

# Konsenzus v distribuovaném systému

---

B4B36PDV – Paralelní a distribuované výpočty

- Opakování z minulého cvičení
- Proč bychom mohli chtít řešit konsenzus v DS?
- RAFT algoritmus
- Zadání semestrální úlohy

## Opakování z minulého cvičení

---

<http://goo.gl/a6BEMb>

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu
3. generují částečné uspořádání zpráv
4. určují reálný čas, kdy byla zpráva poslána
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - TRUE
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu
3. generují částečné uspořádání zpráv
4. určují reálný čas, kdy byla zpráva poslána
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - TRUE
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu - TRUE
3. generují částečné uspořádání zpráv
4. určují reálný čas, kdy byla zpráva poslána
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

Ano, vektorové hodiny správně zdetekují porušení kauzality vždy.

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - TRUE
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu - TRUE
3. generují částečné uspořádání zpráv - TRUE
4. určují reálný čas, kdy byla zpráva poslána
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

Ano, všechny dvojice událostí jsou ve vztahu "nastalo po" nebo "je současné"



## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - TRUE
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu - TRUE
3. generují částečné uspořádání zpráv - TRUE
4. určují reálný čas, kdy byla zpráva poslána - FALSE
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

## Jaké vlastnosti mají vektorové hodiny?

1. jsou paměťově náročnější než skalární hodiny - **TRUE**
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu - **TRUE**
3. generují částečné uspořádání zpráv - **TRUE**
4. určují reálný čas, kdy byla zpráva poslána - **FALSE**
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události - **FALSE**

Hodiny nedokáží detekovat kauzalitu (následek nastal v důsledku příčiny), ale dokáží detekovat potenciální kauzalitu (následek mohl nastat v důsledku příčiny, tzn. existuje kauzální cesta od příčiny k následku)

# Které z následujících algoritmů distribuovaného vzájemného vyloučení jsou férové?

1. Centrální server
2. Kruhové splňování
3. Ricart-Agrawalovo vyloučení

# Které z následujících algoritmů distribuovaného vzájemného vyloučení jsou férové?

1. Centrální server - **FALSE**
2. Kruhové splňování
3. Ricart-Agrawalovo vyloučení

Není splněn požadavek zachování pořadí, latence může změnit pořadí.

## Které z následujících algoritmů distribuovaného vzájemného vyloučení jsou férové?

1. Centrální server - FALSE
2. Kruhové splňování - FALSE
3. Ricart-Agrawalovo vyloučení

Není splněn požadavek zachování pořadí, pořadí dané pozicí v kruhu.

Které z následujících algoritmů distribuovaného vzájemného vyloučení jsou férové?

1. Centrální server - FALSE
2. Kruhové splňování - FALSE
3. Ricart-Agrawalovo vyloučení - TRUE

Který z následujících algoritmů distribuovaného vzájemného vyloučení je nejefektivnější z hlediska počtu poslaných zpráv?

1. Centrální server
2. Kruhové splňování
3. Ricart-Agrawalovo vyloučení

Který z následujících algoritmů distribuovaného vzájemného vyloučení je nejefektivnější z hlediska počtu poslaných zpráv?

1. Centrální server - 3
2. Kruhové splňování
3. Ricart-Agrawalovo vyloučení

Ale centrální server je "single point of failure".



## Který z následujících algoritmů distribuovaného vzájemného vyloučení je nejefektivnější z hlediska počtu poslaných zpráv?

1. Centrální server - 3
2. Kruhové splňování - od 0 do n
3. Ricart-Agrawalovo vyloučení

Token koluje i když nikdo přístup do kritické sekce nepotřebuje, vysoké synchronizační zpoždění, tzn. čas mezi přístupy dvou po sobě jdoucích procesů do kritické sekce ( $O(n)$ ).

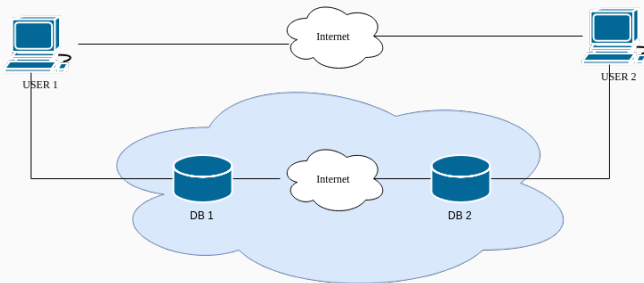
Který z následujících algoritmů distribuovaného vzájemného vyloučení je nejefektivnější z hlediska počtu poslaných zpráv?

1. Centrální server - 3
2. Kruhové splňování - od 0 do n
3. Ricart-Agrawalovo vyloučení -  $2(n-1)$

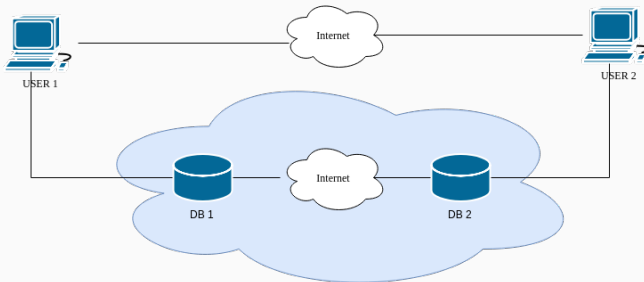
## Konsenzus v distribuovaném světě

---

Uvažujme upravený protokol, kdy klient čeká na potvrzení zápisu...



Uvažujme upravený protokol, kdy klient čeká na potvrzení zápisu...



Co se stane, když server zodpovědný za replikaci selže?

- ⚠ Některé servery mohou znát aktuální data a jiné nikoliv!  
Potřebujeme se shodnout na společné „pravdě“

- ⚠ Některé servery mohou znát aktuální data a jiné nikoliv!  
Potřebujeme se shodnout na společné „pravdě“
- ⚠ Zároveň se nám nesmí ztrácet data, o kterých si klient myslí, že jsou uložena

Ukládání obsahu databáze na disk při každé operaci je drahé...

**⚠** Části databáze si proto raději držíme v paměti

---

<sup>1</sup>do zvláštní sekce disku nebo jiného perzistentního úložiště



Ukládání obsahu databáze na disk při každé operaci je drahé...

**⚠** Části databáze si proto raději držíme v paměti

---

Co když nám server po potvrzení zápisu spadne?

---

<sup>1</sup>do zvláštní sekce disku nebo jiného perzistentního úložiště

Ukládání obsahu databáze na disk při každé operaci je drahé...

⚠ Části databáze si proto raději držíme v paměti

---

Co když nám server po potvrzení zápisu spadne?

**Řešení:** Před potvrzením zápisu požadavek uložíme do logu (žurnálu<sup>1</sup>)!

→ Pokud server spadne, z žurnálu obnovíme ztracená data

Tomu se říká *journaling* nebo *write-ahead logging* a je součástí většiny „rozumných“ databázových systémů.

---

<sup>1</sup>do zvláštní sekce disku nebo jiného perzistentního úložiště

Jak bychom mohli použít journaling v distribuované DB?

Jak bychom mohli použít journaling v distribuované DB?

---

Nemusíme se shodovat na konkrétním obsahu databáze  
(ten může být potenciálně obrovský!)

Stačí, když se shodneme na obsahu žurnálu  
(a doplníme případné chybějící požadavky/záznamy)

## Raft algoritmus

---

Algoritmus pro distribuovanou replikaci  
logů (konzensus)

---



## Algoritmus pro distribuovanou replikaci logů (konzensus)



Cíle:

- Jednoduchost na pochopení
  - Úplnost a použitelnost (pokud je naimplementován tak funguje)
-

## Algoritmus pro distribuovanou replikaci logů (konzensus)



Cíle:

- Jednoduchost na pochopení
- Úplnost a použitelnost (pokud je naimplementován tak funguje)

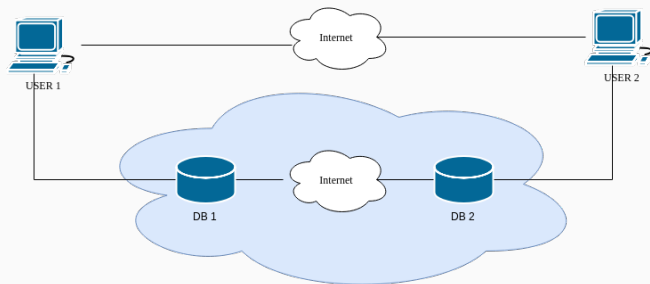
Optimalizovaná verze použitá např. v MongoDB<sup>2</sup>.

---

<sup>2</sup>Slides: How MongoDB replication follows and doesn't the Raft algorithm



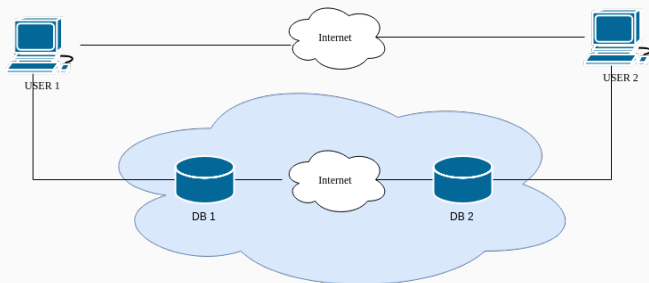
## Jak řešit konkurenční požadavky?



Co když dva klienti budou provádět zápisy současně?  
(potenciálně na různé repliky?)

Možnosti: Dohodnout se na globálním uspořádání

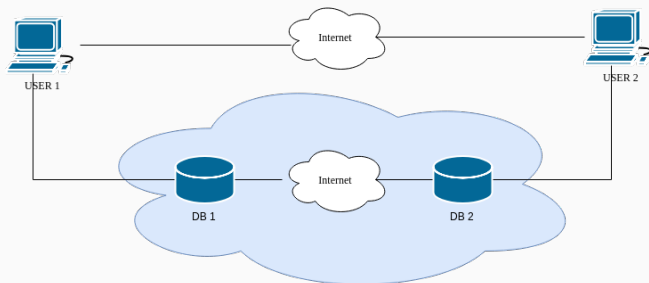
## Jak řešit konkurentní požadavky?



Co když dva klienti budou provádět zápisy současně?  
(potenciálně na různé repliky?)

- Možnosti:
- Dohodnout se na globálním uspořádání
  - Zvolit centrální autoritu, která požadavky seřadí (*leadera*)

## Jak řešit konkurenční požadavky?



Co když dva klienti budou provádět zápisy současně?  
(potenciálně na různé repliky?)

- Možnosti:
- Dohodnout se na globálním uspořádání
  - Zvolit centrální autoritu, která požadavky seřadí (*leadera*)

Co když pak požadavek přijme ne-leader?

Co když pak požadavek přijme ne-leader?

Přepošle uživateli id nodu, o kterém si myslí, že je leader.

Části raftu:

1. Volba leader
2. Replikace logů

## Části raftu:

1. Volba leader
  2. Replikace logů
- 

## Předpokládáme:


- Fixní konfiguraci známou všem nodům, tzn. počet nodů je fixní a známý všem.

<http://thesecretlivesofdata.com/raft/><sup>3</sup>

---

<sup>3</sup>Sources at <https://github.com/benbjohnson/thesecretlivesofdata>



 Leader musí mít v DS autoritu!

Co to znamená?

## Leader musí mít v DS autoritu!

Co to znamená?

→ Musí mít důvěru nadpoloviční většiny serverů

---

### Volba leadera:

1. Pokud si server myslí, že v systému není leader, chce se jím stát sám.  
Kdy si myslí, že v systému není leader?

## ⚠ Leader musí mít v DS autoritu!

Co to znamená?

→ Musí mít důvěru nadpoloviční většiny serverů

---

### Volba leadera:

1. Pokud si server myslí, že v systému není leader, chce se jím stát sám.  
Kdy si myslí, že v systému není leader?
2. Pošle ostatním serverům žádost o to, aby ho respektovali  
(zpráva `RequestVote`)
3. Stane se kandidátem

## ⚠ Leader musí mít v DS autoritu!

Co to znamená?

→ Musí mít důvěru nadpoloviční většiny serverů

---

### Volba leadera:

1. Pokud si server myslí, že v systému není leader, chce se jím stát sám.  
Kdy si myslí, že v systému není leader?
2. Pošle ostatním serverům žádost o to, aby ho respektovali  
(zpráva **RequestVote**)
3. Stane se **kandidátem**
4. Pokud s tím většina serverů souhlasí, stane se leaderem

Přišel Vám RequestVote...

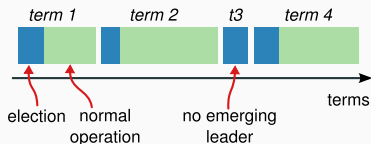
Kdy budete kandidáta volit?

Přišel Vám RequestVote...

Kdy budete kandidáta volit?

Dvě podmínky:

- Požadavek kandidáta musí být aktuální podle „logického času“



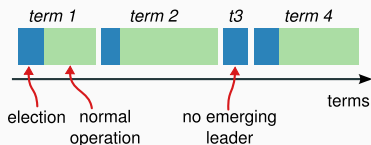
Aktuální **term** posíláme v každé zprávě.  
`currTerm = max(currTerm, msg.term)`

Přišel Vám RequestVote...

Kdy budete kandidáta volit?

Dvě podmínky:

- Požadavek kandidáta musí být aktuální podle „logického času“



Aktuální **term** posíláme v každé zprávě.  
`currTerm = max(currTerm, msg.term)`

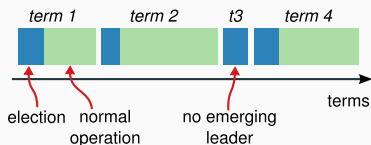
- Log kandidáta obsahuje všechny potvrzené (*committed*) požadavky

Přišel Vám RequestVote...

Kdy budete kandidáta volit?

Dvě podmínky:

- Požadavek kandidáta musí být aktuální podle „logického času“



Aktuální **term** posíláme v každé zprávě.  
 $\text{currTerm} = \max(\text{currTerm}, \text{msg.term})$

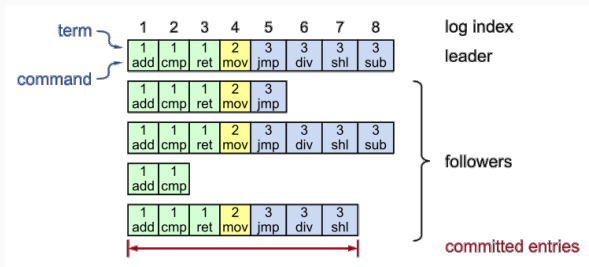
- Log kandidáta obsahuje všechny potvrzené (*committed*) požadavky  
To ale nedokážeme zjistit :-)



# Log („žurnál“)

Jediné, co dokážeme zjistit je, jestli je daný log „aktuálnější“

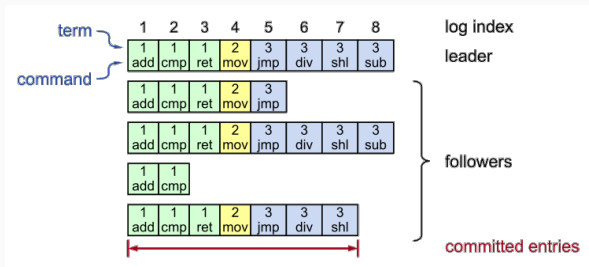
U každého záznamu si držíme **term**, ve kterém byl zapsaný a jeho pozici (**index**)



# Log („žurnál“)

Jediné, co dokážeme zjistit je, jestli je daný log „aktuálnější“

U každého záznamu si držíme **term**, ve kterém byl zapsaný a jeho pozici (**index**)



Log A je aktuálnější než B pokud lexikograficky

$$(A[A.last].term, A[A.last].index) > (B[B.last].term, B[B.last].index)$$

Zajistí požadavek aktuálnějšího logu kandidáta zvolení leadra se všemi potvrzenými záznamy?

Na kolika serverech musí být záznam zapsaný aby byl potvrzený (committed)?

Zajistí požadavek aktuálnějšího logu kandidáta zvolení leadra se všemi potvrzenými záznamy?

Na kolika serverech musí být záznam zapsaný aby byl potvrzený (committed)?

---

**⚠** Stačí nám zapsat záznam na nadpoloviční většinu serverů

- Pokud by měl být zvolený leaderem server, kterému chybí potvrzený záznam, pak bude mít méně „aktuální“ log

Pro zreplicování záznamu v logu zašle leader zprávu `AppendEntries` všem followerům

Pro zreplikování záznamu v logu zašle leader zprávu `AppendEntries` všem followerům

Po obdržení potvrzení od většiny serverů považuje zápis za úspěšný  
... a provede změnu v DB a potvrdí úspěch i klientovi

Pro zreplikování záznamu v logu zašle leader zprávu **AppendEntries** všem followerům

Po obdržení potvrzení od většiny serverů považuje zápis za úspěšný  
... a provede změnu v DB a potvrdí úspěch i klientovi

---

Kdy provedou změnu v DB i followeri?

Pro rekonstrukci stavu z logu, musí být log kompletní!

Nestačí nám tak shoda na zápisu jednoho prvku do logu

→ Musíme se shodnout i na všech předcházejících prvcích



Pro rekonstrukci stavu z logu, musí být log kompletní!

Nestačí nám tak shoda na zápisu jednoho prvku do logu

→ Musíme se shodnout i na všech předcházejících prvcích

---

Co to znamená pro followera?

Ve zprávě `AppendEntries` posíláme

- Informace o replikovaném záznamu (`term`, `index` a obsah záznamu)

a navíc...

- Informace o záznamu předcházejícím (jeho `term` a `index`)

Ve zprávě `AppendEntries` posíláme

- Informace o replikovaném záznamu (`term`, `index` a obsah záznamu)

a navíc...

- Informace o záznamu předcházejícím (jeho `term` a `index`)

---

Follower zápis odmítne, pokud se na předchozím záznamu neshodne

Ve zprávě `AppendEntries` posíláme

- Informace o replikovaném záznamu (`term`, `index` a obsah záznamu)

a navíc...

- Informace o záznamu předcházejícím (jeho `term` a `index`)

---

Follower zápis odmítne, pokud se na předchozím záznamu neshodne

Ale co pak? Log musíme nějak doplnit...

- V každém **termu** zvolíme maximálně jednoho leadera.

- V každém **termu** zvolíme maximálně jednoho leadera.
- Pokud dva logy obsahují stejný záznam (stejný **term** a **index**), pak jsou až po **index** shodné.

- V každém **termu** zvolíme maximálně jednoho leadera.
- Pokud dva logy obsahují stejný záznam (stejný **term** a **index**), pak jsou až po **index** shodné.
- Pokud někdy klientovi potvrdíme úspěšný zápis, pak nikdy nebude leaderem server, kde tento zápis ještě neproběhl.

- V každém **termu** zvolíme maximálně jednoho leadera.
- Pokud dva logy obsahují stejný záznam (stejný **term** a **index**), pak jsou až po **index** shodné.
- Pokud někdy klientovi potvrdíme úspěšný zápis, pak nikdy nebude leaderem server, kde tento zápis ještě neproběhl.

---

Kompletní algoritmus je popsán na:

<https://www.cs.princeton.edu/courses/archive/fall16/cos418/papers/raft.pdf>



- V každém **termu** zvolíme maximálně jednoho leadera.
  - Pokud dva logy obsahují stejný záznam (stejný **term** a **index**), pak jsou až po **index** shodné.
  - Pokud někdy klientovi potvrdíme úspěšný zápis, pak nikdy nebude leaderem server, kde tento zápis ještě neproběhl.
- 

Kompletní algoritmus je popsán na:

<https://www.cs.princeton.edu/courses/archive/fall16/cos418/papers/raft.pdf>

Užitečná je i vizualizace na:

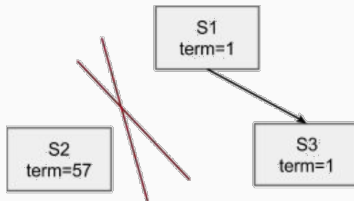
<https://raft.github.io/>

Raft je definovaný záměrně co nejjednodušeji, ale pro praxi jsou využívány různá zefektivnění:

---

# Úpravy raftu - odstranění term inflation

Raft je definovaný záměrně co nejjednodušeji, ale pro praxi jsou využívány různá zefektivnění:

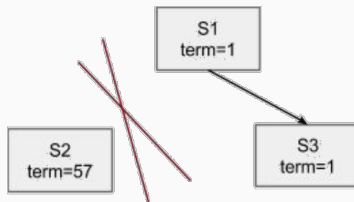


Příklad problému: "Term-inflation":

- Follower odpojený od zbytku systému po každém election timeoutu inkrementuje svůj term. Po zpětném připojení vždy spustí nové volby.

# Úpravy raftu - odstranění term inflation

Raft je definovaný záměrně co nejjednodušeji, ale pro praxi jsou využívány různá zefektivnění:



Příklad problému: "Term-inflation":

- Follower odpojený od zbytku systému po každém election timeoutu inkrementuje svůj term. Po zpětném připojení vždy spustí nové volby.
- Řešení: "Pre-vote", server se stane kandidátem jen pokud má šanci na zvolení<sup>4</sup>.

---

<sup>4</sup>Paper: Four modifications for the Raft consensus algorithm

## Zadání semestrální práce

---

Naimplementujte algoritmus Raft pro replikaci key-value storu

Zpracování musí být **distribuované**, procesy si nesahají vzájemně do paměti!

Naimplementujte algoritmus Raft pro replikaci key-value storu

Zpracování musí být **distribuované**, procesy si nesahají vzájemně do paměti!

Termín odevzdání je **10. 6. 2022 23:59 CET**

(příp. do termínu Vaší zkoušky, pokud byste na ni šli před odevzdáním)

Podrobnosti upřesníme.

Díky za pozornost!

Budeme rádi za Vaši  
zpětnou vazbu! →



[https://forms.gle/  
vwbWazEu14w1Kf487](https://forms.gle/vwbWazEu14w1Kf487)