Programiranje oblaka i softverska okruženja, Deo 1

- MapReduce
- Hadoop
- Pig, Hive
- AWS Elastic MapReduce

Šta je MapReduce?

- Jednostavan model programiranja sa paralelnom obradom podataka (data-parallel)
- Za obradu podataka velike skale:
  - Koristi veliki skup običnih računara
  - Izvršava procese na distribuiran način
  - Nudi visoku raspoloživost
- Razvijen u Google:
  - Obrađuje 20 peta bajta podataka svaki dan
- Popularizovan u projektu Hadoop (open src):
  - Koristi se u Yahoo!, Facebook, Amazon, ...

Za šta se koristi MapReduce?

- U Google:
  - Konstrukcija indeksa (Index) za Google Search
  - Klasterizacija članaka za Google News
  - Statističko mašinsko prevođenje
- U Yahoo!:
  - Pravljenje "Web map" za Yahoo! Search
  - Detekcija spam emaila za Yahoo! Mail
- U Facebook:
  - Pretraga podataka (Data mining)
  - Ad tekst optimizaciju (ATO)
  - Detekcija spam emaila

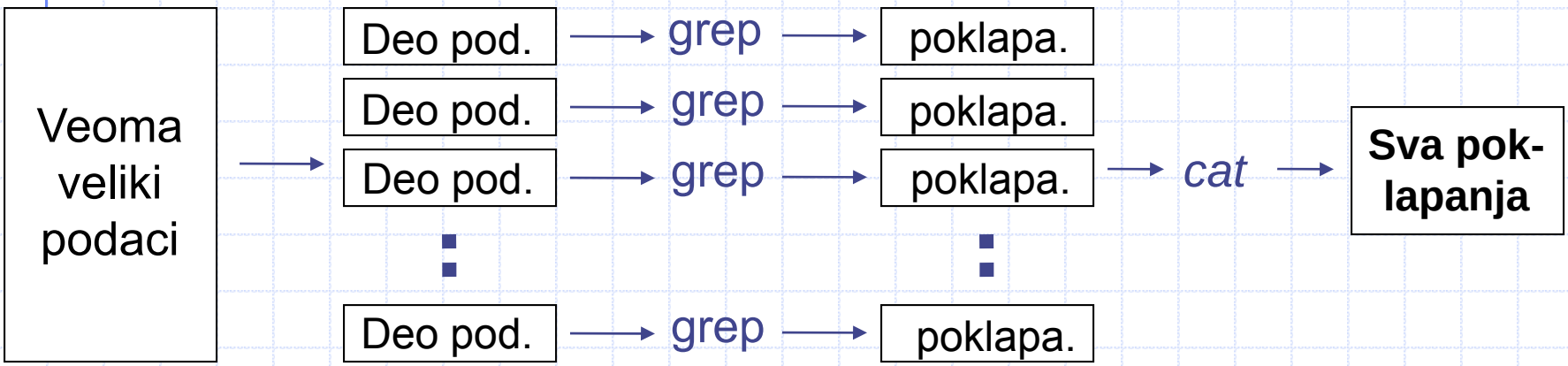
Motivacija: Obrada podataka velike skale

- Mnogi zadaci su komponovani od obrade puno pod. da bi se proizvelo puno drugih pod.
- Želja da se koristi na stotine hiljada CPU-a
  ... ali to treba da bude jednostavno!
- MapReduce obezbeđuje:
  - Korisnički-definisane funkcije
  - Automatsku paralelizaciju i distribuciju
  - Otpornost na otkaze
  - Raspoređivanje U-I
  - Stanje (status) i nadzor

Za šta se koristi MapReduce?

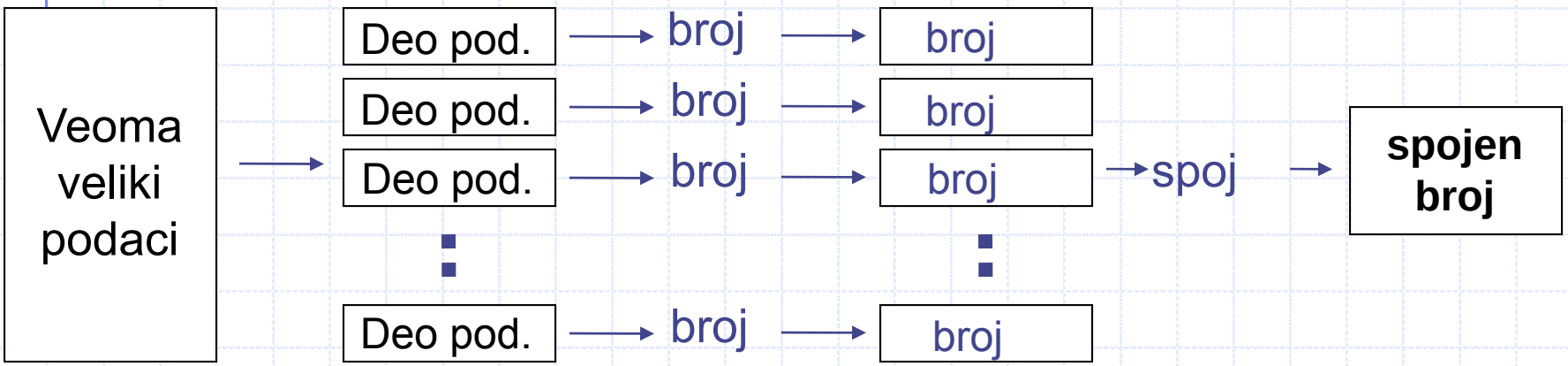
- U istraživanjima:
  - Analiza astronomskih slika (Washington)
  - Bioinformatika (Maryland)
  - Analiza konflikata u Wikipedia (PARC)
  - Obrada prirodnih jezika (CMU)
  - Fizika čestica (Nebraska)
  - Simulacija klime okeana (Washington)

Distribuirani Grep

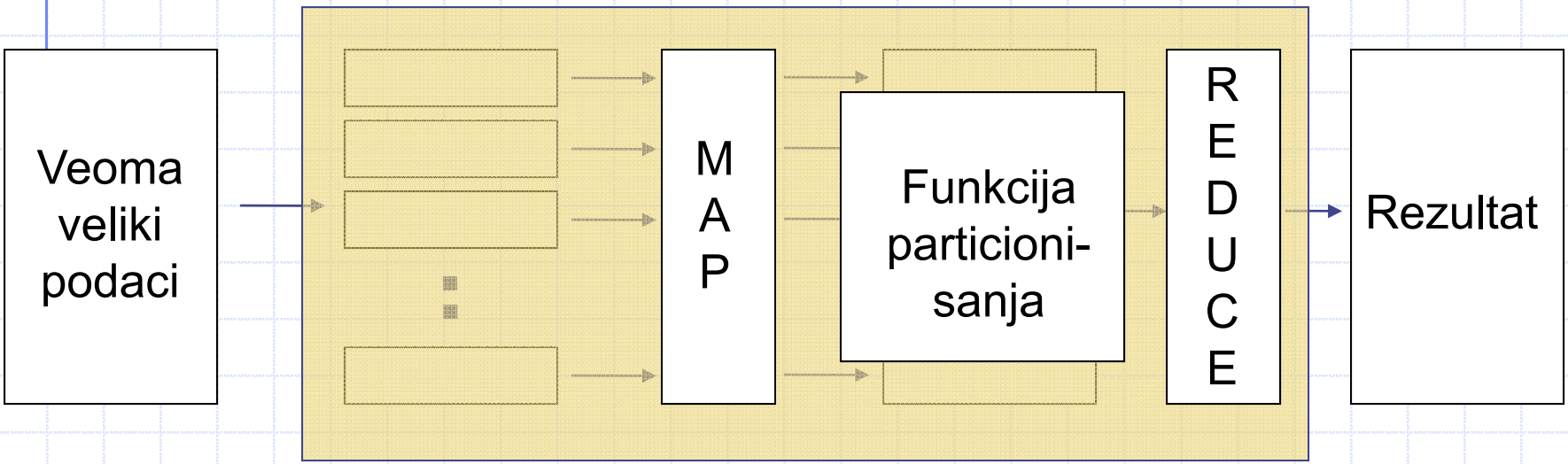


- **grep** je uslužna komanda za pretraživanje skupova pod. običnog teksta radi pronalaženja linija koje imaju poklapanja sa regularnim izrazima
- **cat** je standardna Unix komanda koja spaja i lista (prikazuje) datoteke

Distribuirano brojanje reči

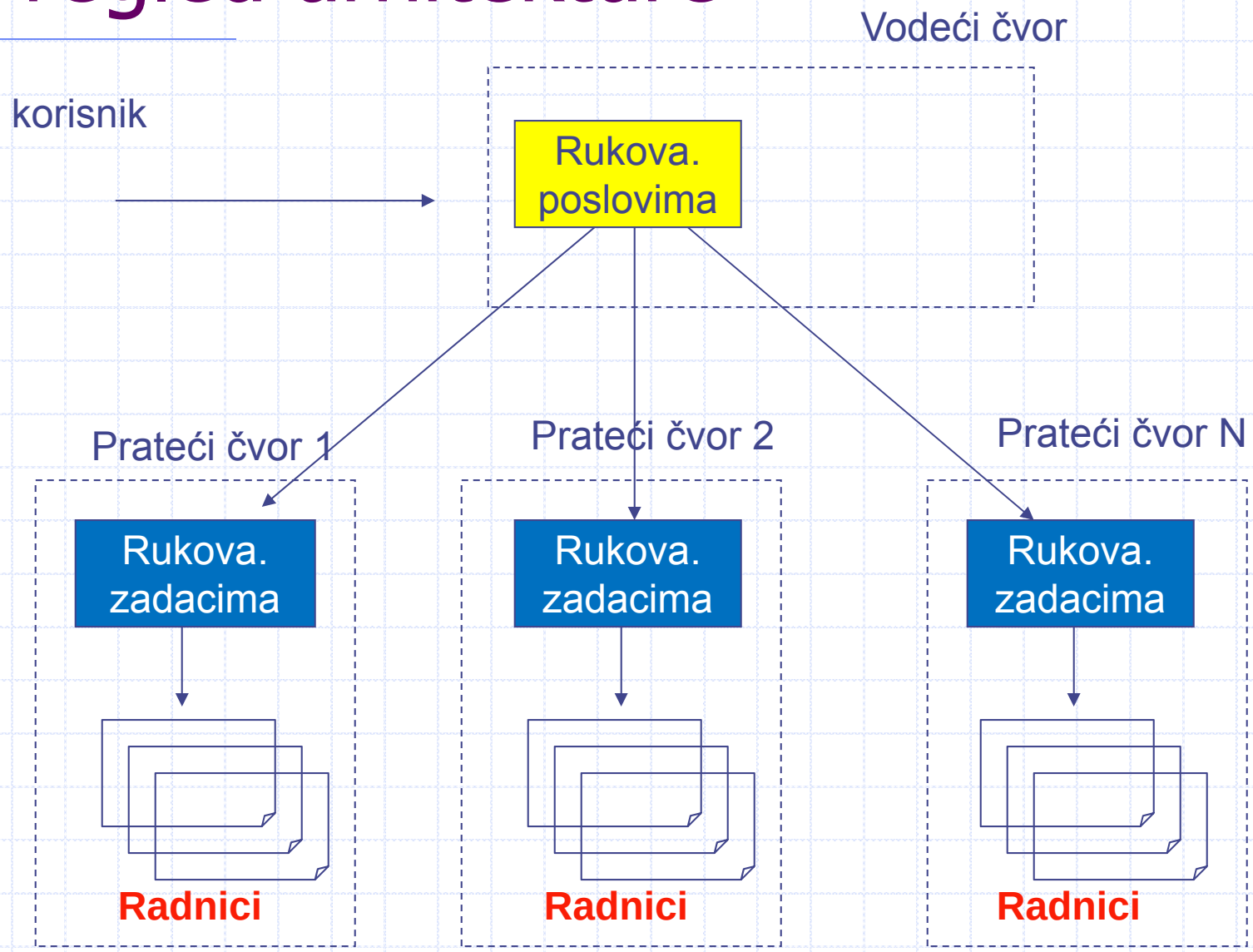


Map+Reduce



- **Map**:
  - Prihvata *ulazne* parove ključ/vrednost
  - Emituje *među* parove ključ/vrednost
- **Reduce**:
  - Prihvata *među* parove ključ/vrednost
  - Emituje *izlazne* parove ključ/vrednost

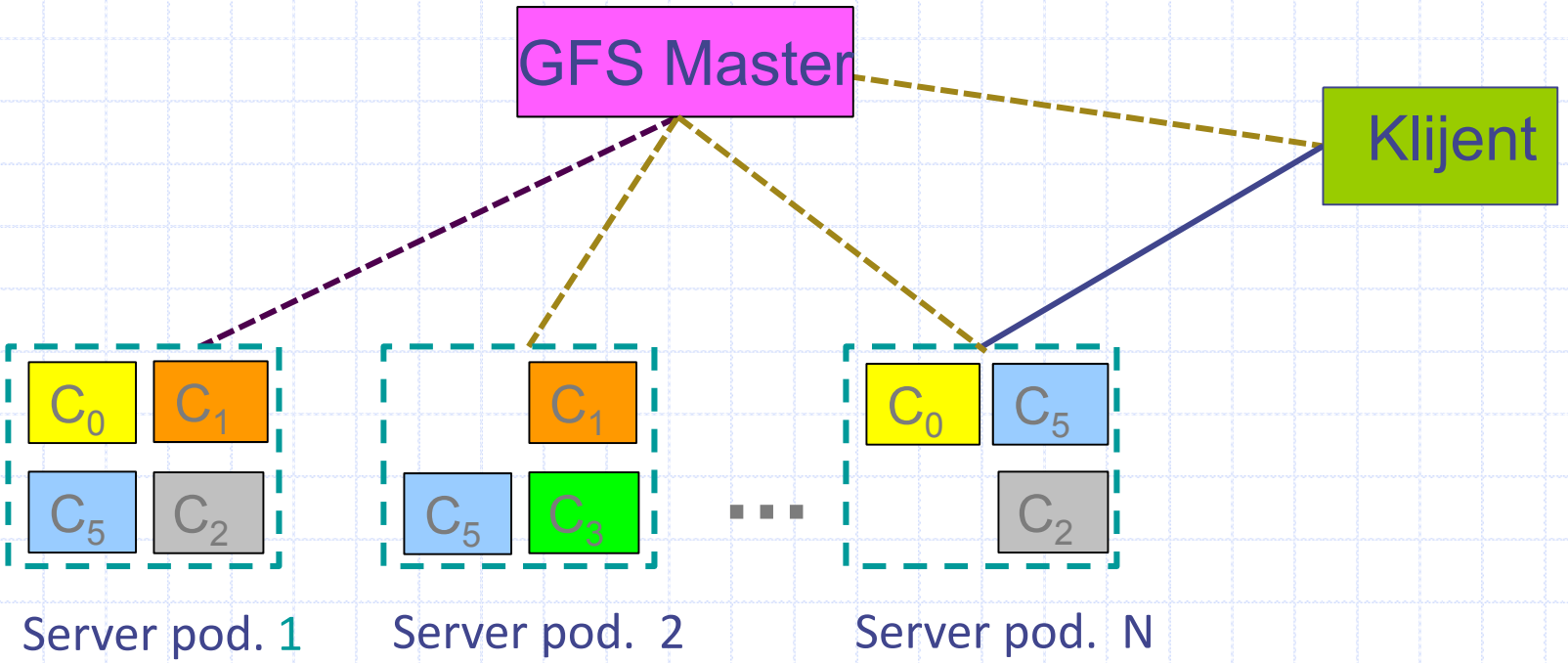
Pregled arhitekture



GFS: pozadinski sistem skladišta

- Cilj:
  - Globalni pogled
  - Raspoloživost ogromnih datoteka u prisustvu otkaza čvorova
- Vodeći čvor (meta server):
  - Centralizovan, indeksira sve blokove (chunks) na serverima podataka
- Server blokova (server podataka):
  - Dat je podeljena u susendne blokove, obično 16-64MB
  - Svaki blok je repliciran (obično 2x ili 3x)
  - Nastoji se da se replike drže u različitim stalcima (racks)

GFS arhitektura



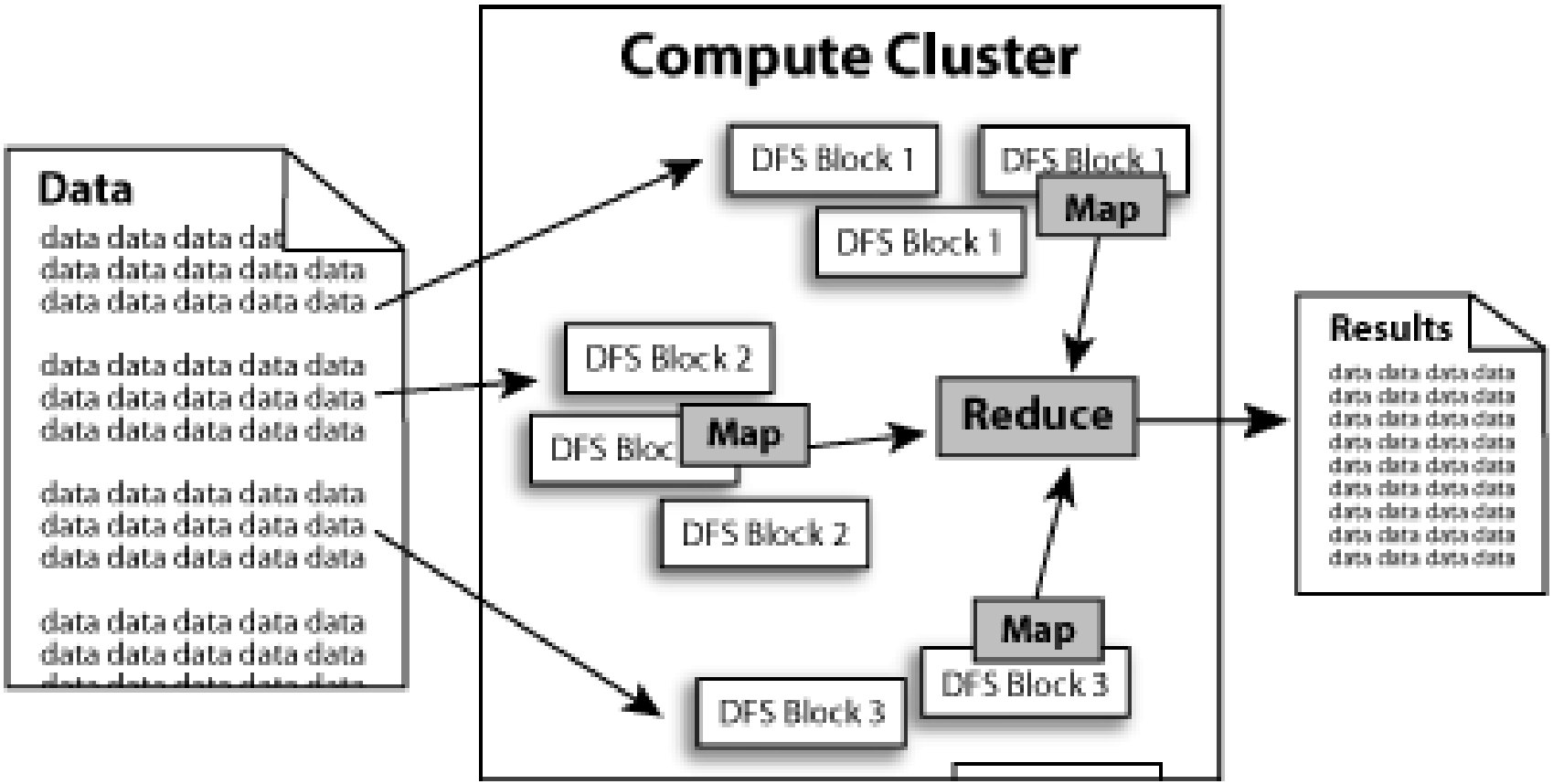
Funkcije u modelu

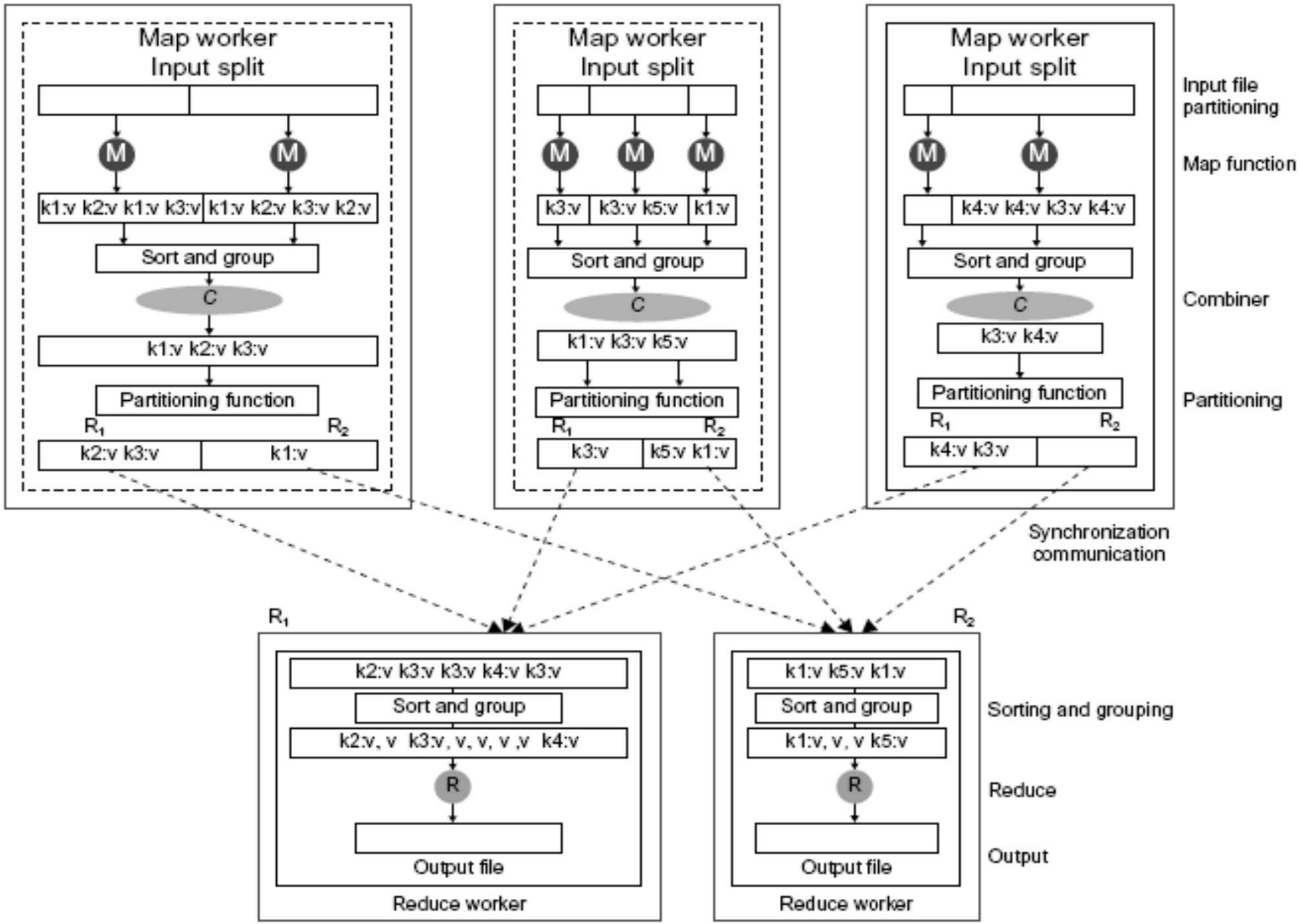
- Mapiranje (Map):
  - Obradi par ključ/vrednost radi generisanja među parova ključ/vrednost
- Redukcija (Reduce):
  - Spoj sve među vrednosti povezane sa istim ključem
- Particionisanje (Partition)
  - Podrazumevano: hash(key) mod R
  - Dobro uravnoteženo (balanced)

Koncept programiranja

- Map:
  - Primeni neku funkciju na pojedinačne vrednosti iz skupa podataka radi pravljenja nove liste vrednosti
  - Primer: square x = x * x
    map square [1,2,3,4,5]
    vraća [1,4,9,16,25]
- Reduce:
  - Kombinuj vred. iz skupa pod. radi dobijanja nove vrednosti
  - Primer: sum = (za svaki elem u arr, total +=)
    reduce [1,2,3,4,5]
    vraća 15 (suma svih elemenata)

MapReduce na rač. klasteru





Implementacija toka podataka u MapReduce

Jednostavan primer: Brojanje reči u velikom skupu dokumenata

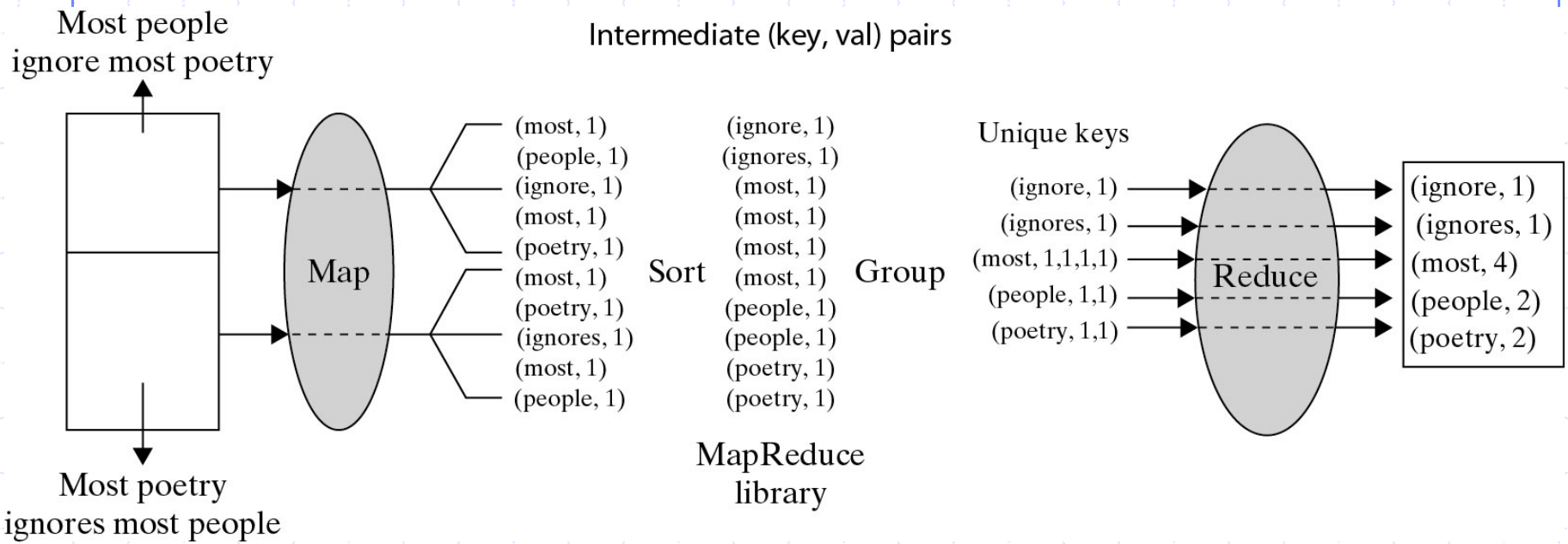
```
map(string value)
  //key: document name
  //value: document contents
  for each word w in value
    EmitIntermediate(w, "1");

reduce(string key, iterator values)
  //key: word
  //values: list of counts
  int results = 0;
  for each v in values
    result += ParseInt(v);
  Emit(AsString(result));
```

Funkcija **map** emituje svaku reč *w* plus pridružen broj pojava (prosto "1" u ovom pseudo-kodu)

Funkcija **reduce** sabira sve brojeve emitovane za određenu reč

Distribucija parova <Ključ, Broj> u primeru brojanja reči



Kako to radi?

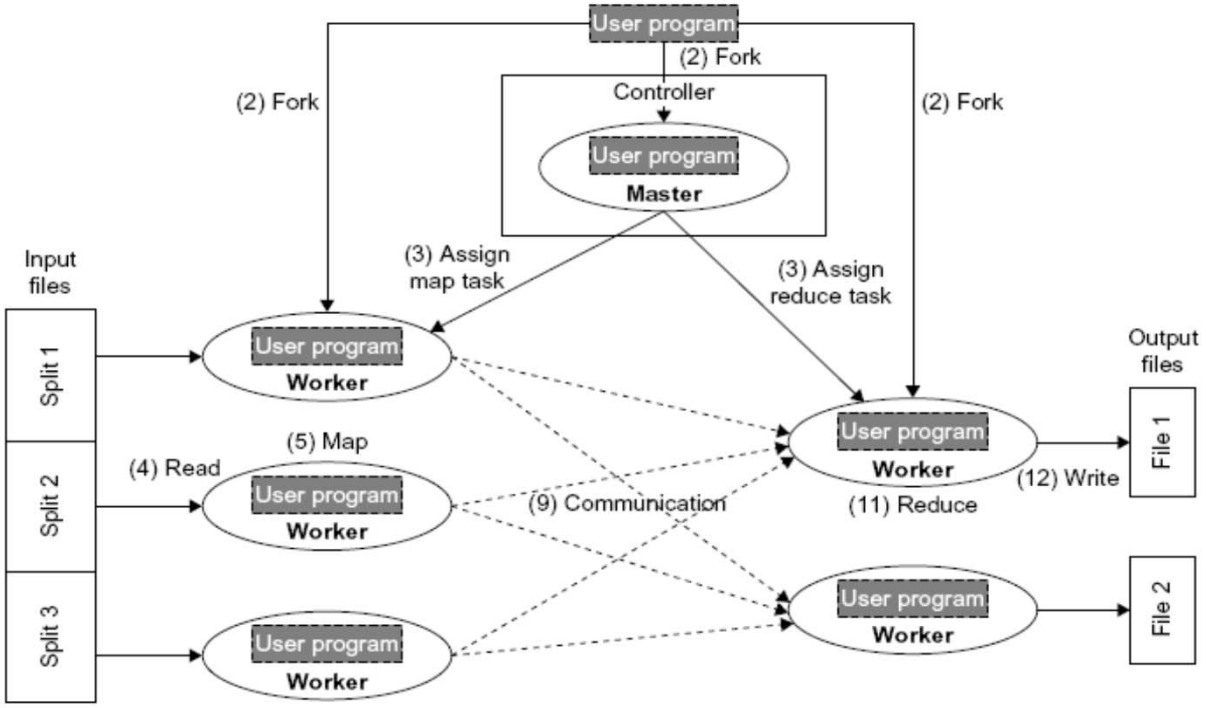


FIGURE 6.6

Control flow implementation of MapReduce.

(Courtesy of Yahoo! Pig Tutorial [54])

- Pozivi funkcije Map se distribuiraju na više mašina putem automatskog particionisanja ulaznih podataka u skup od M delova
- Pozivi funkcije Reduce se distribuiraju putem particionisanja prostora među ključeva u R delova korišćenjem heš funkcije: hash(key) mod R
  - Programer specificira R i funkciju particionisanja

MapReduce: Operativni koraci (1/4)

- Kada korisnički program poziva MapReduce funkciju, dešava se sledeći niz akcija:
  - 1) MapReduce biblioteka u korisničkom programu prvo deli ulazne datoteke u M delova – 16M bajta do 64M bajta po delu. Iza toga pokreće mnogo kopija programa na klasteru mašina
  - 2) Jedna od kopija programa je vodeća (master) ostale su radnici (workers) kojima vodeća kopija dodeljuje posao

MapReduce: Operativni koraci (2/4)

- 3) Radnik kome je dodeljen map zadatak:
  - Čita sadržaj odgovarajućeg ulaznog dela
  - Parsira parove ključ/vrednost iz ulaznih podataka i prosleđuje svaki par korisniči-definisanoj Map funkciji
  - Izlazni među parovi ključ/vrednost proizvedeni od Map funkcije se baferuju u memoriji

- 4) Baferovani parovi se zapisuju na lokalni disk, particionisani u R regiona od strane funkcije za particionisanje.

  Lokacije ovih baferovanih parova na lokalnom disku se prosleđuju nazad do vodeće kopije (master), koja upućuje te lokacije radnicima redukcije.

MapReduce: Operativni koraci (3/4)

- 5) Kada radnik redukcije dobije info od vodeće kopije o tim lokacijama, on čita baferovane podatke sa lokalnih diskova radnika mapiranja.

  Kada radnik redukcije pročita sve među podatke, on ih sortira po među ključevima tako da grupiše zajedno sve pojave istog ključa.

- 6) Radnik redukcije iterira preko sortiranih među podataka i za svaki jedinstven među ključ, prosleđuje ključ i odgovarajući skup među vrednosti do korisničke funkcije redukcije.

  Izlaz iz funkcije redukcije se dodaje na kraj konačne izlazne datoteke.

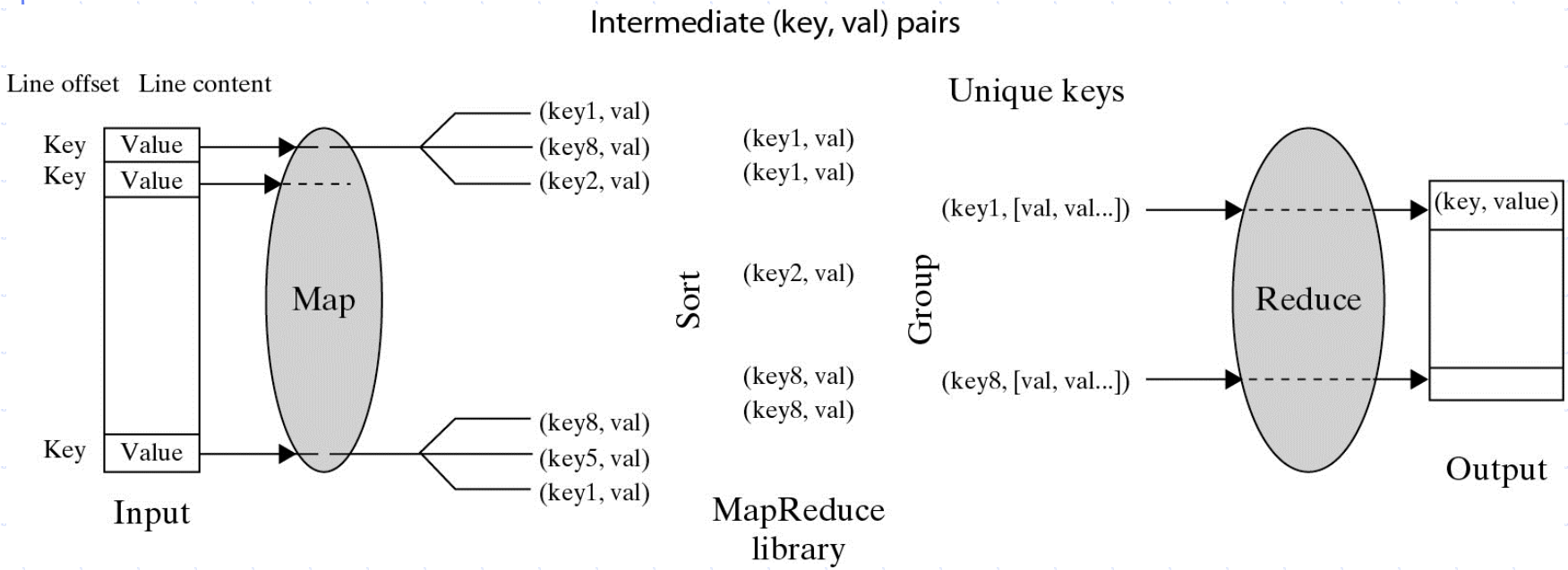
MapReduce: Operativni koraci (4/4)

- 7) Kada se završe svi zadaci mapiranja i zadaci redukcije, vodeća kopija (master) vraća upravljanje korisničkom programu.

  U toj tački, MapReduce poziv u korisničkom programu se vraća nazad u korisnički kod.

  Po uspešnom završetku, rezultat izvršenja MapReduce (izlaz) je raspoloživ u R izlaznih datoteka.

Logički tok podataka u 7 koraka obrade u MapReduce procesu



- Map funkcija generiše parove (Ključ, Vrednost) na više raspoloživih radnika mapiranja (VM instanci). Ovi parovi se onda sortiraju i grupišu na osnovu uređenja ključeva. Različite grupe ključeva se onda obrađuju paralelno od strane više radnika redukcije.

Pitanje lokalnosti

- Politika raspoređivanja od strane vodeće kopije:
  - Pitaj GFS za lokacije replika blokova ulazne datoteke
  - Zadaci mapiranja obično rade nad blokovima pod. od 64MB (== veličina bloka u GFS)
  - Zadaci mapiranja se raspoređuju tako da su replike ulaznih GFS blokova na istoj mašini ili u istom stalku
- Efekt:
  - Hiljade mašina čitaju ulaz brzinom lokalnih diskova
  - Bez toga, komutatori u stalku ograničavaju brzinu čitanja

Otpornost na otkaze (1/3)

- Reaktivan rad
  - Otkaz radnika
    - Poruke javljanja, vodeći periodično proziva radnike
      - NEMA odgovora = radnik otkazao
    - Ako procesor radnika otkaže, zadaci tog radnika se dodeljuju drugom radniku
  - Otkaz vodećeg (master)
    - Vodeći periodično zapisuje kontrolne tačke
    - Drugi vodeći može biti pokrenut iz stanja zadnje kontrolne tačke
    - Ako vodeći ipak otkaže, posao će biti nasilno uništen

Otpornost na otkaze (2/3)

- Proaktivan rad (Redundantno izvršenje)
  - Problem "zaostalih" (sporih radnika)
    - Drugi poslovi troše resurse na mašini
    - Loši diskovi sa soft greškama prenose podatke vrlo polako
    - Čudne stvari: onemogućene procesorske keš memorije (!!)
  - Kada je računanje skoro gotovo, mogućno ponovno raspoređivanje zadataka čije izvršenje je u toku
  - Kad god se, bilo primarna ili rezervna izvršenja završe, treba ih označiti kao završene

Otpornost na otkaze (3/3)

- Greška ulaza: loši slogovi
  - Map/Reduce funkcije ponekad otkažu za neke ulaze
  - Najbolje rešenje je pronaći grešku i otkloniti je, ali to nije uvek moguće
  - U slučaju otkaza segmenta
    - Iz rukovaoca signalom, pošalji UDP paket vodećem
    - Uključiti redni broj sloga koji je obrađen
  - Preskočiti loše slogove
    - Ako vodeći vidi dva otkaza za isti slog, sledećem radniku se saopštava da preskoči slog

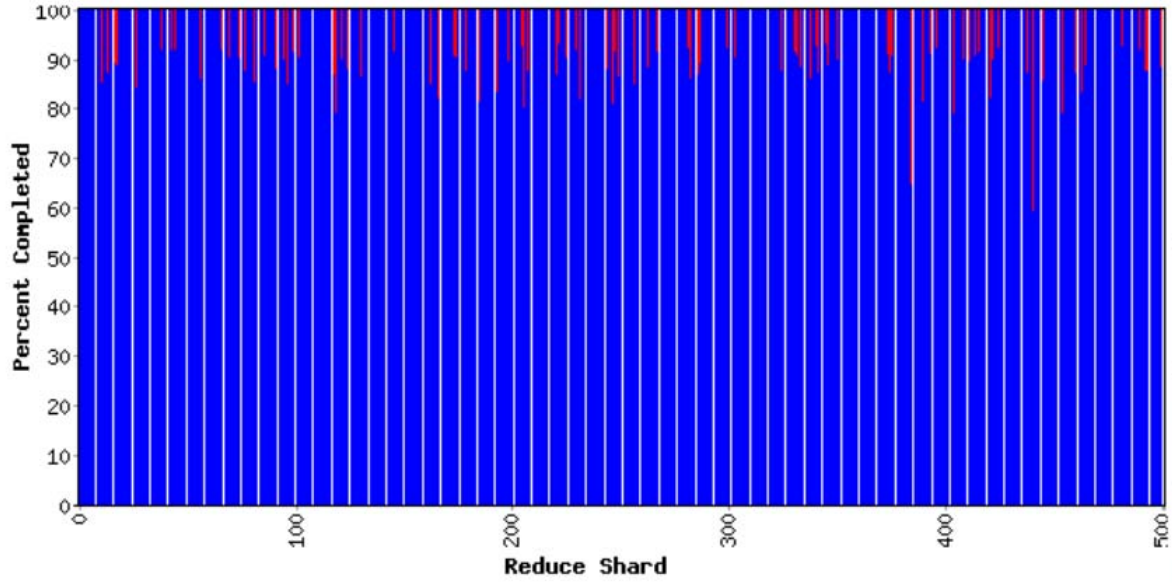
Nadzor stanja

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	849.5
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

Neki važni zahtevi

- Ni jedna redukcija ne može da započne dok se mapiranje ne završi
- Vodeći mora da saopštava lokacije među dat.
- Zadaci se raspoređuju na osnovu lokacije pod.
- Ako radnik mapiranja otkaže bilo kad pre nego se završi redukcija, zadatak se mora ponovo izvršiti
- MapReduce biblioteka radi najviše teškog posla za nas!

Drugi primeri (1/2)

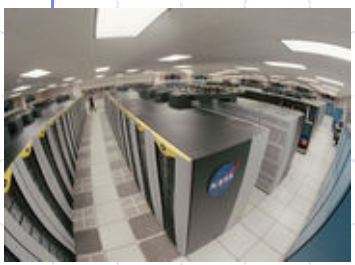
- Distribuirani Grep:
  - Map funkcija emituje liniju ako ima poklapanja sa zadatim šablonom.
  - Reduce funkcija je jedinična funkcija koja kopira dostavljene među pod. na izlaz.
- Broj URL pristupa:
  - Map funckija obrađuje log zahteva web stranice i pravi izlaz <URL, 1>,
  - Reduce funkcija sabira sve vred. za isti URL, emituje parove <URL, total count>.

Drugi primeri (2/2)

- Invrezni Web-Link graf; svi URL sa referencom na tgt:
  - Map funkcija pravi izlaz <tgt, src> za svaki link na cilj u str. sa imenom src,
  - Reduce spaja listu svih izvorišnih URL u vezi sa datim ciljnim URL i emituje par: <tgt, list(src)>.
- Obrnut indeks; svi URL na kojima se pojavljuje reč w:
  - Map funkcija parsira svaki dokument, emituje niz parova <word, doc_ID>,
  - Reduce funkcija prihvata sve parove za datu reč, sortira odgovarajuće doc_ID i emituje par <word, list(doc_ID)>.
  - Skup svih izlaznih parova formira jednostavan obrnuti indeks.

Implementacije MapReduce

MapReduce



Cluster,
1, Google
2, Apache Hadoop

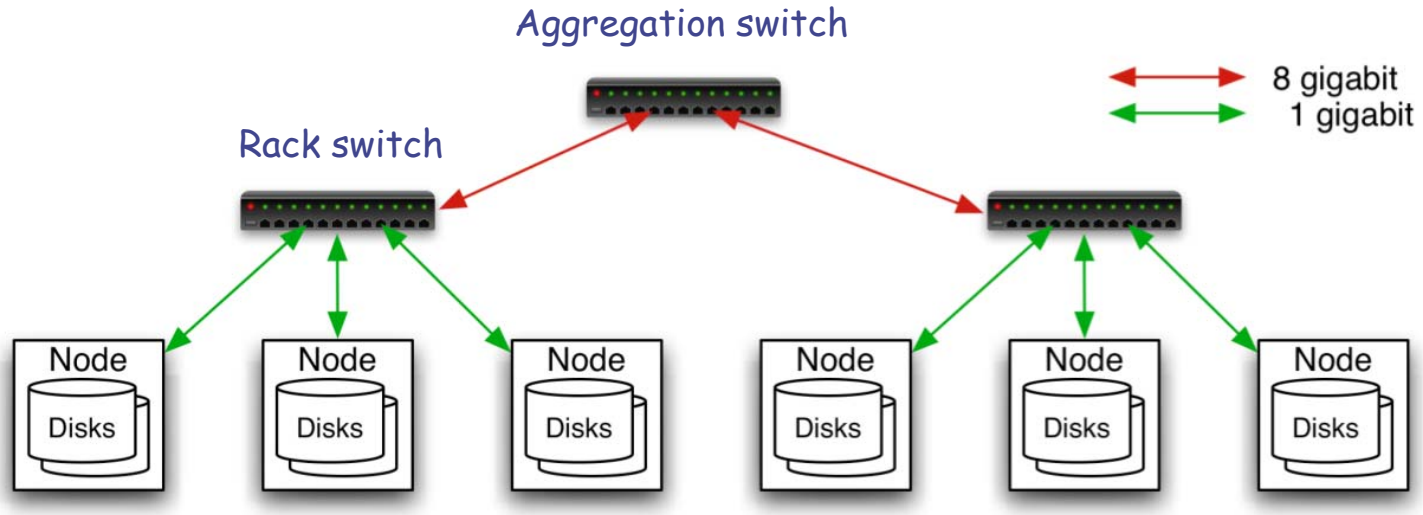
Multicore CPU,
Phoenix @ stanford

GPU,
Mars@HKUST

Hadoop:
SW platforma razvijena od Yahoo

- Atraktivne osobine Hadoop:
  - Skalabilnost: lako se skalira za skladištenje i obradu peta bajta podataka u Web prostoru
  - Ekonomičnost: MapReduce otvorenog koda minimizira režiju u pokretanju zadataka i masivnoj komunikaciji podataka
  - Efikasnost: Obrada podataka sa visokim stepenom paralelizma preko velikog broja komercijalno dostupnih čvorova
  - Pouzdanost: Automatski održava više kopija podataka radi ponovnog rasporeda računskih zadataka u slučaju otkaza

Tipičan Hadoop klaster



- 40 čvorova/stalku, 1000-4000 čvorova u klasteru
- 1 Gbps propusni opseg unutar stalka, 8 Gbps izvan stalka
- Specifikacija čvora(Yahoo terasort):
  8 x 2GHz jezgra, 8 GB RAM, 4 diska (= 4 TB?)

Tipičan Hadoop klaster

Izazovi

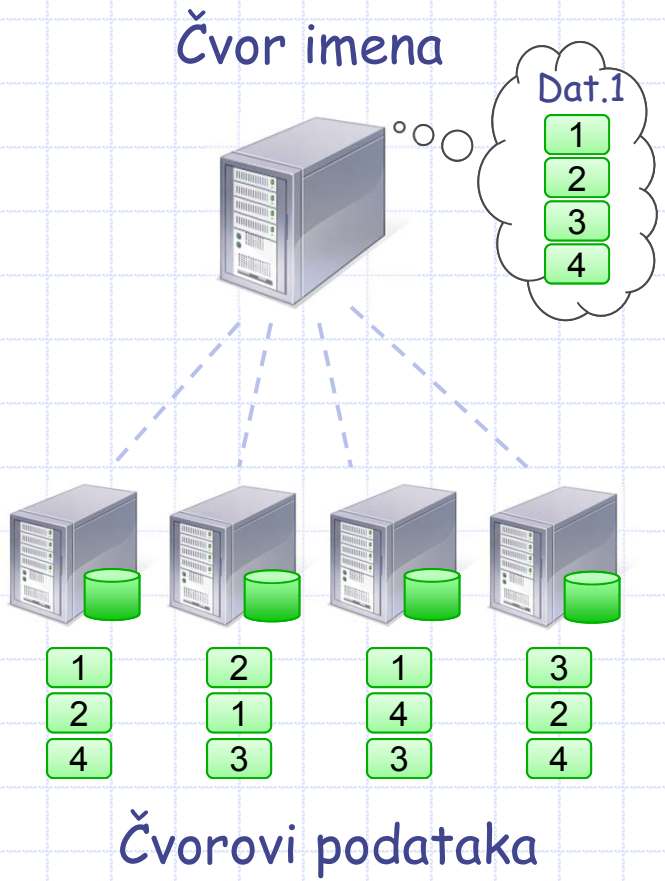
- Jeftini čvorovi otkazuju, posebno ako ih je puno
  - Srednje vreme između otkaza (MTBF) za 1 čvor 3 god
  - MTBF za 1000 čvorova = 1 dan
  - Rešenje: ugraditi otpronost na otkaze u sistem
- Obična mreža = nizak propusni opseg
  - Rešenje: Distribuirati računanje kod podataka
- Programiranje distribuiranih sistema je teško
  - Rešenje: Model programiranja sa paralelizmom pod.: korisnici pišu “map” i “reduce” funkcije, sistem distribuira rad (obradu) i rukuje otkazima

Hadoop komponente

- Distribuirani sistem datoteka (HDFS)
  - Jedan prostor imena za ceo klaster
  - Replicira podatke 3x radi otpornosti na otkaze
- MapReduce okruženje
  - Izvršava korisničke poslove specificirane putem "map" i "reduce" funkcija
  - Rukuje distribucijom rada i oporavkom od otkaza

Hadoop distribuirani sistem datoteka

- Datoteke podeljene u 128MB blokove
- Blokovi replicirani na nekoliko čvorova podataka (obično 3)
- Jedan čvor imena čuva meta podatke (imena dat., lokacije blokova, itd.)
- Optimizovan za velike dat., i sekvencijalno čitanje
- Pod. se mogu dodavati samo na kraj dat. (append-only)



Hadoop Arhitektura HDFS i MapReduce

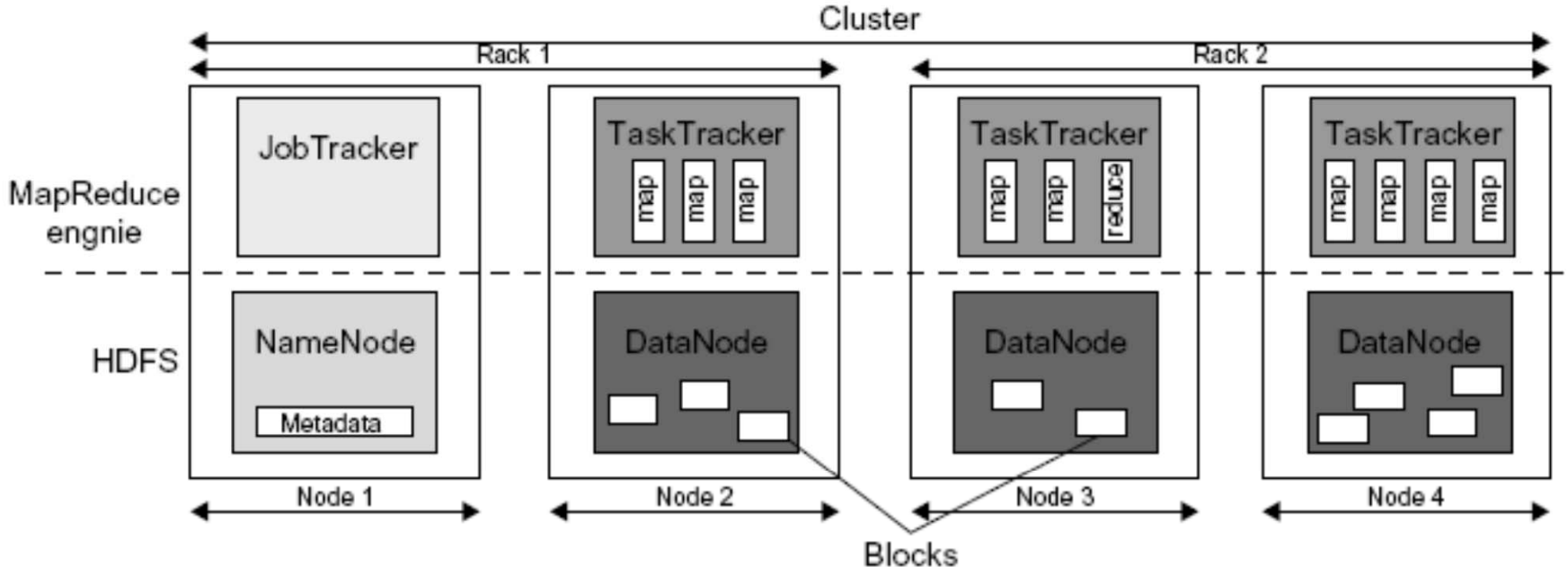


FIGURE 6.11

HDFS and MapReduce architecture in Hadoop.

Zaštićena obrada upita sa Hadoop/MapReduce

- Principi prepisivanja upita i optimizacije su definisani i implementirani za dva tipa podataka
  - (i) Relacioni podaci: Zaštićena obrada upita sa HIVE
  - (ii) RDF podaci: Zaštićena obrada upita sa SPARQL
- Demonstrirano sa XACML politikama
- Demo: Kings College i Univerzitet Insubria
  - Prvi demo (2010): Svaka strana je podnela svoje podatke i politike
  - Jedan oblak rukuje podacima i politikama
  - Drugi demo (2011): Više oblaka

Jezici višeg niova iznad Hadoop: Pig i Hive

Motivacija

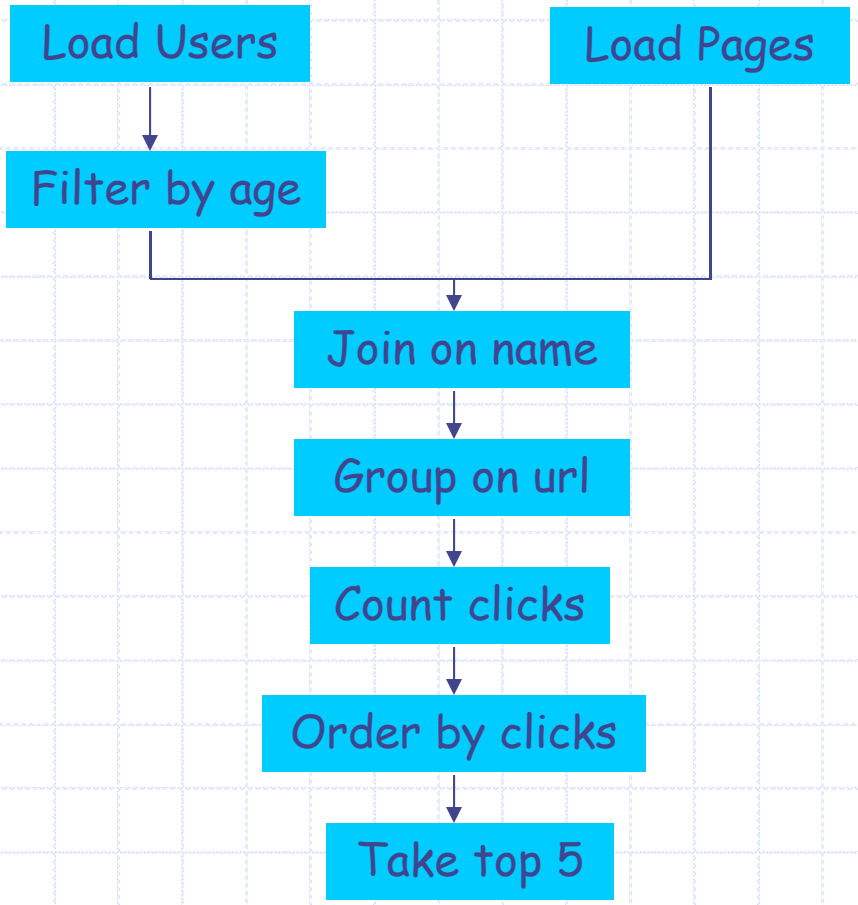
- Mnogi paralelni algoritmi se mogu izraziti sekvencom MapReduce poslova
- Ali MapReduce je dosta niskog nivoa: mora se razmišljati o ključevima, vrednostima, particionisanju, itd.
- Da li možemo pronaći zajedničke "gradivne blokove poslova"?

Pig

- Započet u Yahoo! Research
- Radi oko 30% Yahoo!-ovih poslova
- Osobine:
  - Izražava sekvence MapReduce poslova
  - Model podataka: ugnježdene "torbe" stavki
  - Obezbeđuje relacione (SQL) operatore (JOIN, GROUP BY, itd.)
  - Lako se utiče u (plug in) Java funkcije
  - Pig Pen razvojno okruženje za Eclipse

Primer problema

Neka su podaci o korisnicima u jednoj dat., podaci o poseti stranica u drugoj dat., i treba pronaći prvih 5 najposećenijih str. od korisnika starosti 18-25



U MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadPrinted extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see
            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join users and pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top  100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

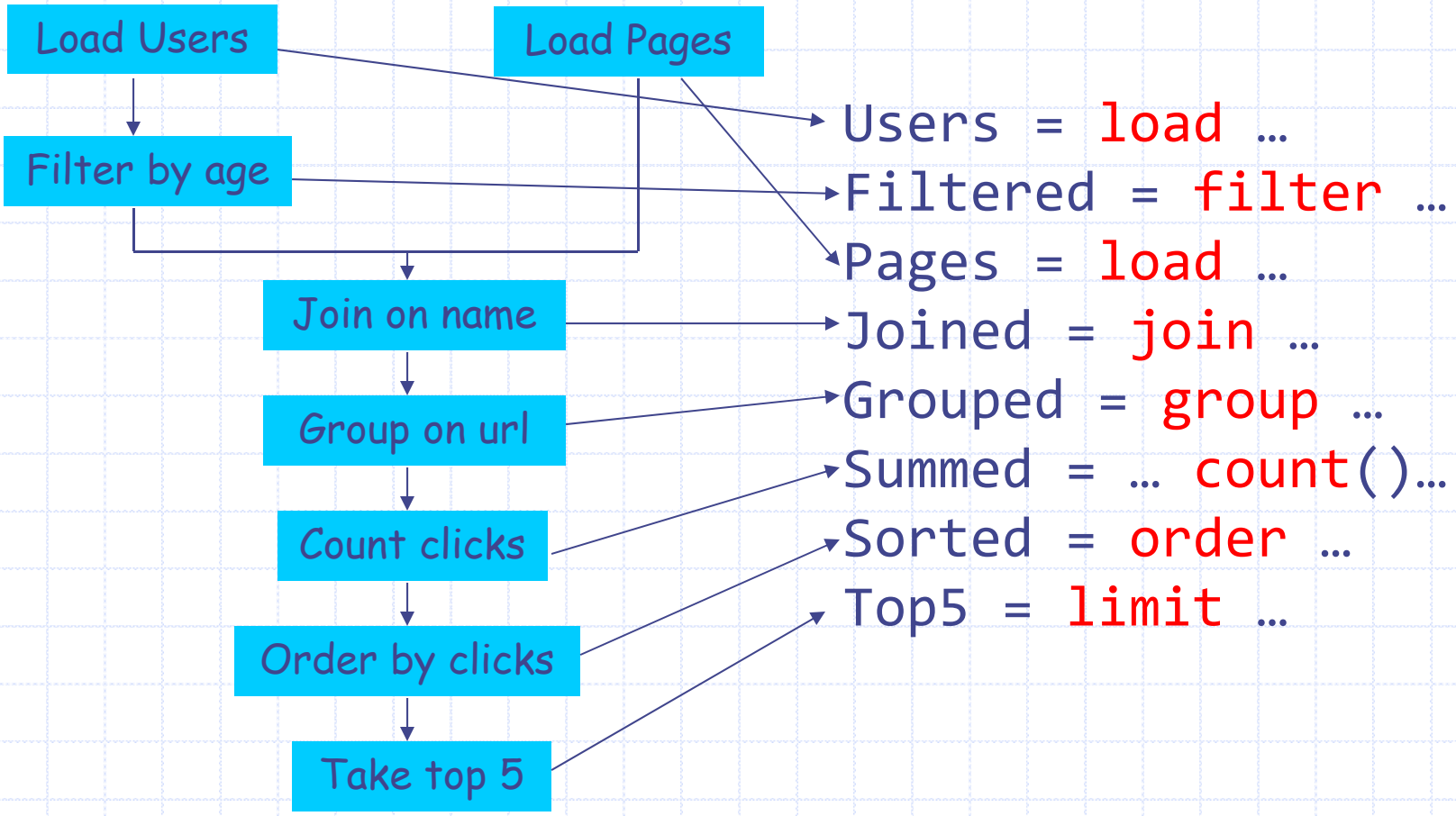
U Pig Latin

```
Users    = load ‘users’ as (name, age);
Filtered = filter Users by
                  age >= 18 and age <= 25;
Pages    = load ‘pages’ as (user, url);
Joined   = join Filtered by name, Pages by user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
                   count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;

store Top5 into ‘top5sites’;
```

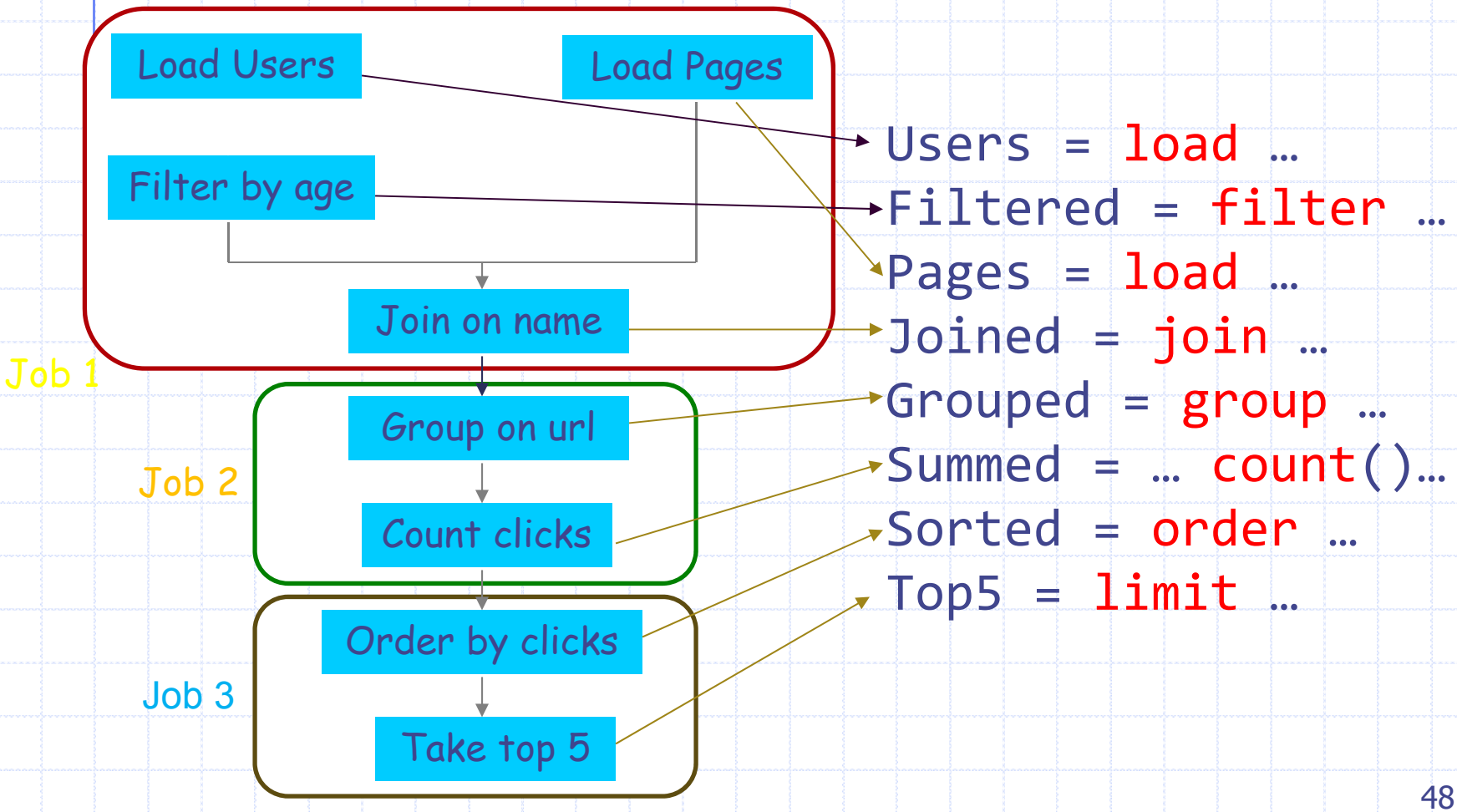
Jednostavnost prevođenja (1/2)

Primetite kako se prirodno komponente posla prevode u Pig Latin.



Jednostavnost prevođenja (2/2)

Primetite kako se prirodno komponente posla prevode u Pig Latin.



Hive

- Razvijen u Facebook
- Koristi se za većinu Facebook poslova
- "Relaciona baza podataka" iznad Hadoop
  - Održava listu šema tabela
  - Upitni jezik sličan SQL-u (HQL)
  - Iz HQL se mogu pozvati Hadoop Streaming skripte
  - Podržava particionisanje tabela, klasterizaciju, složene tipove podataka i neke optimizacije

Jednostavni Hive upiti

Pronađi prvih 5 str. posećenih od kor. starosti 18-25:

```
SELECT p.url, COUNT(1) as clicks
FROM users u JOIN page_views p ON (u.name = p.user)
WHERE u.age >= 18 AND u.age <= 25
GROUP BY p.url
ORDER BY clicks
LIMIT 5;
```

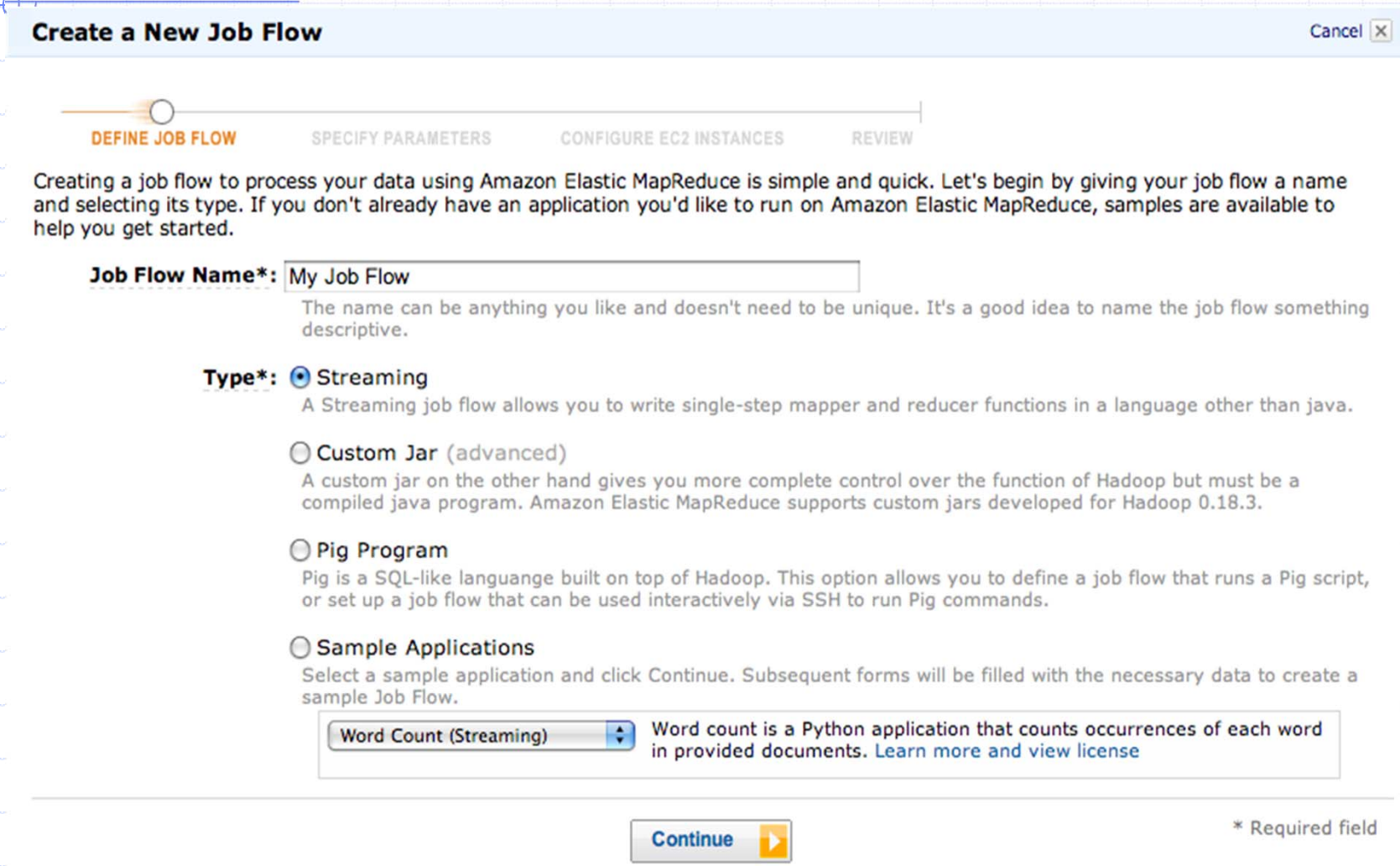
Filtriraj posete stranica putem Python skripte:

```
SELECT TRANSFORM(p.user, p.date)
USING 'map_script.py'
AS dt, uid CLUSTER BY dt
FROM page_views p;
```

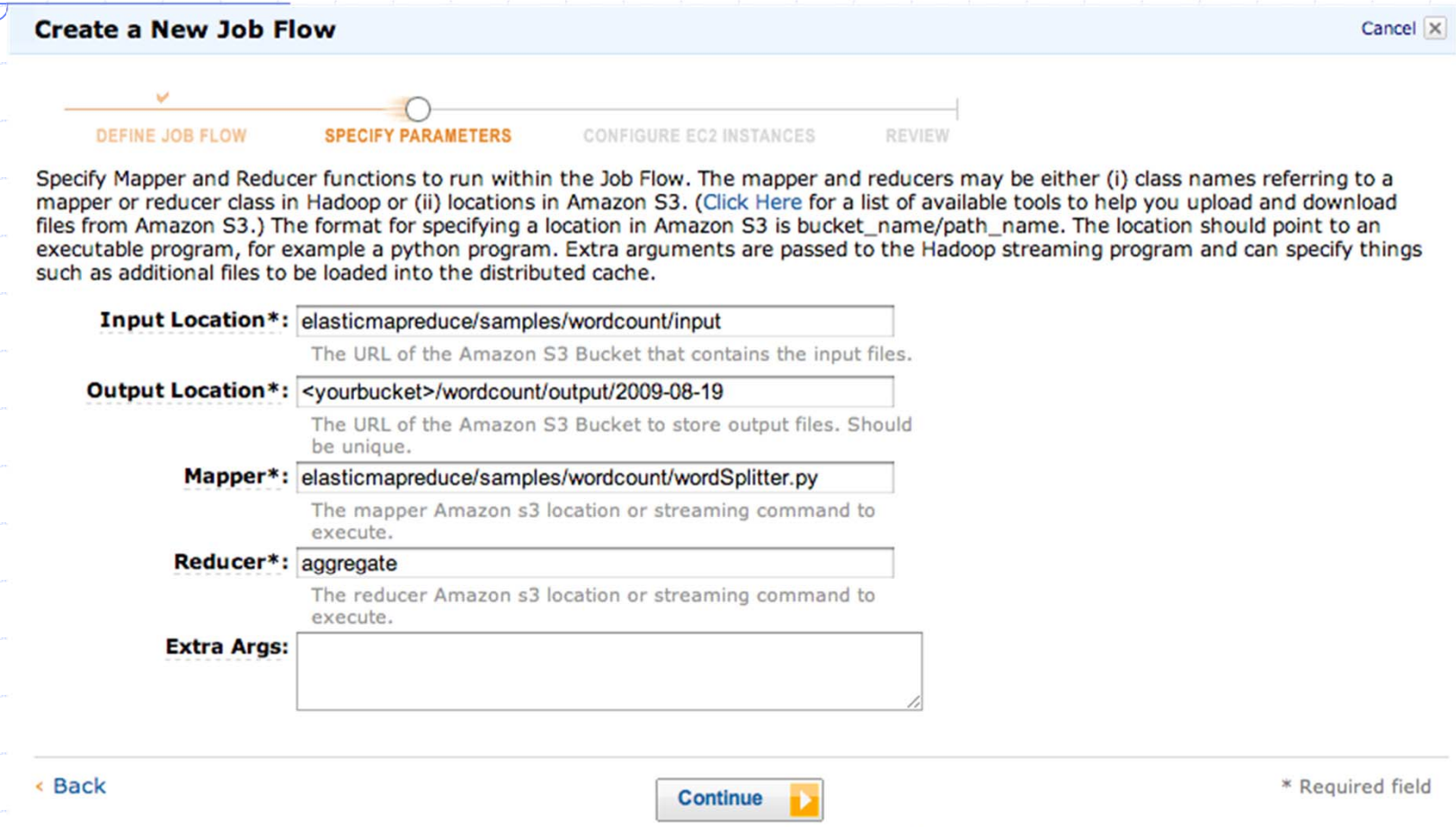
Amazon Elastic MapReduce

- Osobine:
  - Obezbeđuje web spregu i alate sa komandama za izvršenje Hadoop poslova na Amazon EC2
  - Podaci se skladište u Amazon S3
  - Nadzor poslova i zaustavljanje mašina posle korišćenja
  - Mali dodatni trošak iznad redovnog EC2 plaćanja
  - Ako je potrebna veća kontrola nad izvršenjem Hadoop, moguće je ručno lansirati Hadoop klaster na EC2 koristeći skripte u src/contrib/ec2

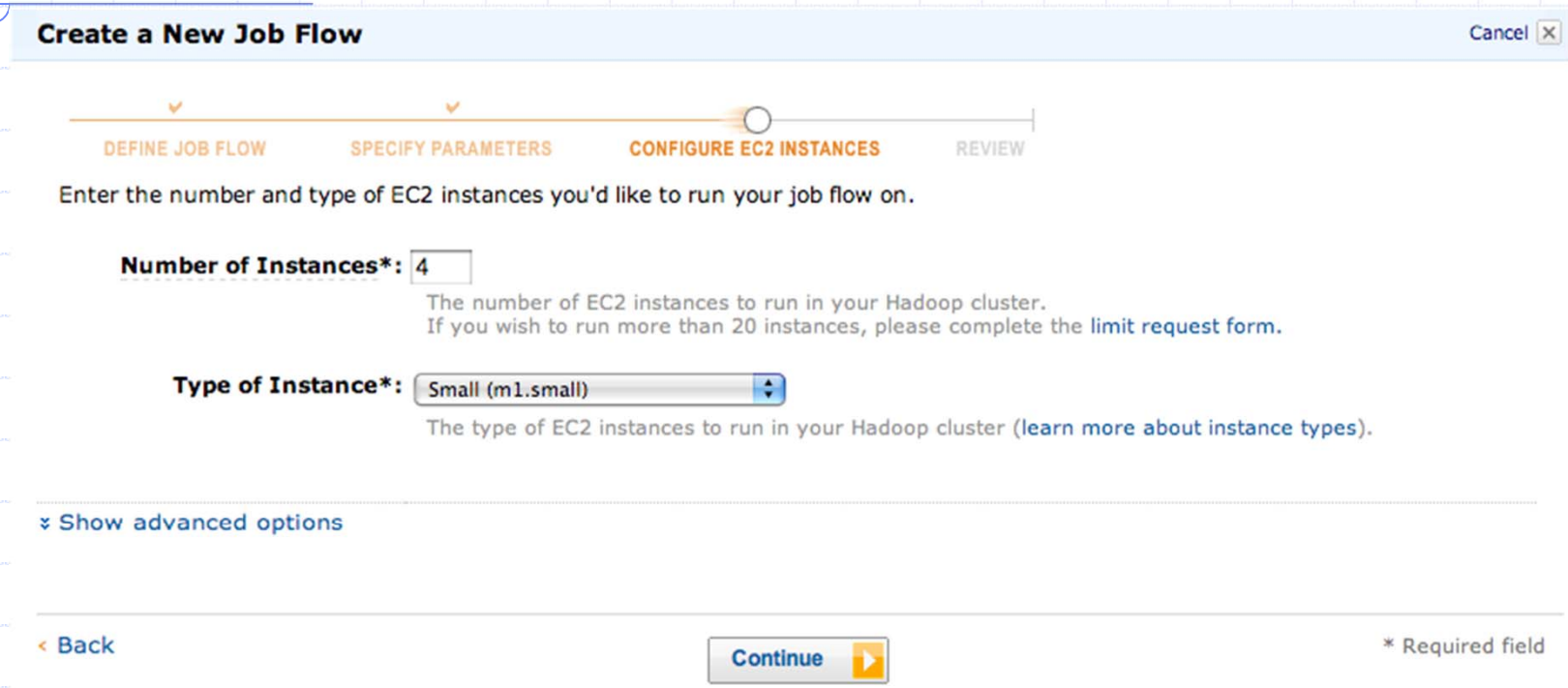
Elastic MapReduce Workflow (1/4)



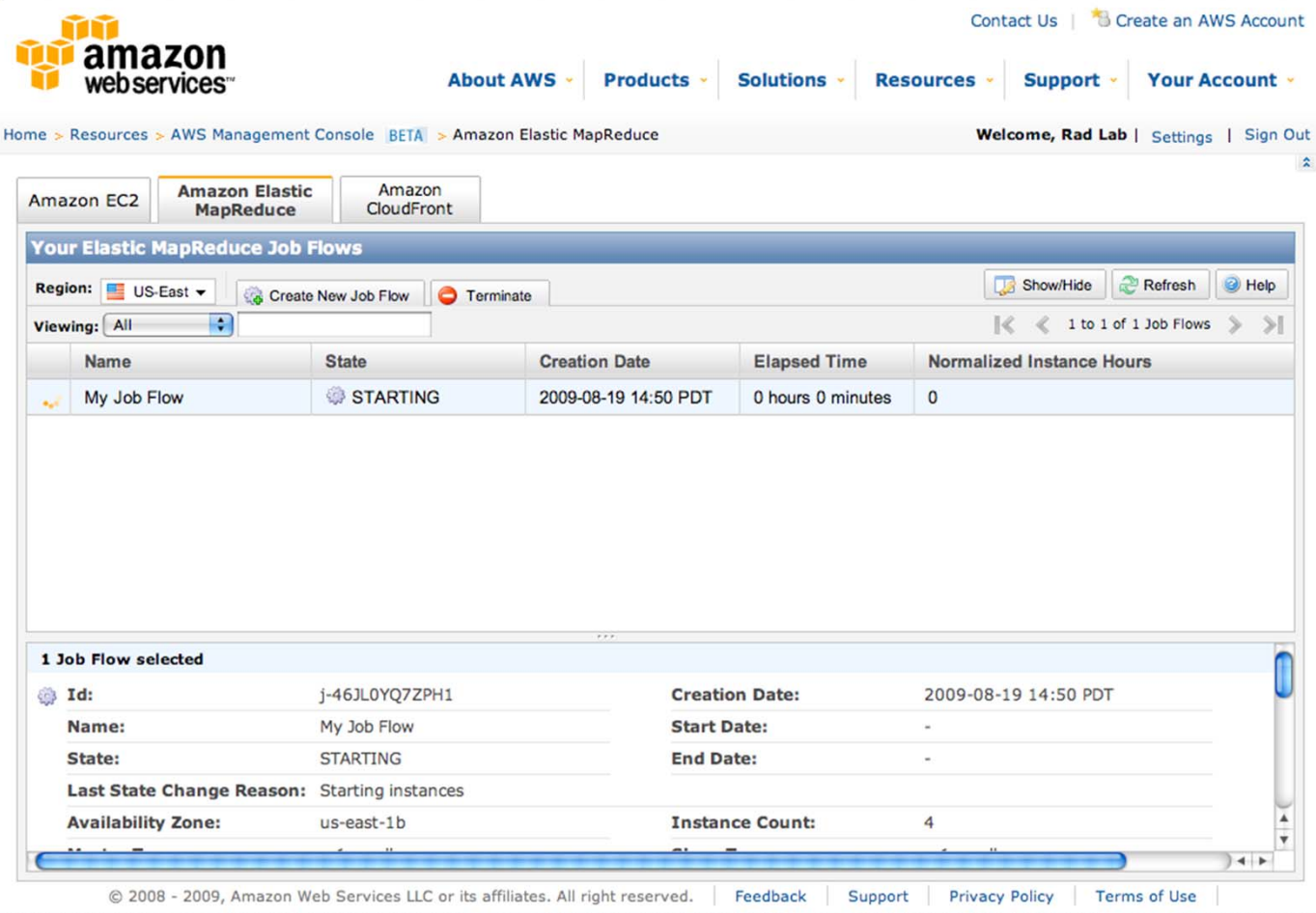
Elastic MapReduce Workflow (2/4)



Elastic MapReduce Workflow (3/4)



Elastic MapReduce Workflow (4/4)



Zaključci

- Model programiranja MapReduce skriva složenost distribucije rada i oporavka od otkaza
- Načelni projektantski principi:
  - Napravi ga skalabilno, tako da se može upotrebiti raspoloživ HW
  - Napravi ga jeftino, sniženjem troškova HW, programiranja i admin.
- MapReduce nije pogodan za sve probleme, ali kada radi, može uštedeti dosta vremena
- Računanje u oblaku omogućava jednostavno korišćenje Hadoop (i drugog paralelnog SW) sa potrebnom skalom

Resursi

- Hadoop: http://hadoop.apache.org/core/
- Pig: http://hadoop.apache.org/pig
- Hive: http://hadoop.apache.org/hive
- Video tutorijali: http://www.cloudera.com/hadoop-training

- Amazon Web Services: http://aws.amazon.com/
- Amazon Elastic MapReduce guide: http://docs.amazonwebservices.com/ElasticMapReduce/latest/GettingStartedGuide/