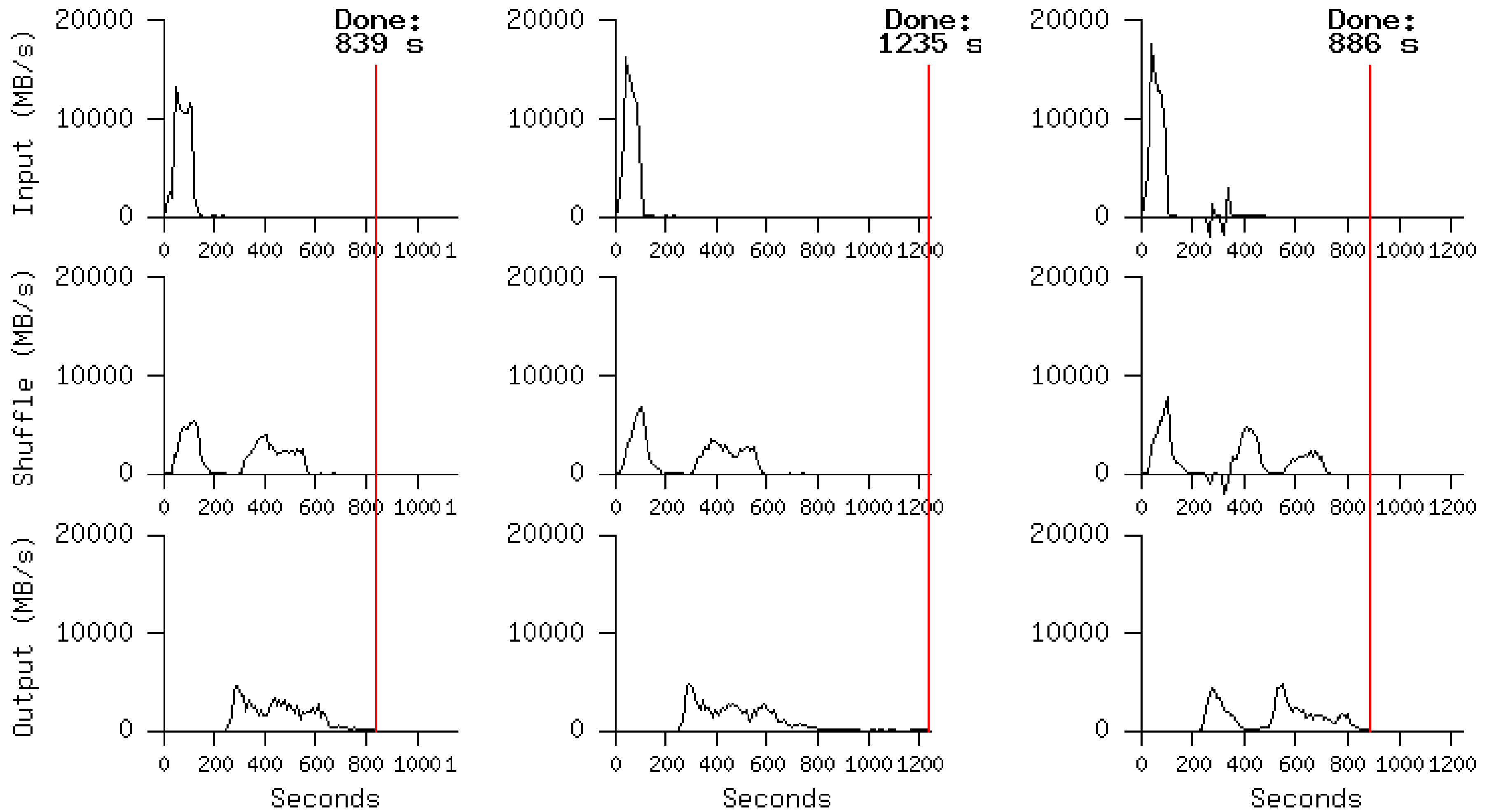


•Tempo przeglądania danych wejściowych rośnie wraz z dodawaniem kolejnych maszyn do wykonania MapReduce

•Max tempo to ok. 30 GB/s dla 1764 przydzielonych pracowników

•Całość zajmuje ok. 150 s (z czego ponad 1 minutę zajmuje narzut – propagacja programu na maszyny pracowników i opóźnienia związane z otwarciem przez GFS 1000 plików wejściowych)

Sortowanie

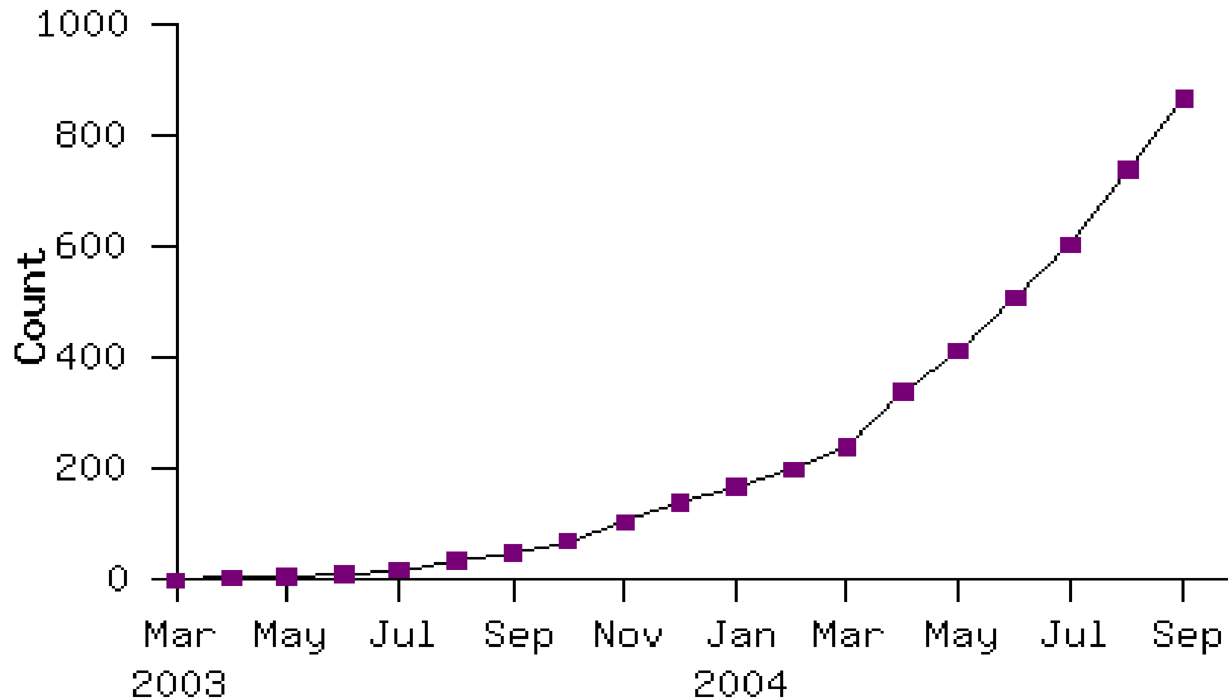


(a) normalne wykonanie (b) bez dodatkowych zadań (c) zabito 200 z 1764 zadań

Sortowanie – cd.

- Dodatkowe zadania znacznie przyspieszają wykonanie obliczenia
- System dobrze daje sobie radę z awariami

Zastosowanie modelu



- Liczba aplikacji w Google korzystających z modelu rośnie znacząco

Hadoop

- Hadoop jest projektem typu open source rozwijanym w ramach Apache Software Foundation
- Hadoop implementuje MapReduce, przy użyciu Hadoop Distributed File System (*HDFS*)
- Hadoop był używany na klastrach złożonych z 2000 węzłów
- Odnośniki: http://hadoop.apache.org/core

Podsumowanie

- MapReduce jest wygodną abstrakcją programowania równoległego
- Istotnie uprościło obliczenia dużej skali wykonywane w Google'u, na klastrze standardowych maszyn z systemem plików Google FS
- Pozwala skoncentrować się na problemie, pozostawiając bibliotece techniczne szczegóły implementacji