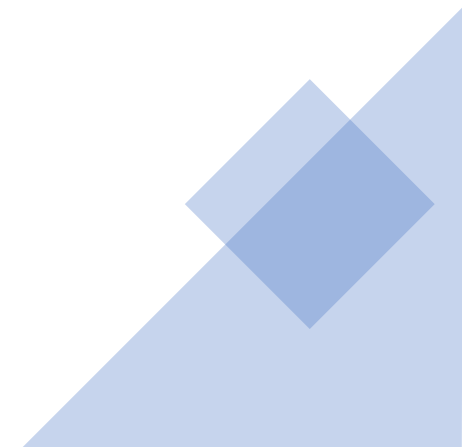


# Map Reduce programski model (revisited)

Računarstvo u oblaku  
(Cloud Computing)



# Sadržaj

- Uvod
  - Motivacija
  - Map Reduce programski model
    - Implementacija
    - Strukture podataka
    - Lokalnost podataka
    - Stragglers
  - Funkcija particionisanja
  - Map Reduce na većim klasterima
  - Scheduling
    - Spekulativni zadaci
  - Unapređenje – LATE scheduler
- 

# Uvod

- Map Reduce smo već spomenuli kao jedan od modela za grupnu obradu
  - Grupna obrada (batch preocesanje)
  - Obradivali ste ga i na NASP
- *“Nova apstrakcija koja nam omogućava da izrazimo jednostavne proračune (obrade) koje smo pokušavali da izvršimo, ali skriva nezgodne detalje paralelizacije, tolerancije kvarova, distribucije podataka i raspoređivanja opterećenja.”*

# Motivacija

- Potreba za mnogim izračunavanjima u odnosu na velike/ogromne skupove podataka:
  - Ulazni podaci: dokumenti, evidencije Veb zahteva
  - Izlazni podaci: obrnuti indeksi, rezime stranica popisanih po domaćinu, skup najčešćih upita u datom danu
- Većina ovih izračunavanja je relativno jednostavna
- Da bismo ubrzali izračunavanje i skratili vreme obrade, možemo da distribuiramo podatke preko 100 mašina i paralelno ih obrađujemo
  - Ali, paralelne račune je teško i složeno za upravljanje: *racing conditions*, otklanjanje grešaka, distribucija podataka, tolerancija kvarova, raspoređivanje opterećenja itd.
- U idealnom bi bilo da paralelno obrađujemo podatke, ali ne i da se bavimo kompleksnošću paralelizacije i distribucije podataka

# Map Reduce programski model

- Ulaz: skup *key/value* parova
- Izlaz: skup *key/value* parova
- Obrada predstavlja izvršavanje dve funkcije:
  1. Zadatak "mapiranja" *Map task*: jedan par  $\rightarrow$  lista parova međurezultata
    - $\text{map}(\text{input-key}, \text{input-value}) \rightarrow \text{list}(\text{out-key}, \text{intermediate-value})$
    - $\langle k_i, v_i \rangle \rightarrow \{ \langle k_{int}, v_{int} \rangle \}$
  1. Zadatak redukovanja *Reduce task*: svi međurezultati sa istim  $k_{int} \rightarrow$  lista izlaznih vrednosti
    - $\text{reduce}(\text{out-key}, \text{list}(\text{intermediate-value})) \rightarrow \text{list}(\text{out-values})$
    - $\langle k_{int}, \{v_{int}\} \rangle \rightarrow \langle k_o, v_o \rangle$

# Map Reduce programski model

- Primer : funkcije za prebrojavanje broja pojava neke reči u dokumentu

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

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

# Map Reduce implementacija

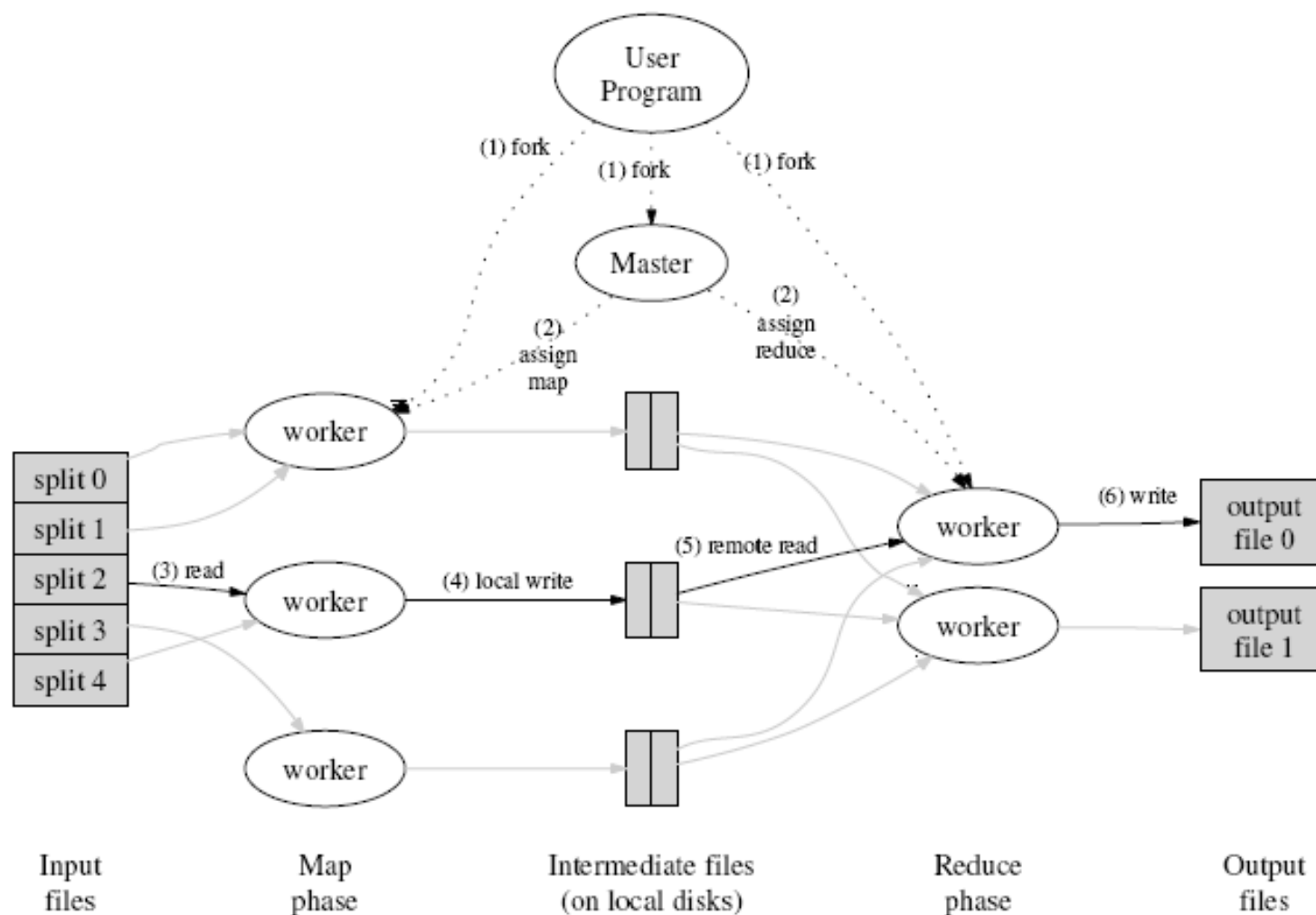
- Opisana u radu:
- Odgovara tadašnjoj implementaciji na tadašnjoj Googleovoj arhitekturi:
  1. Veliki klaster standardnih (*commodity*) PC-ja povezan preko *switched Ethernet*
  2. Tipično dvoprocesorska x86 arhitektura, Linux, 2-4GB of memorije! (za današnje standard skromne mašine)
  3. Klaster mašina – otkazi očekivani
  4. Smeštanje podataka - Google File System (GFS - 2003) na IDE diskovima na PC-ima. GFS – distribuirani FS, koristi replikaciju kako bi postigao dostupnost i pouzdanost.
    - Fajlovi podeljeni na blokove predefinisane veličine (tipičo 16-64 MB)
    - Svaki blok replicira se uobičajeno tri puta
- Sistem raspoređivanja:
  1. Korisnik postavi “posao”
  2. Posao se izdela na zadatke, raspoređivač dodeljuje zadatke pojedinačnim računarima

# Map Reduce implementacija –paralelizacija posla

- Korisnik definiše:
  - **M**: broj *map* taskova
  - **R**: broj *reduce* zadataka
- Map:
  - MapReduce biblioteka deli ulazne fajlove na M blokova (tipično 16-64MB)
  - Zadatak mapiranja se distribuira na raspoložive čvorove
- Reduce:
  - Prostor ključeva dobijenih u međurezultatima deli se na R delova
  - $\text{hash}(\text{intermediate\_key}) \bmod R$
- Tipično radno okruženje:
  - 2,000 mašina
  - $M = 200,000$
  - $R = 5,000$



# Map Reduce programski model – tok izvršavanja



# Map Reduce – osnovne strukture podataka

- Za svaki map/reduce task:
  - Stanje (status) {*idle, in-progress, completed*}
  - Identitet mašine na kojoj se izvršava (za aktivne - *non-idle* - zadatke)
- Lokacije regiona u kome se čuvaju međureultati se prosleđuje od mapera ka reducerima preko master čvora
- Ova informacija se inkrementalno (kako koji mapper završi posao) dostavlja onim *worker* čvorovima koji imaju *in-progress reduce* zadatke.

# Šta se dešava u slučaju otkaza čvora

## Dva tipa otkaza:

### 1. Otkazi worker čvorova:

- Identifikuju se preko “otkucaja srca” koji se redovno šalju s *master* čvora. Ako odgovor ne stigne u predviđenom roku – master smatra da je čvor otkazao.
- *In-progress* i *completed* map zadaci se preraspoređuju → *idle*
- *In-progress* reduce zadaci se preraspoređuju → *idle*
- Radni čvorovi (*worker*) koji izvršavaju reduce taskove koji su afektirani otkazom map čvora se obaveštavaju o preraspoređivanju
- Pitanje: Zašto se preraspoređuju i map zadaci koji su u statusu *completed*?
- Odgovor: izlaz mapera se čuva na lokalnom fs čvora koji je radio mapiranje (za razliku od finalnog rezultata koji se smešta na GFS)

### 2. master failure:

1. Retki
2. Oporavak moguć iz “checkpointa”
3. Rešenje: prekini trenutnu obradu i počni iznova

# Lokalnost podataka

- Mrežni komunikacija - propusni opseg je relativno “redak” resurs, a l povećava kašnjenje
- Cilj je štedeti mrežni propusni opseg
- Korišćenje GFS tipično zahteva pisanje tri kopije na tri različite mašine
- Zadaci mapiranja se raspoređuju tako da su blizu podacima
  - Na čvorovima na koji su podaci već lokalno smešteni
  - Ako to ne može, na čvoru koji su mrežno “blizu” samim podacima (e.g., isti switch)

# Granulacija zadatka

- Broj map zadatka  $>$  broja radnih čvorova
  - Bolji load balancing
  - Bolji oporavak
- Ali povećava posao za master čvor
  - Mora obaviti više raspoređivanja (scheduling)
  - Mora snimiti više *state* informacija
- **M** se može birati tako da se uzima u obzir veličina blokova podataka
  - Kako bi se povećala “lokalnost” podataka
- **R** obično specificira korisnik
  - Svaki *reduce* task kreira izlazni fajl

# Čvorovi sa problemima u obradi - stragglers

- Spori radni čvorovi izazivaju kašnjenje cele ovrade → **stragglers**
  - Problemi sa diskovima (diskovi sa *soft errors*)
  - Drugi zadaci na mašini oduzimaju resurse
  - Neodgovarajuće konfigurisane mašine
- Kada je MapReduce obrada blizu kraja **master** raspoređuje i spekulativne rezervne (*backup*) obrade preostalih in-progress zadataka. Zadatak se potom označava završenim kada ili primarni ili rezervni čvor prijave završetak, koji god to uradi pre.
- Empirijski: operacije sortiranja traju I do 44% duže ako neka rezervnih čvorova.

# Poboljšanja – funkcija particionisanja

- Funkcija za particionisanje određuje reduce zadatak
  - Korisnik specificira  $R$  (broj izlaznih fajlova)
  - Ali moguće je da postoji više od  $R$  ključeva u međurezultatima
  - Iskoristiti same ključeve i  $R$
  - Podrazumevana:  $\text{hash}(\text{key}) \bmod R$
- Bitno je izabrati dobro balansiranu funkciju particionisanja:
  - $\text{hash}(\text{hostname}(\text{urlkey})) \bmod R$
  - Za ključeve koji su url

# Poboljšanja – funkcija za kombinovanje

- Uvodi se *mini-reduce* faza pre nego se međurezultati pošalju na reduce deo
- Kada ima velikog ponavljanja ključeva međurezultata
  - Spaja vrednosti po ključevima međurezultata pre nego što se pošalju u reduce
  - Primer: brojanje reči, veliki broj rezultata u formi <word\_name, 1>. Spojiti sve međurezultate sa istim word\_name
  - Impelemntacija po logici slična reducer zadatku
- Štedi mrežnu komunikaciju



# Map Reduce na velikim klasterima

- Da li je originalni model zadovoljavajući i na današnjim velikim i heterogenim klasterima

Diskusija u radu: “Improving MapReduce Performance in Heterogeneous Environments”, By Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz and Ion Stoica, published in Usenix OSDI conference, 2008

- Map Reduce je postao vrlo popularan
  - Open-source implementacija, Hadoop, koriste Yahoo!, Facebook, Last.fm, ...
  - Opseg korišćenja: 20 PB/dan Google, O(10,000) čvorova Yahoo, 3000 poslova/dan Facebook

# Map Reduce na velikim klasterima

- Problem sa **strugglers**
- Da li je osnovni način mitigacije problema – spekulativne rezervne kopije – dovoljan i u velikim heterogenim okruženjima
  - Glavni problem za Hadoop na EC2 instancama je šarolikost performansi pojedinih čvorova, što nepovoljno utiče na pretpostavke s kojima raspoređivač radi pri dodeli zadataka
  - Noviji LATE raspoređivač može da skрати ukupno vreme odziva za 50%

# Map Reduce raspoređivanje zadataka - scheduling

- Kada postoji slobodno vreme na nekom čvoru, Hadoop bira sledeći task koji će mu dodeliti po prioritetu iz sledeće tri kategorije:
  1. Zadatku koji je otkazao dodeljuje se viši prioritet
  2. Još neraspoređeni zadaci – za mapiranje, zadaci koji imaju lokalne podatke se biraju prvi.
  3. Traži spekulativne backup zadatke koji čekaju izvršavanje.

# Map Reduce – standardno odlučivanje o raspoređivanju spekulativnih zadataka

- Koje zadatke treba spekulativno izvršavati?
- Hadoop nadgleda napredovanje zadataka koristeći meru napretka *progress score*: broj od 0, ..., 1
- Za mapere: Skor je jednak udelu učitanih podataka
- Za reducere: Izvršavanje se deli u tri jednake faze, od kojih svaka dodaje 1/3 ukupnog skora:
  - Faza kopiranja: udeo map rezultata koji su već učitani
  - Faza sortiranja – izlazi iz mapiranja se sortiraju po ključevima – procenat podataka koji su već spojeni
  - Faza redukovanja: udeo podataka koji su prošli kroz funkciju redukovanja
- Example: a task halfway through the copy phase has  
progress score =  $1/2 * 1/3 = 1/6$ .
- Example: a task halfway through the reduce phase has

# Map Reduce – standardno odlučivanje o raspoređivanju spekulativnih zadataka

- Primer: zadatak koji je na pola faze kaopiranja ima ukupan skor:

$$1/2 * 1/3 = 1/6.$$

- Primer: zadatak koji je na pola faze redukovanja ima skor:

$$1/3 + 1/3 + 1/2 * 1/3 = 5/6$$

# Map Reduce – standardno odlučivanje o raspoređivanju spekulativnih zadataka

- Hadoop traži srednju vrednost progress indikatora za svaku od kategorija mapera i reducera i zatim definiše prag (*threshold*):
- **Kada je progress nekog zadatka manji od srednja\_vrednost - 0.2, i izvršavao se minimum 1 minut, označava se kao straggler:**

$$\text{threshold} = \text{avgProgress} - 0.2$$

- Svi zadaci sa indikaotrom progresa manjim od praga su *struggler*-i
- “Nerešeni” se razrešavaju na osnovu lokalnosti podataka
- Ovakav pristup radi sasvim dobro u homogenim klasterima

# Map Reduce – standardno odlučivanje o raspoređivanju spekulativnih zadataka

- Hadoop traži srednju vrednost progress indikatora za svaku od kategorija mapera i reducera i zatim definiše prag (*threshold*):
- **Kada je progress nekog zadatka manji od srednja\_vrednost - 0.2, i izvršavao se minimum 1 minut, označava se kao straggler:**

$$\text{threshold} = \text{avgProgress} - 0.2$$

- Svi zadaci sa indikaotrom progresa manjim od praga su *struggler*-i
- “Nerešeni” se razrešavaju na osnovu lokalnosti podataka
- Ovakav pristup radi sasvim dobro u homogenim klasterima

# Map Reduce – standardno odlučivanje o raspoređivanju spekulativnih zadataka

Ovakav pristup radi sasvim dobro u homogenim klasterima jer raspoređivač radi na osnovu sledećih pretpostavki:

1. Čvorovi mogu obavljati posao približno istom brzinom
2. Zadaci napreduju konstantnom brzinom tokom svog izvršavanja
3. Nema “troškova” pokretanja spekulativnog zadatka
4. Progres pojedinog zadatka je približno jedan njegovom udelu u ukupnom poslu
5. Zadaci imaju tendenciju da se završavaju u “talasima”, pa onaj zadatak koji ima nizak indicator progressa je verovatno usporen
6. Različiti zadaci iste kategorije sadrže okvirno sličnu količinu posla the same amount of work



# Map Reduce – pretpostavke za raspoređivanje u heterogenim okruženjima

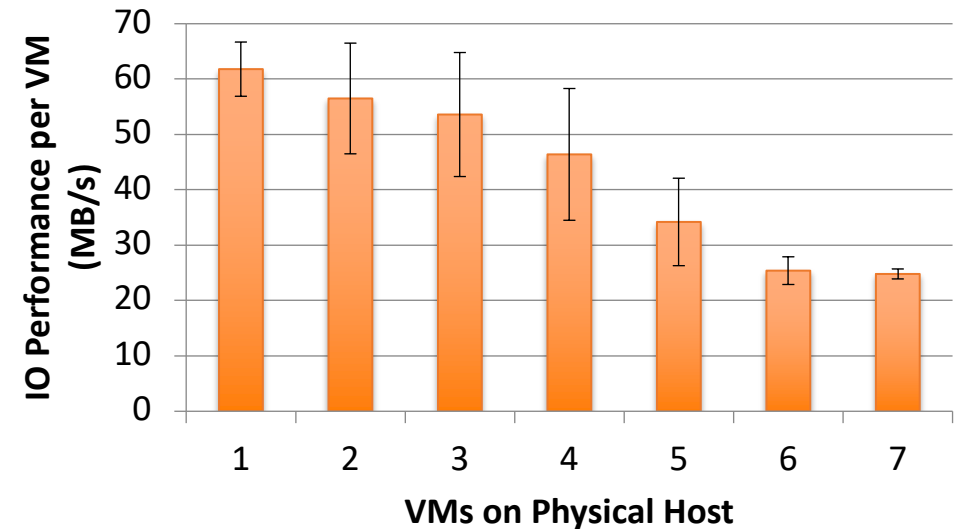
U heterogenim sistemima ne važe ključne pretpostavke o sličnoj brzini čvorova i konstantnoj brzini izvršavanja zadataka :

1. U heterogenim klasterima, neki čvorovi su sporiji od ostalih (starija konfiguracija)
2. Virtualizovani klasteri pate od tzv. interferencije na deljenom resursu (kolokaciona interferencija)

# Heterogenost virtualizovanih okruženja

Tehnologija VM izoluje CPU i memoriju, ali pristup diskovima i mreži je i dalje deljeni resurs

- Puna brzina kada nema istovremenih pristupa sa više VM
- Jednak udeo ako više VM istovremeno pristupa
- Razlika u performansama u tim slučajevima može ići i do 2.5x



# Map Reduce – pretpostavke za raspoređivanje u heterogenim okruženjima (nastavak)

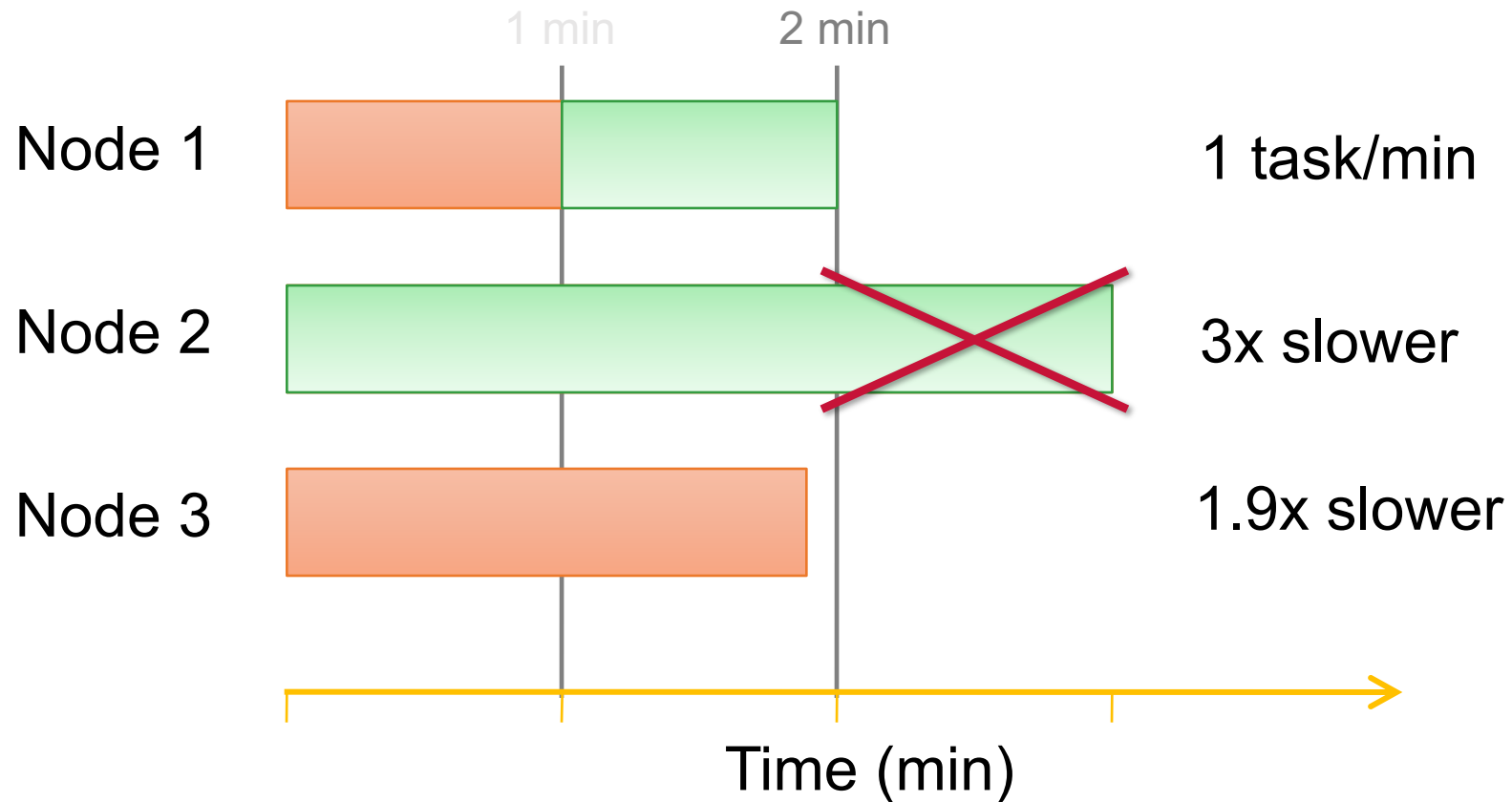
I pretpostavke o nepostojanju troškova spekulativnih izvršavanja, proporcionalnom udelu i završetku u talasima u heterogenim okruženjima nisu realne:

- Previše spekulativnih zadataka oduzima resurse drugim zadacima
- Faza kopiranja kod redukovanja je najsporija, a ona se računa kao 1/3 posla faze redukovanja.
- Zadaci iz različitih “generacija” mogu se izvršavati konkurentno, što dovodi do toga da se noviji, brži zadaci se pri raspoređivanju evaluiraju zajedno sa stariji – srednji indicator progresa se uveliko menja.

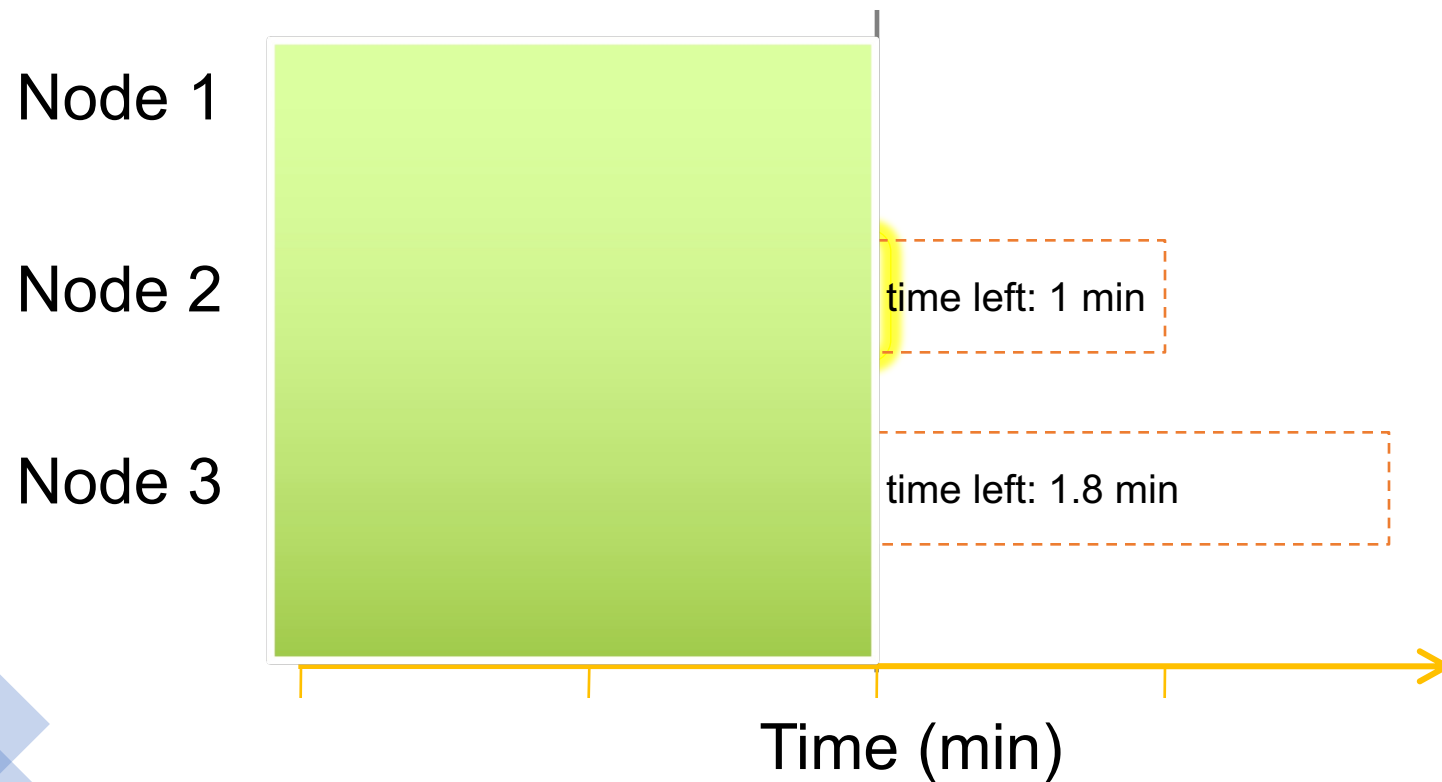
# Progress rate – ideja za bolje raspoređivanje

- Umesto indikatora napretka - **progress score values**, izračunavati stopu progresu - **progress rates**, i napraviti backup zadatke za one koji su dovoljno ispod srednje vrednosti
- Problem: I dalje može selektovati pogrešne zadatke

# Progress rate – ideja za bolje raspoređivanje



# Progress rate – ideja za bolje raspoređivanje



Čvor 2 jeste najsporiji, ali bi mogao da izvršava backup zadatak čvora 3

# LATE raspoređivač

- Ideja: napraviti backup za zadatke kod kojih je procenjeno vreme završetka najduže
  - “Longest Approximate Time to End” → LATE
  - Gledati unapred umesto unazad
- Zaštitne mere:
  - Ograničiti ukupan broj backup zadataka
  - Lansirati backup zadatke na utvrđeno brzim čvorovima
  - Praviti backup samo za one zadatke koji su „dovoljno“ spori

# LATE raspoređivač

- Procena vremena završetka

$$\textit{progress rate} = \frac{\text{progress score}}{\text{execution time}}$$

$$\textit{estimated time left} = \frac{1 - \text{progress score}}{\text{progress rate}}$$

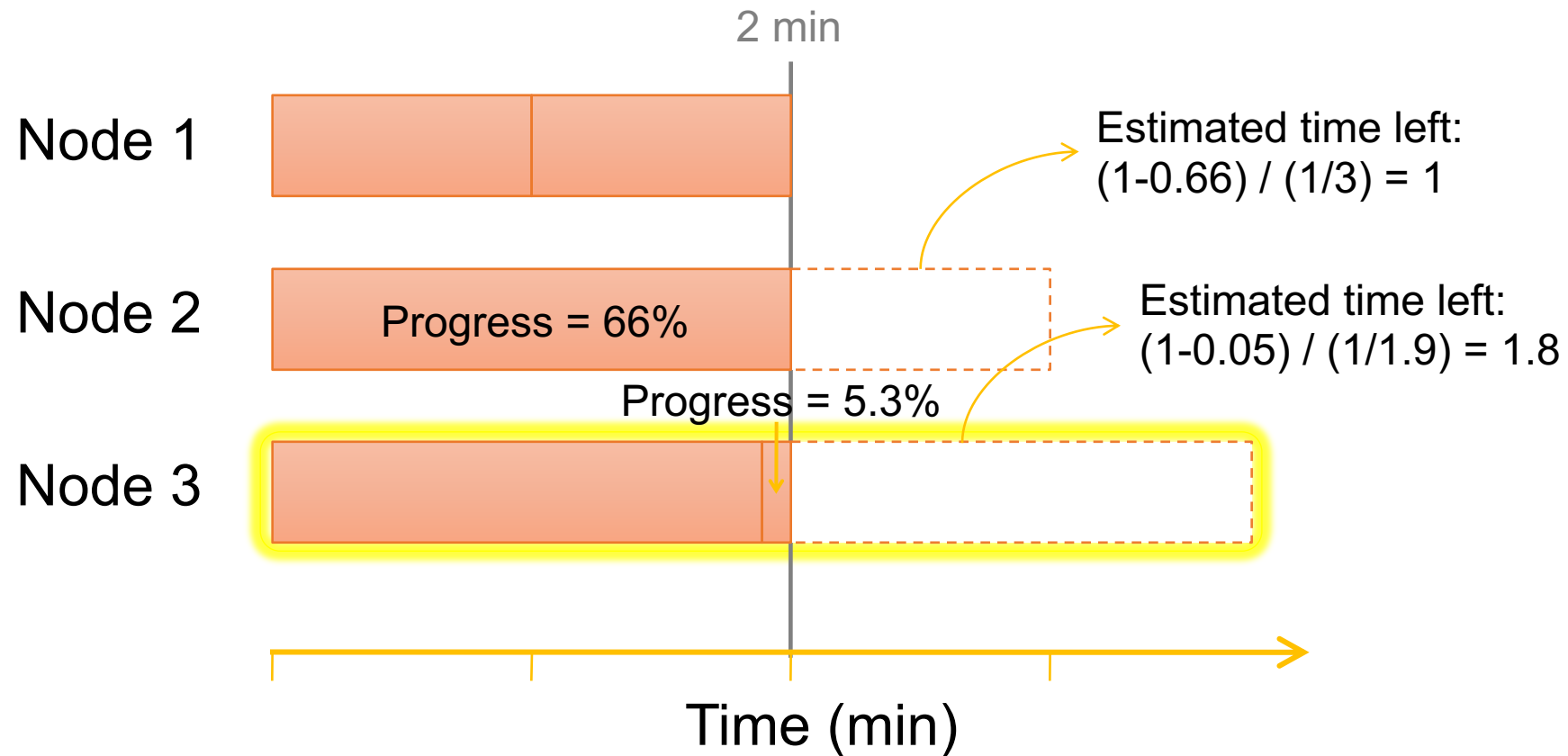


# LATE raspoređivač

- Ukoliko se stvori slobodno mesto za izvršavanje zadatka na čvoru i izvršava se manje od *SpeculativeCap* zadataka, tada:
  1. Ignorirati zahtev za dodelu zadatka ukoliko je ukupan progress na čvoru manji od *SlowNodeThreshold* (=25%)
  2. Rangiraj trenutne, ne spekulativne zadatke koji se izvršavaju, na osnovu preostalog vreme do završetka
  3. Pokreni spekulativnu kopiju najviše rangiranog takvog zadatka koji ima stopu progresa manju od *SlowTaskThreshold* (=25%)
- Pragovi:
  - 10% limit na broj backup zadataka, 25% za spore čvorove/zadatke
  - Validirati pragove analizom

# LATE raspoređivač

- LATE bira zadatak 3

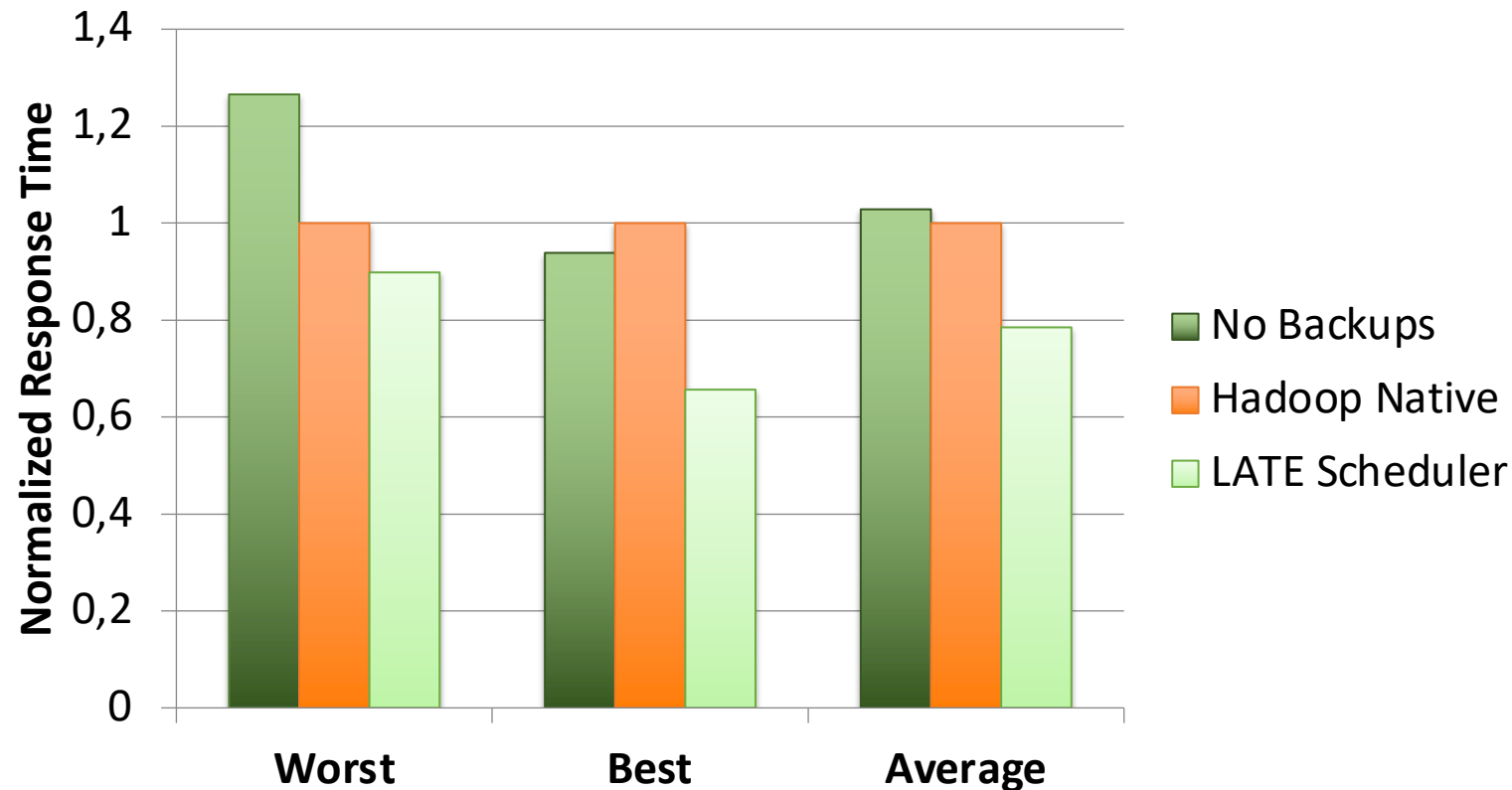


# Evaluacija

- Environments:
  - EC2 (3 job types, 200-250 nodes)
  - Small local testbed
- Self-contention through VM placement
- Stragglers through background processes

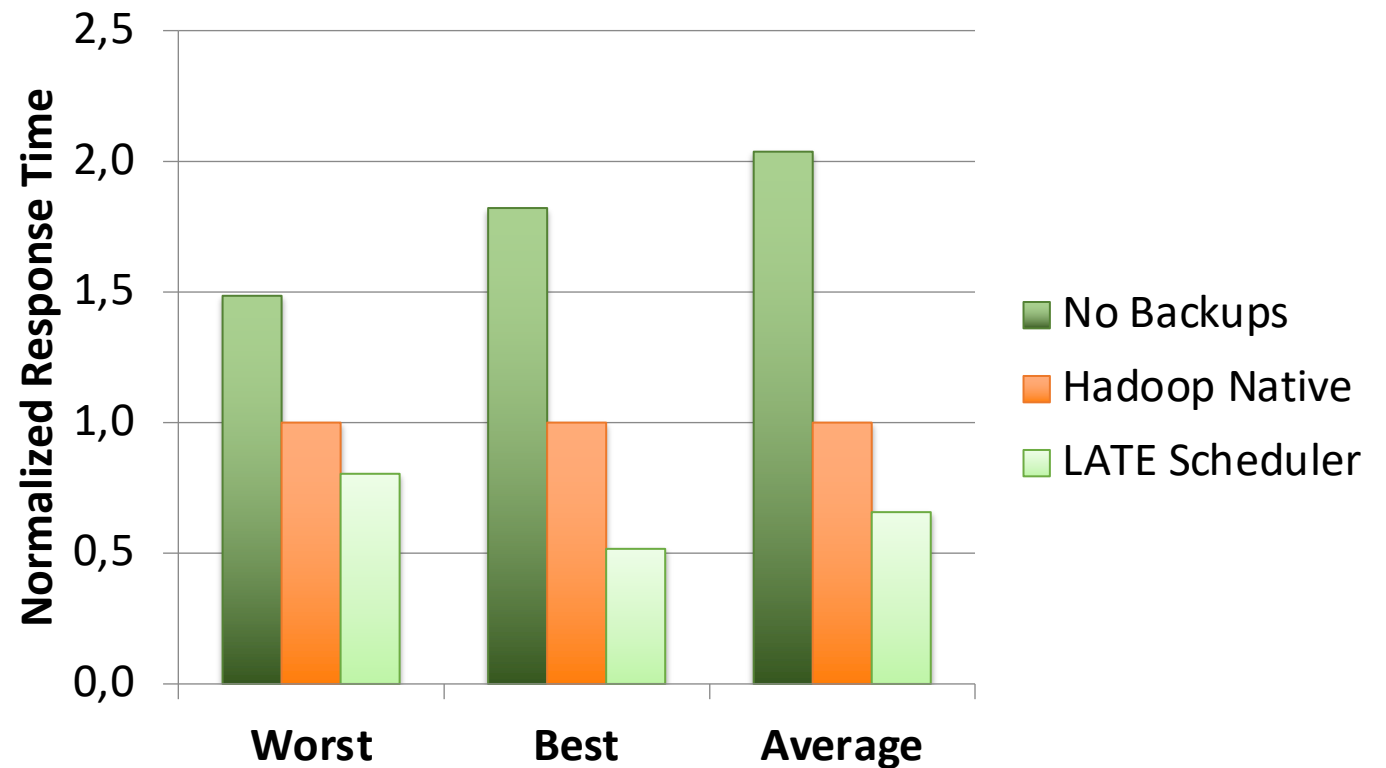
# EC2 Sort without Stragglers (Sec 5.2.1)

- 106 machines , 7-8 VMs per machine → total of 243 VMs
- 128 MB data per host, 30 GB in total
- 486 map tasks and 437 reduce tasks
- average 27% speedup over native, 31% over no backups



# EC2 Sort with Stragglers (Sec 5.2.2)

- 8 VMs are manually slowed down out of 100 VMs in total
- running background of CPU- and disk-intensive jobs
- average 58% speedup over native, 220% over no backups
- 93% max speedup over native



# Zaključak

- Map Reduce – popularan programski model, omogućava ubrzavanje obrade velikih skupova podataka
- Heterogena okruženja – izazov za originalni algoritam raspoređivanja koji je pretpostavljao homogeni kluster
- Optimizacije algoritma mogu značajno unaprediti ponašanje u heterogenim okruženjima
- Pitanja?