

Cloud computing

Cloud Computing

- Un nou **model** de **calcul** client – server (consumator – furnizor)
 - resursele (CPU, memorie...) oferite de server ca **utilitati**
 - la cererea clientului (**on-demand**)
 - clientul plateste pentru utilizarea resurselor (**pay-as-you-go**)
- *Acces simplu* la resurse
 - oferite ca **servicii** in Cloud
 - accesibile prin **orice echipament** cu conectivitate Internet

- *Scalabilitate nelimitata*
 - furnizorul poate oferi orice volum de resurse
- *Elasticitate*
 - resursele se aloca in functie de necesitati
 - pot fi adaugate la cresterea solicitarii si eliberate cand solicitarea scade
- *Mai putine probleme pentru utilizator – problemele sunt asumate de furnizori*
 - riscuri precum caderi ale echipamentelor
 - mentenanța hardware si instruirea personalului IT

Definitia NIST pentru Cloud computing

- *Cloud computing is*
 - *a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (networks, servers, storage, applications, and services)*
 - *that can be rapidly provisioned and released*
 - *with minimal management effort or service provider interaction.*
- Cloud computing is a new operations model that
 - brings together a set of existing technologies
 - virtualization
 - utility-based pricing

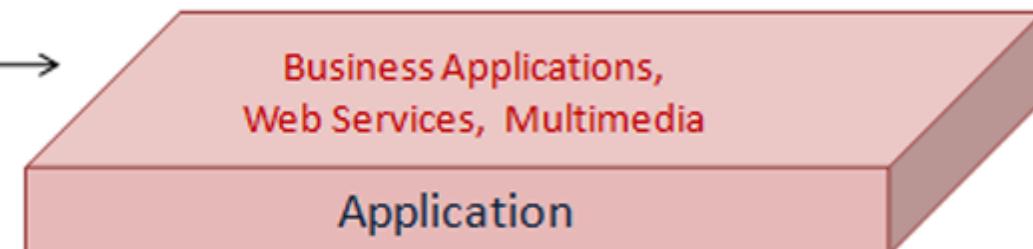
Arhitectura Cloud computing

End Users



Software as a Service (SaaS)

Resources Managed at Each layer



Examples:

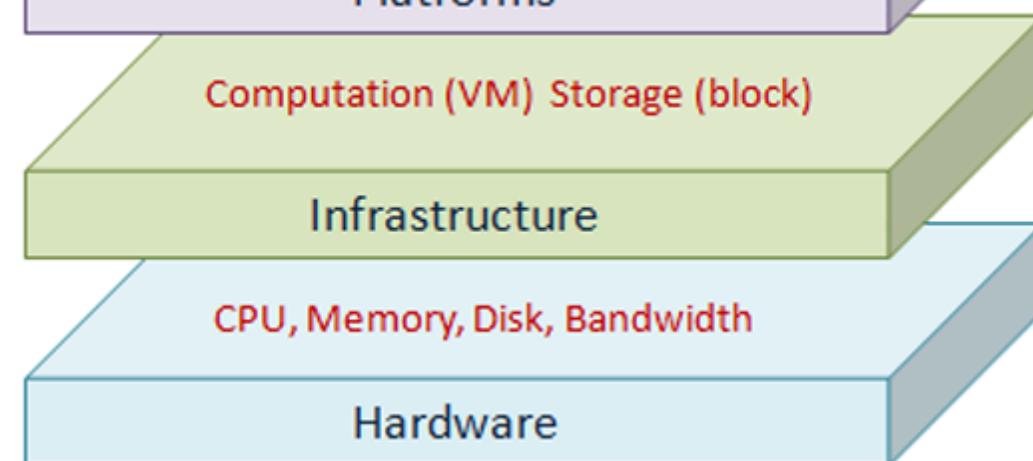
Google Apps,
Facebook, YouTube
Salesforce.com

Platform as a Service (PaaS)



Microsoft Azure,
Google AppEngine,
Amazon SimpleDB/S3

Infrastructure as a service (IaaS)



Amazon EC2,
GoGrid
Flexiscale

Data Centers

CPU, Memory, Disk, Bandwidth

Hardware

Nivelul hardware

- **Rol:** gestiunea resurselor fizice
 - servere, rutere, switch-uri, surse de energie, sisteme de racire
- Implementat in **Data Centers**
 - contine mii de servere interconectate prin switch-uri, rutere sau alte echipamente
- **Probleme** tipice la nivelul hardware
 - configurare hardware
 - toleranta la defectari
 - gestiunea traficului
 - gestiunea surselor de alimentare si a echipamentelor de racire

Nivelul infrastructura IaaS

- Provizionarea la cerere a resurselor de infrastructura
- Partitioneaza resursele fizice folosind virtualizarea
 - masina virtuala, VM = implementare software a unui calculator, care executa programe ca o masina fizica
 - ex. Xen, KVM, VMware
- Exemple furnizori IaaS
 - Amazon EC2, GoGrid and Flexiscale.

Nivelul platforma PaaS

- Furnizeaza resurse la nivelul platformei
 - suport pentru sistemul de operare
 - framework-uri pentru dezvoltare de software
 - baze de date
- Exemple de furnizori PaaS
 - Google App Engine, Microsoft Windows Azure, Force.com.
 - Google App Engine suporta
 - implementarea depozitelor de date,
 - baze de date,
 - logica de business (ex. workflow) pentru aplicatii

Nivelul aplicatie SaaS

- Se refera la aplicatiile oferite in Cloud prin intermediul Internetului
- Exploataza **scalabilitatea** Cloud-ului pentru a realiza
 - performanta crescuta
 - disponibilitate
 - cost de operare redus
- Exemple de furnizori SaaS
 - Salesforce.com, Rackspace, SAP Business ByDesign.

Model bazat pe servicii

Accent pe gestiunea serviciilor

- Furnizorul ofera serviciile conform unui **Service Level Agreement** (SLA) negociat cu consumatorii

Aspecte QoS

- *Performanță*
- *Disponibilitate*
- *Siguranta*
- *Scalabilitate*
- *Cost*

Metrici QoS

- *temp de răspuns*
- *rata de abandon, rata de utilizare*
- *temp intre defectari, temp mediu de recuperare*
- *crește volum servicii cu pastrare performanță*
- *cost energie, cost financiar*

Service Level Objective - oferă QoS la valori ale metricilor mai bune decât un prag specificat

SLA = set de SLOs țintă, negociat intre furnizor si consumator

Model de plată similar utilitatilor

- Se platește doar “consumul” - pay-per-use
- Schema de plată difera de la un serviciu la altul
 - inchiriere cu ora a unei mașini virtuale
 - taxarea consumatorului pe baza
 - numărului de clienți care folosesc serviciul
 - numărului de invocări pe secundă a unui serviciu

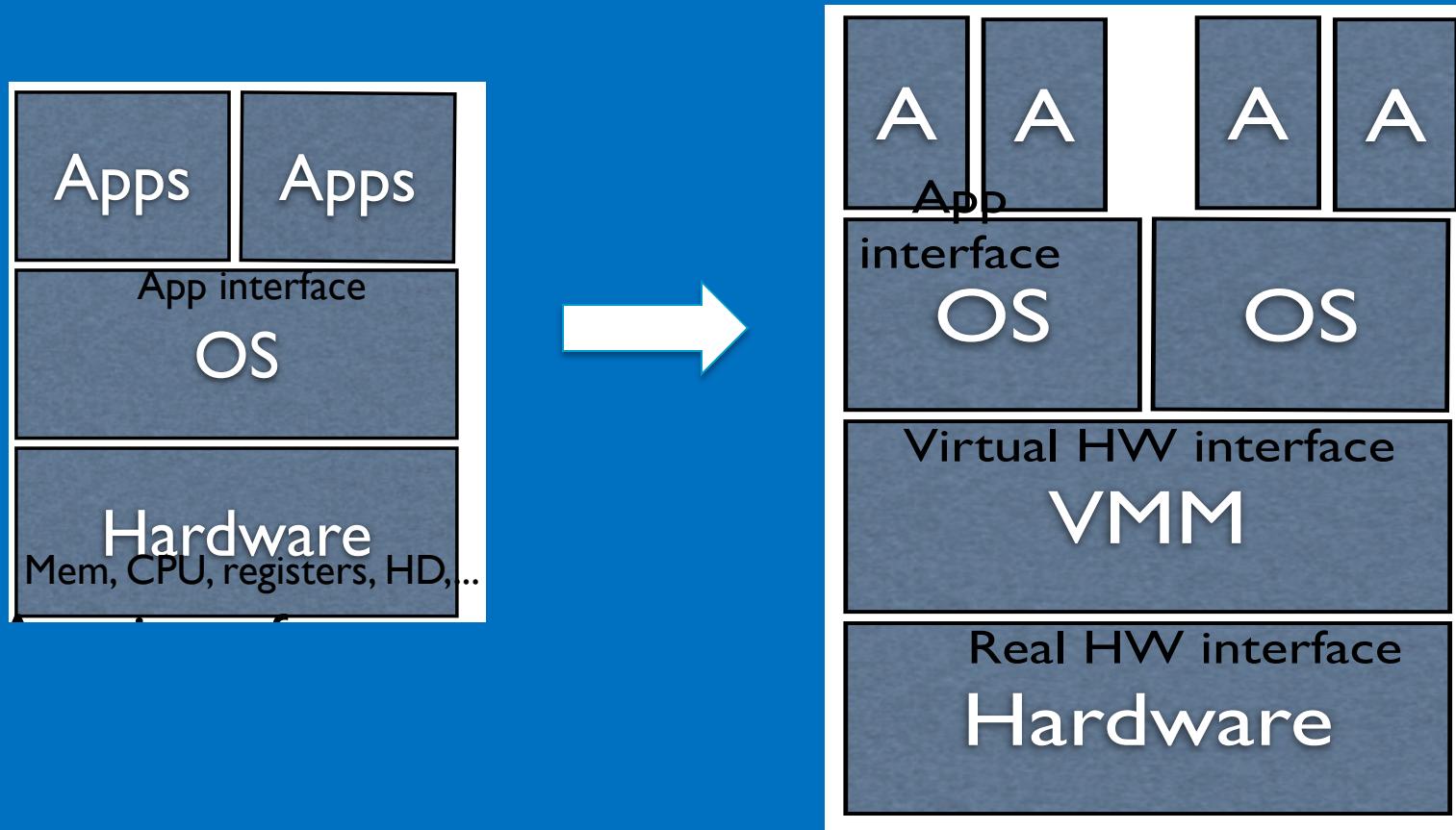
Costuri la Amazon EC2 (Elastic Compute Cloud)

- instante **la cerere** - on-demand
 - plata pe ora cu angajament pe termen scurt
- instante **rezervate**
 - plata redusa, pe ora, facuta o singura data
 - trei tipuri: utilizare usoara, medie, grea
- instanțe **spot**
 - se liciteaza capacitatile Amazon EC2 nefolosite
 - pretul fluctueaza in functie de cerere si oferta
- **transferul de date** se plateste pe luna
- **stocarea** in Elastic Block **Store** este platita pe GB-luna sau dupa numarul de cereri I/O

Tipuri de Cloud

- **Public** – Furnizorii ofera resursele ca servicii pentru publicul larg
- **Private** – Folosit exclusiv de membrii unei singure organizatii
 - cunoscute ca **internal clouds**
 - construit si gestionat de organizatie sau de furnizori externi
- **Hibrid** – combinatie de modele public privat
 - parte a serviciilor ruleaza in cloud privat, restul in cloud public
- **Virtual Privat** – ruleaza peste cloud-uri publice
 - VPC este **mai holistic** decat VPN - virtualizeaza
 - servere, aplicatii si
 - reteaua de comunicare

Arhitectura IaaS – Masini Virtuale



- VM Monitor supporta mai multe OSs
- Poare rula direct peste hardware
- sau peste un OS

Motivatia pentru virtualizare

- Utilizarea mai buna a resurselor
 - Masini puternice sunt **folosite mai eficient** prin multiplexarea hardware
- Pot **migra VMs** de la o masina la alta fara oprirea acestora (fara shutdown)
- **Utilizarea** software in paralel cu **dezvoltarea** lui
- Poate rula simultan mai multe OSs
- Poate face dezvoltarea sistemului (de ex. OS) la nivel utilizator

- Portabilitate
 - OSs si aplicatii pot rula fără modificari
- Izolare
 - VMM protejeaza resursele si VMs una fata de alta
- Performanta (!?)
 - VMM este inca un nivel de software → overhead
 - Ex. VMware:
 - aplicatii CPU-intensive : overhead 2-10%
 - aplicatii I/O-intensive : overhead 25-60%
 - La fel ca la OS, se doreste minimizarea acestui overhead

Amazon EC2

Un segment din matricea de tipuri de instanțe

Detalii la
<https://aws.amazon.com/ec2/instance-types/>

EBS – Elastic Block Store

Instance Type	vCPU	Memory (GiB)	Storage (GB)	Networking Performance	Physical Processor
t2.nano	1	0.5	EBS Only	Low	Intel Xeon family
t2.micro	1	1	EBS Only	Low to Moderate	Intel Xeon family
t2.small	1	2	EBS Only	Low to Moderate	Intel Xeon family
t2.medium	2	4	EBS Only	Low to Moderate	Intel Xeon family
t2.large	2	8	EBS Only	Low to Moderate	Intel Xeon family
m4.large	2	8	EBS Only	Moderate	E5-2676 v3

PaaS – Sistem de fisiere distribuit

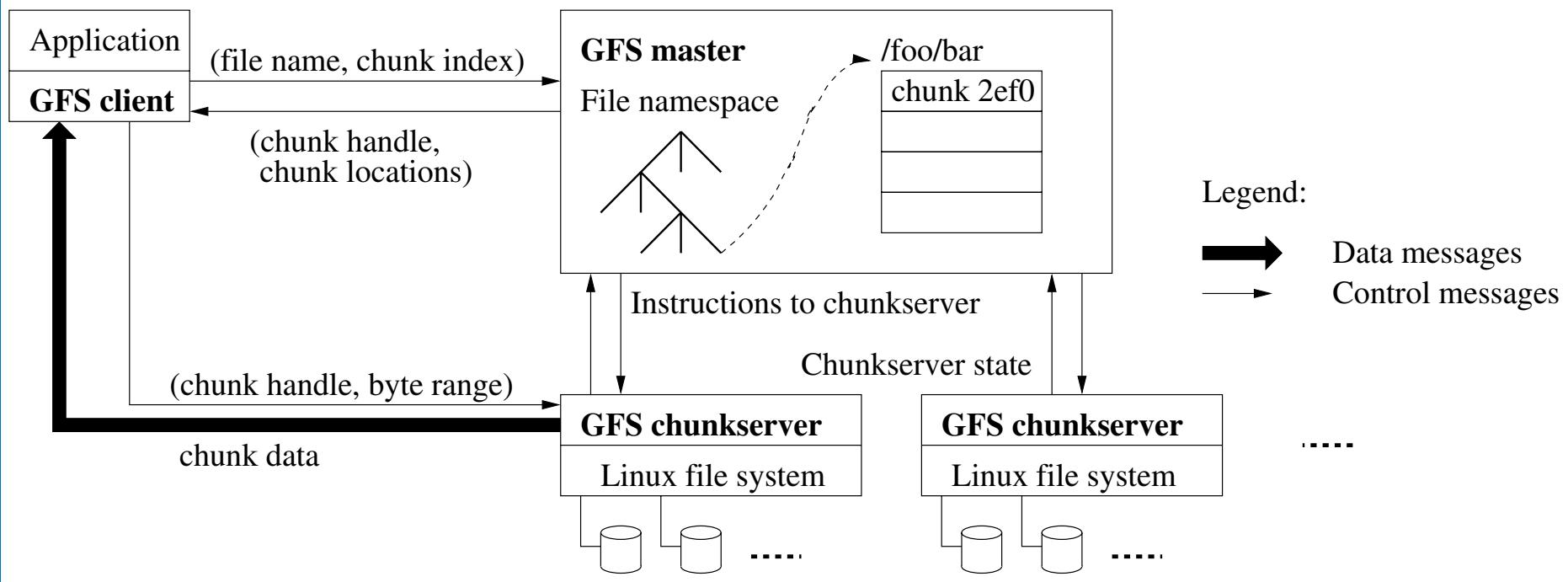
Google File System (GFS)

- Proiectat pentru eficiență și acces sigur la date
- Nu are interfață standard (POSIX) dar fisierele sunt organizate ierarhic în directoare și identificate prin numele caii
- Folosește Clustere de servere obisnuite, de mari dimensiuni
- Un cluster GFS conține un singur master și mai multe servere de fisiere – chunkservere (chunk = bloc de date)
- Serverul master central păstrează metadatele despre fisiere
 - namespace, informația de control al accesului, maparea de la fisier la chunk-uri, plus locatiile curente ale chunk-urilor
 - metadatele sunt pastrate în memoria masterului (pentru operare rapidă)

Chunk-uri si operatii

- Fisierele sunt divizate in **chunk-uri** de 64 MB (tipic o operatie read / write transfera **100 MB** de date)
 - dimensiunea mare a chunk-ului **reduce** interactiunea clientilor cu serverul master, dimensiunea metadatelor si overhead-ul retelei (pastreaza o legatura TCP pentru mai multe operatii pe chunk)
 - fiecare chunk este identificat de un **chunk handle** unic de 64 biti, asignat de **master** la crearea chunk-ului
- **Operatii:** create, delete, open, close, read, write files.
 - uzual, operatiile pe inregistrari sunt de **append** si **read** (extrem de rar de supra-scriere).
 - In plus, operatia **snapshot**: creaza o **copie** a unui fisier sau a unui arbore de directoare, la un cost redus
 - operatia **append**: admite multiple apendd-uri concurente la acelasi fisier, cu garantia atomicitatii fiecarui append

Arhitectura GFS



- Periodic, **masterul** schimba cu fiecare **chunkserver** mesaje **HeartBeat** pentru a trimite **instructiuni** si a primi informatii de **stare**.
- Codul **GFS client** legat cu fiecare aplicatie implementeaza API-ul sistemului de fisiere si comunica cu **masterul** si **chunkservere** pentru a citi si scrie date in numele aplicatiei
- **client** si **chunkserver** nu salveaza date in cache (dimensiune mare a datelor) → nu este nevoie de “cache coherence”

Executia unei operatii read-write

- Clientul trimite masterului o cerere cu un chunk index (obtinut din **file name + byte offset**, specificate de aplicatie, si cunoscand **chunk size**)
- Masterul raspunde cu **chunk handle** si **locatiile** replicilor
- Clientul memoreaza informatia
- Clientul trimite cererea (**chunk handle + byte range** in chunk) uneia din replici
- Urmatoarele citiri folosesc informatia memorata de client (pana la expirare)
- Obs. Clientul poate cere **mai multe chunk-uri** odata

Caracteristici GFS

- Considera că defectarea componentelor este **normă**, nu excepție
- **Optimizat** pentru fisiere voluminoase (TBs), operatii **append** concurente și **read** (de regula secventiale)
- Oferă **toleranță la defectari** prin monitorizare, replicarea datelor importante (trei replici), recuperare automata rapidă
- Oferă **productivitate (throughput) agregată ridicată** multor cititori și scriitori concurenți care executa o varietate mare de task-uri.
 - realizata prin **separarea** partii de **control**, care trece prin master, de **transferul datelor**, facut direct intre chunk servers si clienti
- Master centralizat simplu, care nu este o gătuire
 - **Implicitarea** masterului in operatii comune este **minimizata** de **dimensiunea mare a chunk-urilor** si prin **delegarea** autoritatii pentru operatii pe date (**chunk leases**) catre replicile primare

Framework de aplicatii distribuite in cloud

- MapReduce = model de programare si implementarea asociata, pentru prelucrarea si generarea volumelor mari de date
- Programele sunt paralelizate automat si sunt executate pe un cluster de masini obisnuite
 - Model introdus de Google
 - Proiectul “open source” Hadoop MapReduce este inspirat de Google
- Utilizatorul specifica
 - o functie **map** care prelucreaza o pereche key/value si genereaza un set intermediar de perechi key/value
 - o functie **reduce** care fuzioneaza (merges) toate valorile intermediare asociate cu aceeasi cheie intermediara

Exemplu map: numarare cuvinte

Problema: gasire numar de aparitii al fiecarui cuvant intr-o colectie voluminoasa de documente

map(String key, String value):

// **key**: document name

// **value**: document contents

for each word w in value:

EmitIntermediate(w, "1");

- **map** emite fiecare cuvant si un contor de aparitii
- biblioteca **MapReduce** grupeaza toate valorile intermediare asociate cu aceeasi cheie si le paseaza functiei **reduce**

Exemplu reduce: numarare cuvinte

reduce(String key, Iterator values):

```
// key: a word  
// values: a stream of counts  
int result = 0;  
for each v in values:  
    result += ParseInt(v);  
Emit(AsString(result));
```

- **reduce** accepta o cheie intermediara si un set de valori pentru acea cheie
- valorile intermediare sunt date lui **reduce** printr-un **Iterator**
- **reduce** insumeaza valorile corespunzatoare cheii

Tipuri

- exemplul precedent este scris in termenii unor intrari si iesiri de tip **string**
- functiile **map** si **reduce** date de utilizator au asociate **tipuri diverse**

$$\begin{array}{ccc} \text{map } (\text{k1},\text{v1}) & \rightarrow & \text{list}(\text{k2},\text{v2}) \\ \text{reduce } (\text{k2},\text{list}(\text{v2})) & \rightarrow & \text{list}(\text{v2}) \end{array}$$

- care respecta urmatoarele reguli
 - cheile si valorile **input** (parametrii map) sunt dintr-un domeniu **diferit** de cel al cheilor si valorilor **output** (rezultat reduce)
 - cheile si valorile **intermediare** sunt din **acelasi** domeniu ca al cheilor si valorilor **output**

Implementare

- Depinde de mediul de executie
- Ex. Google: **clustere mari** de PCs obisnuite conectate prin Ethernet
 - **masinile** sunt dual-processor x86, cu Linux, 2-4 GB de memorie per masină
 - **retea** cu viteza de 100 megabits/sec sau 1 gigabit/sec
 - un **cluster** are sute sau mii de masini (care se pot defecta)
 - **storage** – discuri ieftine IDE (Integrated Drive Electronics) atasate la masini individuale + sistem de fisiere distribuit + replicare
- Utilizatorii submit **job-uri** unui sistem de **scheduling**
 - fiecare **job** consta dintr-un set de **task-uri**
 - si este **mapat** de scheduler pe un set de **masini** disponibile din cluster

Executia programului

- Programele sunt **paralelizate automat** si execute
- Invocarile **map** sunt distribuite de un **master** mai multor masini prin **partitionarea automata** a **datelor** de intrare intr-un set de **M splituri**
 - spliturile pot fi procesate in **paralel** de masini diferite
- Invocarile **reduce** sunt distribuite prin **partitionarea** spatiului de **chei intermediare** in **R bucati**, folosind o functie de partitionare (ex., $\text{hash}(\text{key}) \bmod R$).
 - numarul de partitii (**R**) si functia de partitionare sunt specificate de utilizator

MapReduce – pe scurt

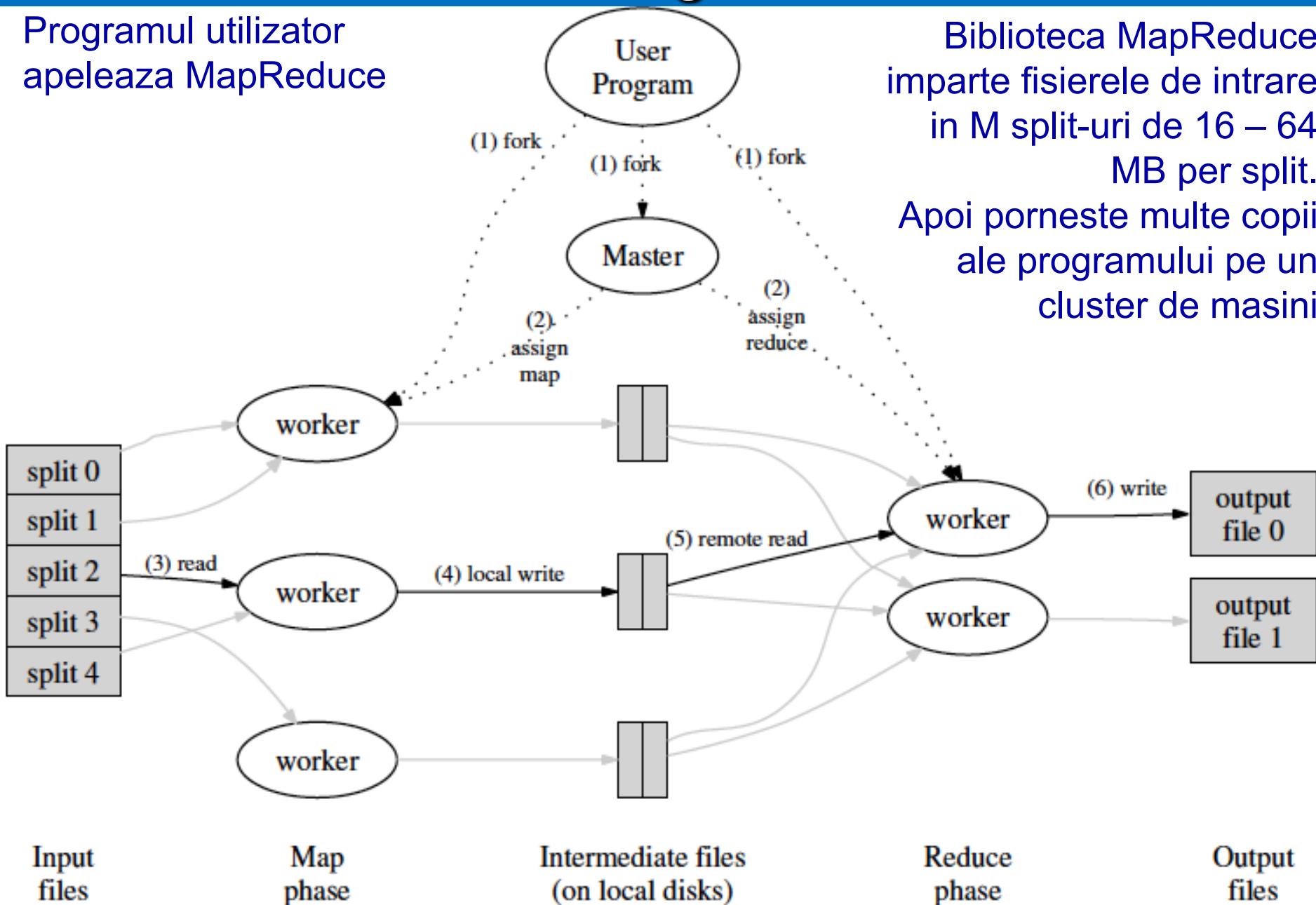
- **Master** distribuie taskuri “map” si “reduce” worker-ilor liberi
- Fiecare worker “**map**”
 - citeste datele de intrare dintr-un **split input**
 - aplica functia “map” fiecarei perechi key/value
 - produce perechi intermediare key/value
- Fiecare worker “**reduce**”
 - citeste si sorteaza dupa cheie perechi intermediare key/value
 - aplica “reduce” fiecarei chei si valori asociate
 - adauga rezultatul la un fisier output
- Cand toate task-urile “map” si “reduce” termina, masterul notifica utilizatorul

Executia Programului - 1

Programul utilizator apeleaza MapReduce

Biblioteca MapReduce imparte fisierile de intrare in M split-uri de 16 – 64 MB per split.

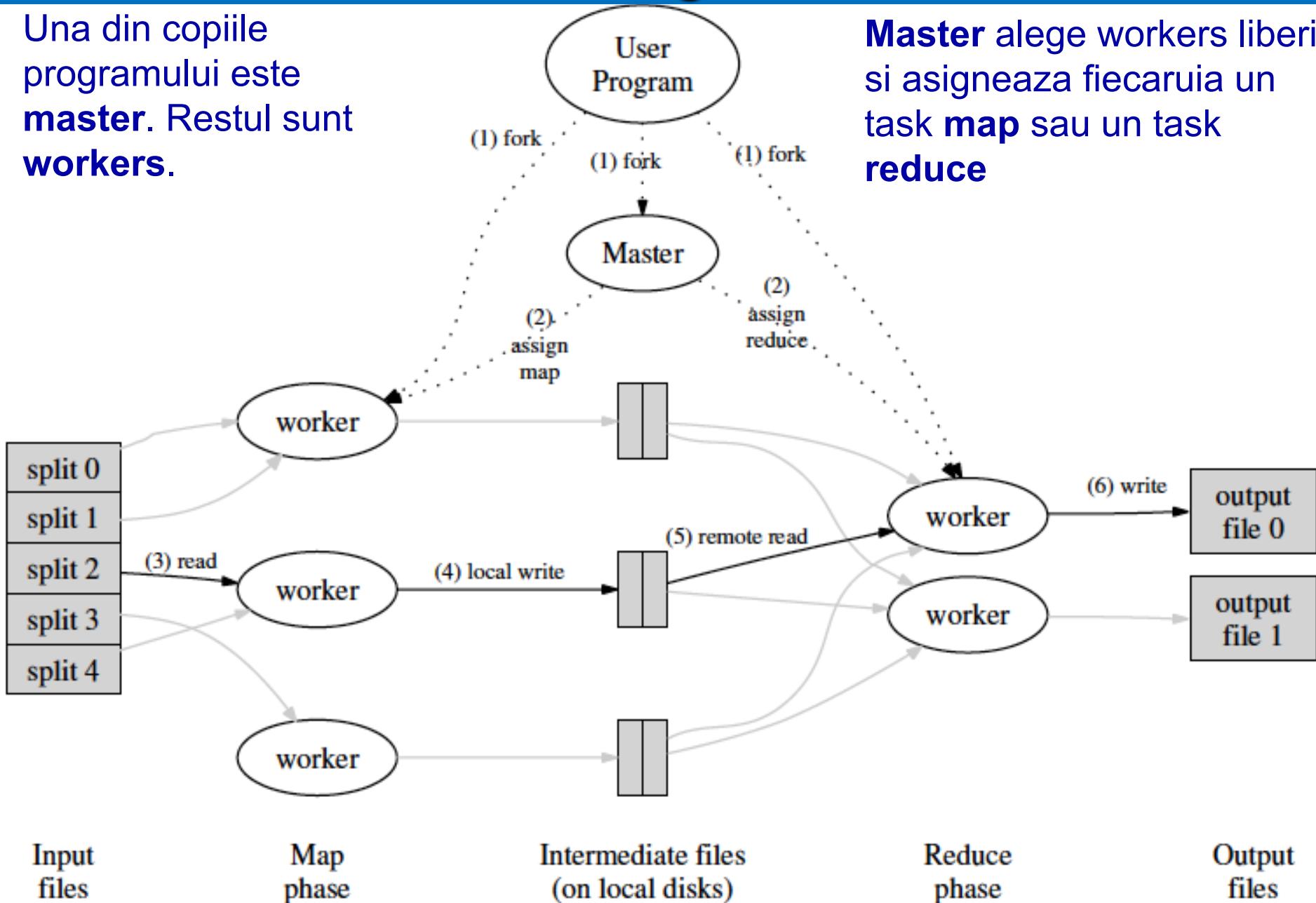
Apoi porneste multe copii ale programului pe un cluster de masini



Executia Programului - 2

Una din copiile programului este **master**. Restul sunt **workers**.

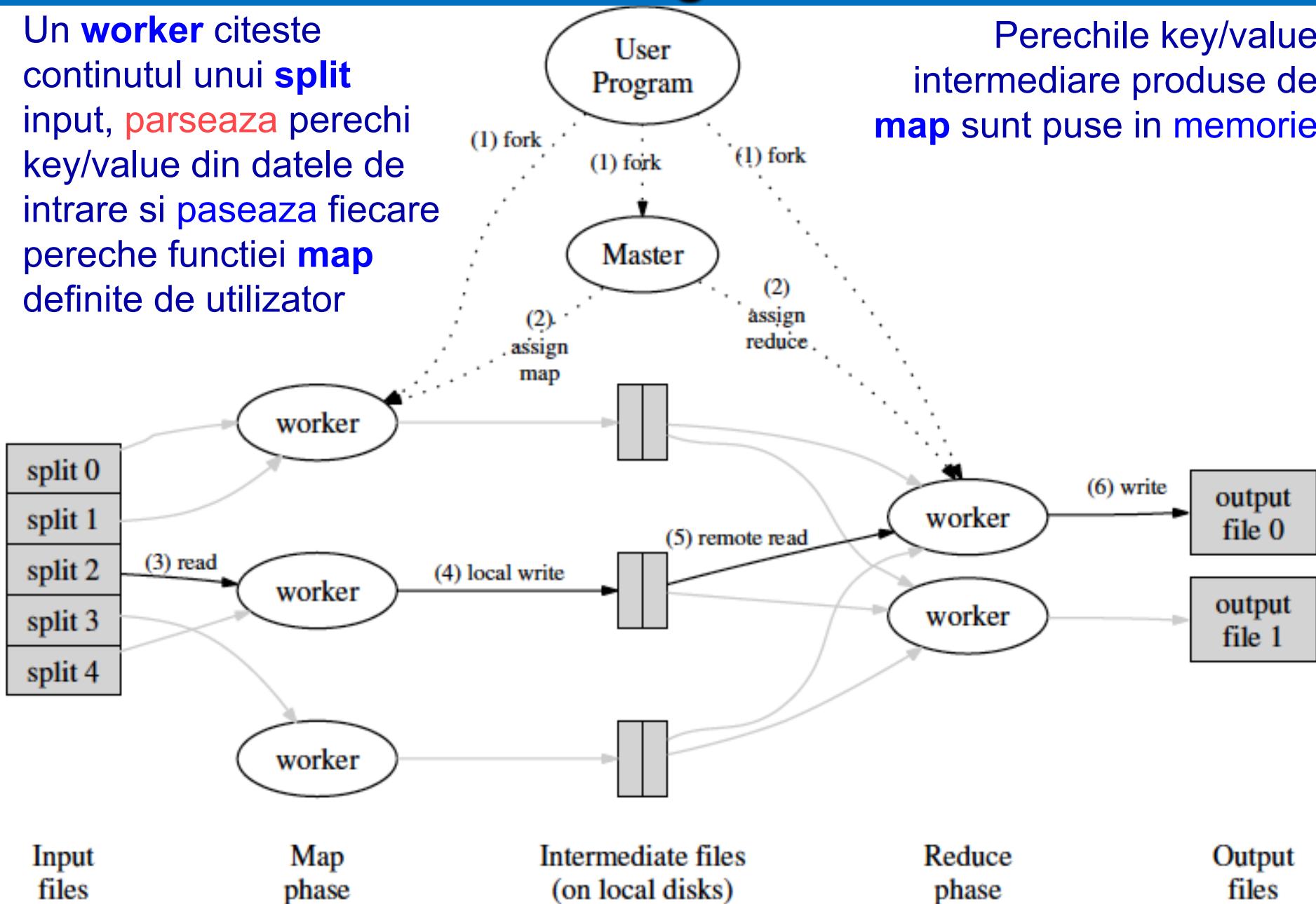
Master alege workers liberi si asigneaza fiecaruia un task **map** sau un task **reduce**



Executia Programului - 3

Un **worker** citeste continutul unui **split** input, **parseaza** perechi key/value din datele de intrare si paseaza fiecare pereche functiei **map** definite de utilizator

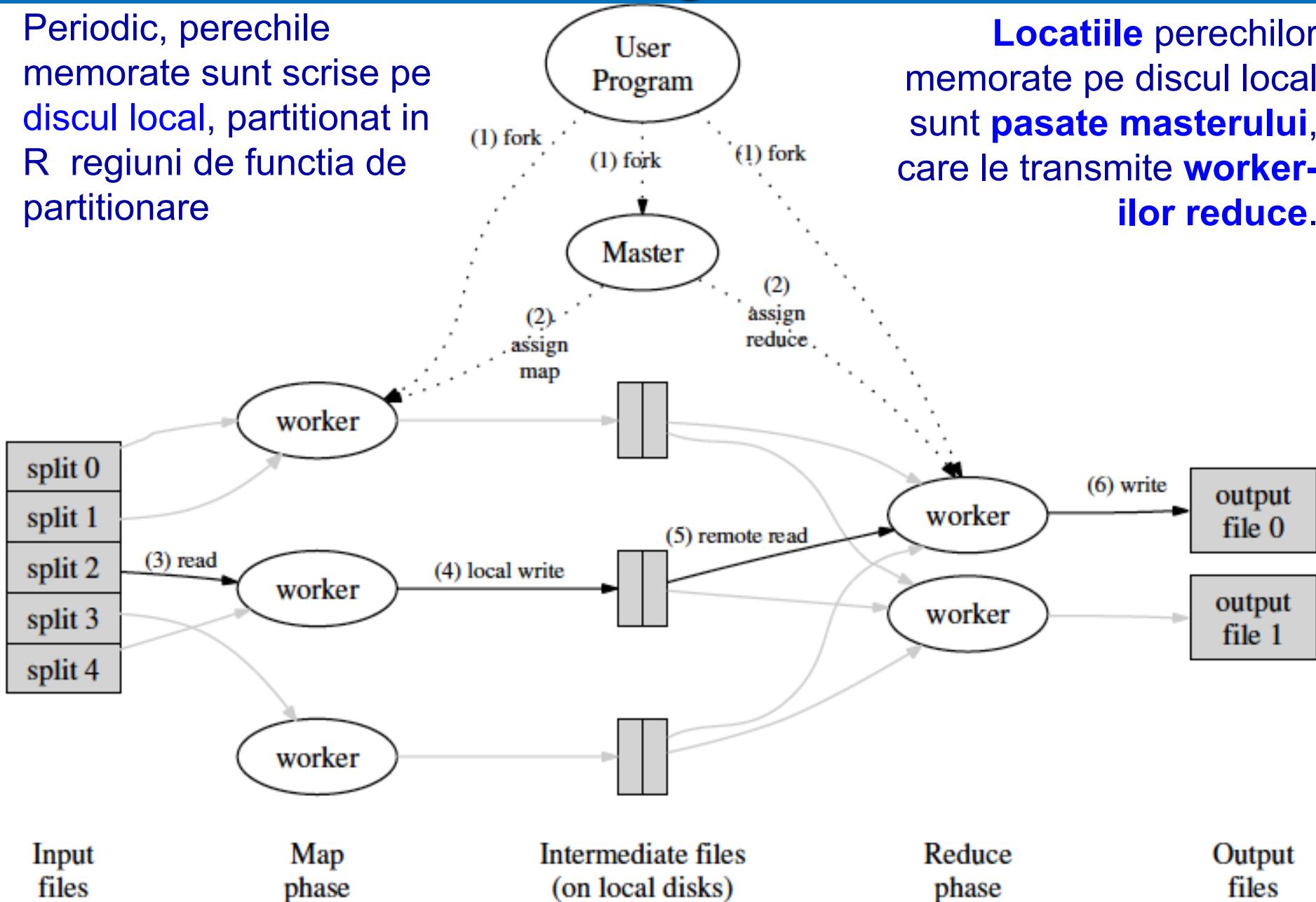
Perechile key/value intermediare produse de **map** sunt puse in memorie



Executia Programului - 4

Periodic, perechile memorate sunt scrise pe discul local, partitionat in R regiuni de functia de partitionare

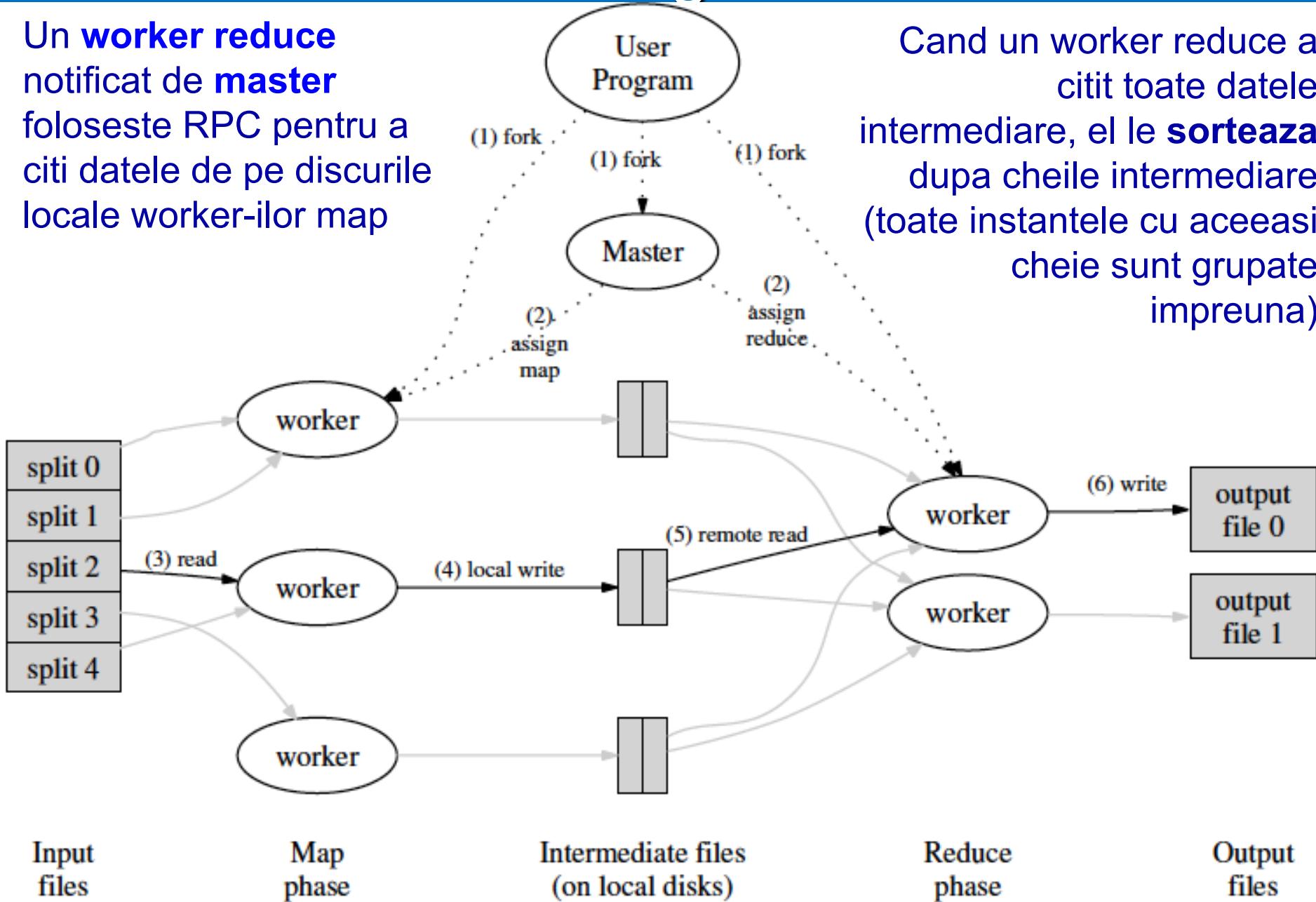
Locatiile perechilor memorate pe discul local sunt **pasate masterului**, care le transmite **worker-ilor reduce**.



Executia Programului - 5

Un **worker reduce** notificat de **master** foloseste RPC pentru a citi datele de pe discurile locale worker-ilor map

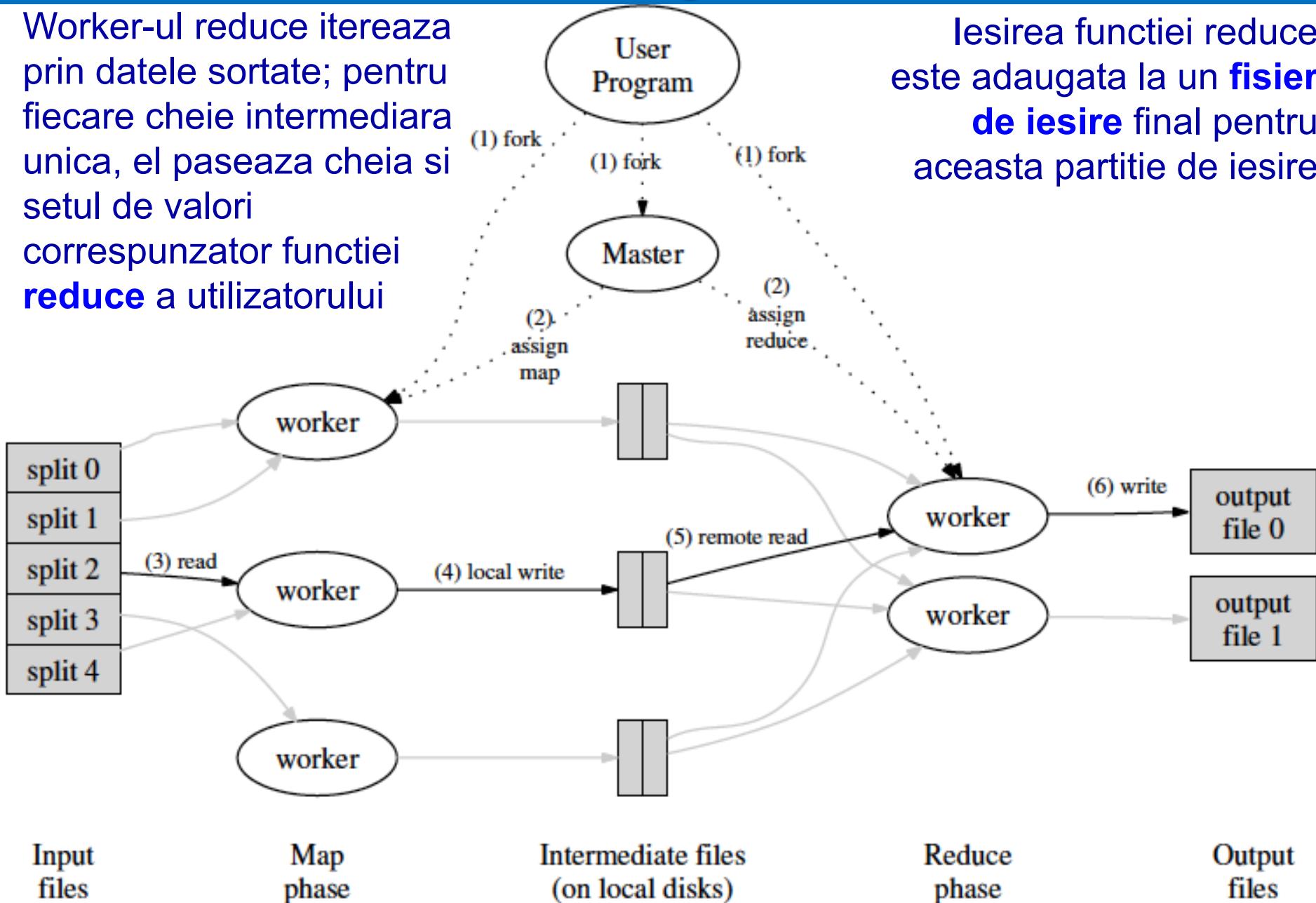
Cand un worker reduce a citit toate datele intermediare, el le **sorteaza** dupa cheile intermediare (toate instantele cu aceeasi cheie sunt grupate impreuna)



Executia Programului - 6

Worker-ul reduce itereaza prin datele sortate; pentru fiecare cheie intermediara unica, el paseaza cheia si setul de valori corespunzator functiei **reduce** a utilizatorului

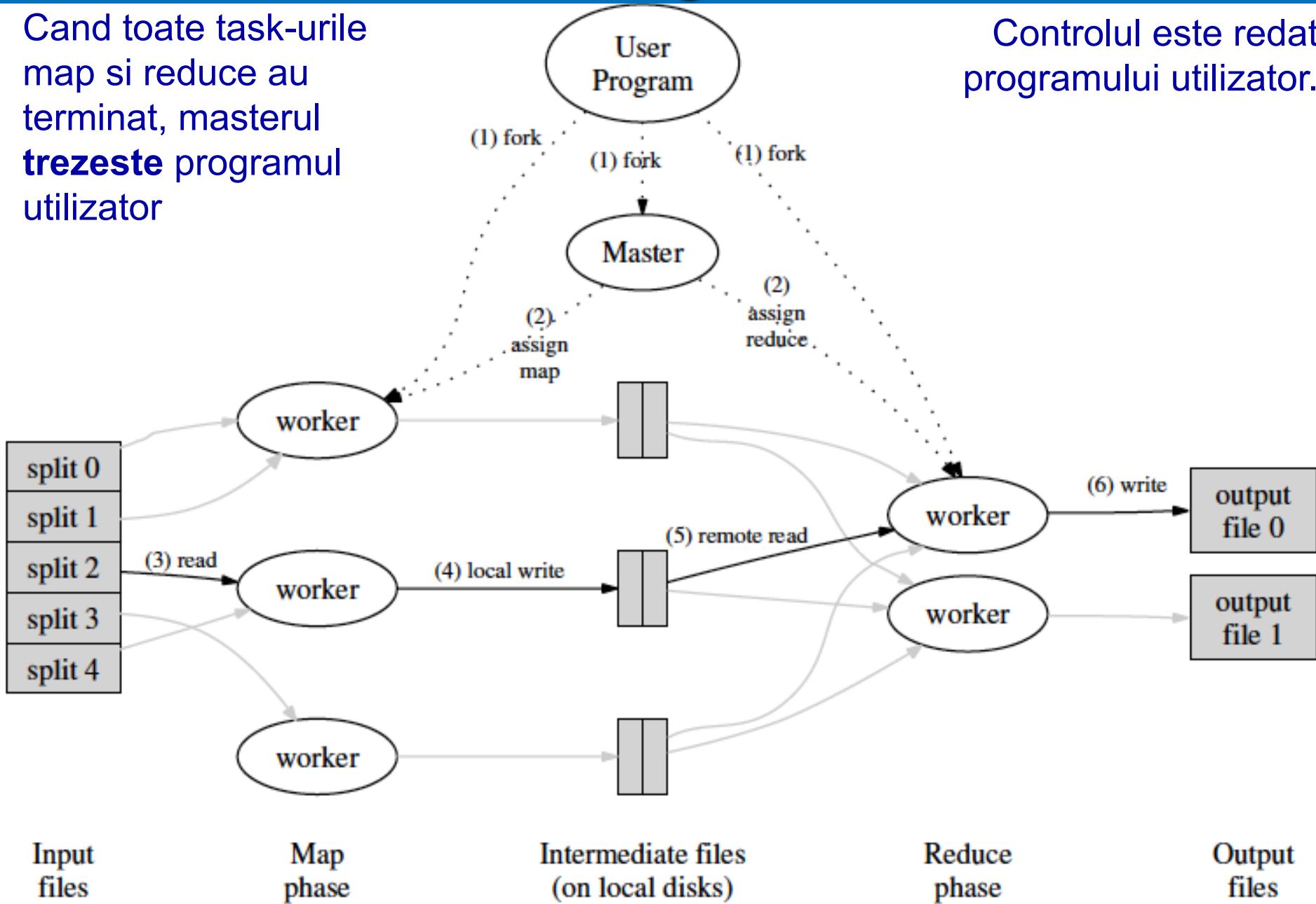
Iesirea functiei reduce este adaugata la un **fisier de iesire** final pentru aceasta partitie de iesire



Executia Programului - 7

Cand toate task-urile map si reduce au terminat, masterul **trezeste** programul utilizator

Controlul este redat programului utilizator.



Toleranta la defectari

- **Master** trimite periodic fiecarui worker mesaje **ping**; daca nu primeste raspuns marcheaza taskul ca **defect**
- task-urile **map terminate** de worker sunt trecute in starea initiala **idle** si devin eligibile pentru planificare pe un alt worker
 - rezultatul produs de el este memorat pe **discurile locale** ale masinii defecte si este **inaccesibil**
 - workerii care executa taskuri **reduce** sunt instiintati de re-executie
- taskurile **reduce terminate** nu trebuie re-executate deoarece iesirea lor este stocata intr-un sistem de fisiere global
- orice task **map** sau **reduce** care erau **in executie** pe un worker care se **defecteaza** este resetat la **idle** si devine eligibil pentru re-planificare

Toleranta la defectari (2)

- Master pastreaza mai multe **structuri de date** pentru taskurile map si reduce
 - **starea** (idle , in-progress, completed)
 - **identitatea** masinii worker (pentru taskuri non-idle)
- Master scrie **checkpoint-uri** periodice ale structurilor de date pastrate de el
 - daca taskul master se defecteaza, se poate porni o noua copie de la ultimul checkpoint.

Referinte

Cloud computing: state-of-the-art and research challenges

Qi Zhang · Lu Cheng · Raouf Boutaba

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

END