Distribuované algoritmy - přehled

Přednášky z Distribuovaných systémů

Ing. Jiří Ledvina, CSc.

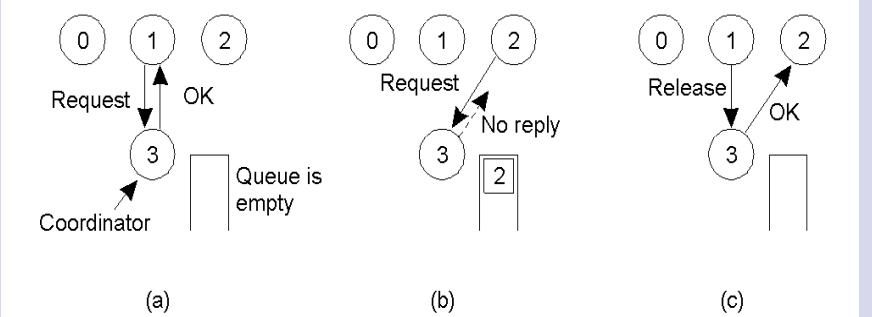
Úvod

- Distribuované algoritmy
  - Princip soupeření
  - Princip předávání pověření
- Typy úloh
  - Vzájemné vyloučení
  - Výběr 1 z N
  - Algoritmy shody
  - Algoritmy detekce ukončení
  - Algoritmy detekce a prevence uvíznutí

Kritické sekce

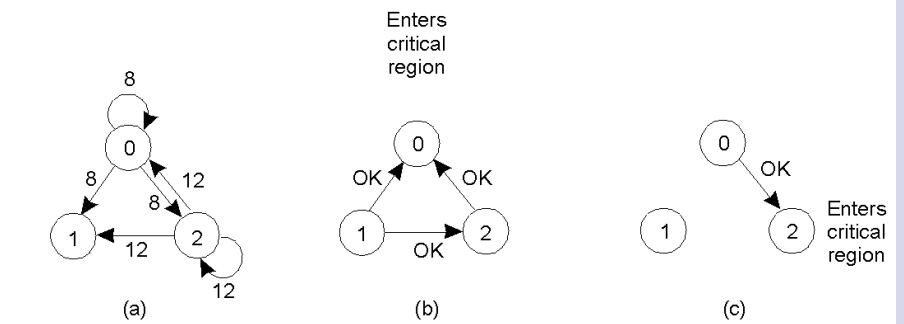
- Sdílení zdrojů
  - Nekoordinovaný přístup
  - Potřeba řídit přístup ke zdrojům
  - Centralizované a decentralizované řešení
  - Těsně vázané a volně vázané systémy
  - Semafory, kritické sekce, monitory
- Podpora hardware nebo software
  - Test and set, compare and swap
  - Semafor
  - Sequencer, čítač událostí

Příklad - Vzájemné vyloučení: centralizovaný algoritmus



a) Proces 1 žádá koordinátora o povolení vstoupit do kritické sekce. Dostává povolení.
b) Poté žádá o povolení vstoupit do téže kritické sekce proces 2. Koordinátor neodpovídá.
c) Když proces 1 opouští kritickou sekci, oznámí to koordinátorovi a ten opoví procesu 2.

Příklad - Distribuovaný algoritmus vzájemného vyloučení



a) Dva procesy chtějí vstoupit do kritické oblasti v tentýž moment.
b) Proces 0 má nižší časovou známku a tak vítězí.
c) Když proces 0 opouští kritickou sekci, posílá OK a proces 2 může vstoupit do kritické sekce.

Distribuované vzájemné vyloučení

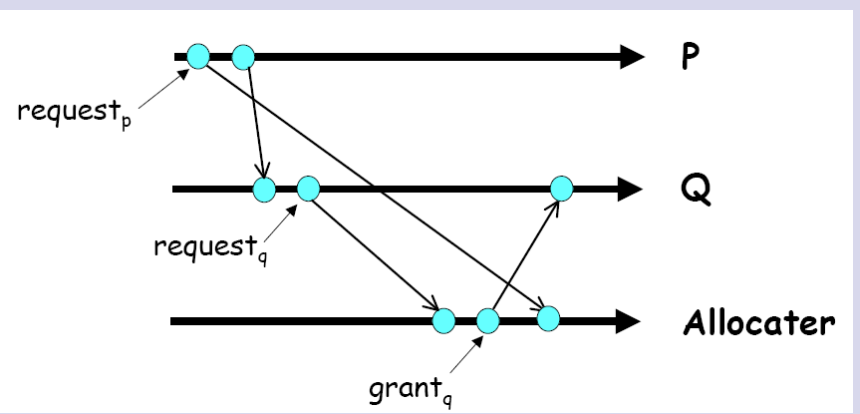
- Základní rozdělení
  - Centralizované metody (sequencer)
  - Decentralizované metody
    - Založené na soupeření
    - Založené na předávání pověření

Distribuované vzájemné vyloučení

- Existuje pevný počet procesů, které sdílí jeden zdroj.
- Zdroj může používat v jednu chvíli pouze jeden proces.
- Podmínky
  - Proces, kterému bylo povoleno užívat zdroj jej musí uvolnit dříve, než jej začne používat jiný proces.
  - Požadavky procesů musí být zpracovány v pořadí, ve kterém vznikly.
  - Jestliže platí, že zdroj je procesy v konečném čase uvolněn, pak musí být také každý požadavek v konečném čase zpracován.

Distribuované vzájemné vyloučení

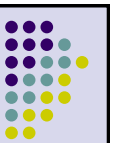
- Centralizované řešení



Distribuované vzájemné vyloučení

- Decentralizované metody založené na soupeření
  - Lamportův algoritmus – libovolná topologie, časové značky (3 fáze)
  - Ricart a Agrawala – libovolná topologie, časové značky (2 fáze)
  - Maekawa – vytváření hlasovacího quora
- Decentralizované metody založené na předávání pověření
  - Suzuki a Kasami – broadcast
  - LeLann – logický kruh
  - Reymond – stromová struktura, rozšíření pro sdílení K identických zdrojů

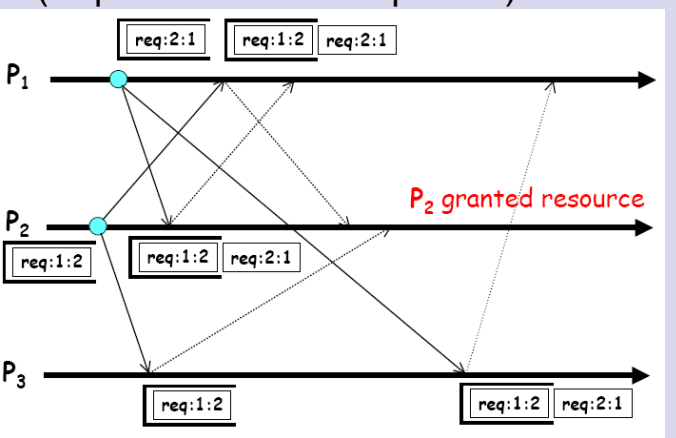
Lamportův algoritmus



- Decentralizovaný algoritmus vzájemného vyloučení
- Předpokládá obousměrné FIFO kanály mezi procesy
- Každý proces udržuje vlastní frontu požadavků
- Používá časových značek k uspořádání požadavků
- Vyžaduje 3(n-1) zpráv
- Probíhá ve třech fázích
  - Požadavek (req)
  - Potvrzení (ack)
  - Uvolnění (rel)

Lamportův algoritmus

Příklad (req: časová značka :proces)

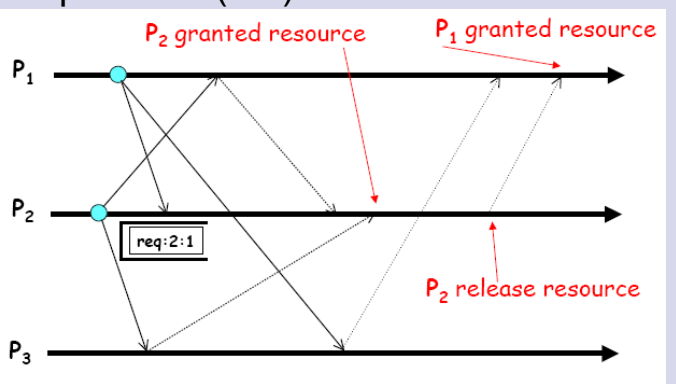


Lamportův algoritmus

- Algoritmus
  - Proces P požaduje zdroj posláním požadavku do všech procesů zasláním zprávy **req**
  - Při příjmu požadavku je požadavek zařazen do fronty požadavků uspořádané dle časových značek, poslání zprávy **ack**
  - Ukončení zpracování požadavku – odstranění požadavku z fronty a poslání zprávy **rel** ostatním procesům
  - Přijetí zprávy **rel** od procesu P – odstranění požadavku procesu P z fronty
  - Povolení zpracování požadavku – požadavek je na prvním místě ve frontě a je potvrzen ostatními procesy

Algoritmus vzájemného vyloučení Ricard - Agrawala

- Optimalizace Lamportova algoritmu snížením počtu zpráv na 2(n-1)



Algoritmus vzájemného vyloučení Ricard - Agrawala

- Algoritmus
  - Proces P požaduje zdroj posláním požadavku do všech procesů zasláním zprávy **req**
  - Při příjmu požadavku je požadavek potvrzen zprávou **ack** pouze tehdy, jestliže přijímající proces zdroj neužívá nebo o něj právě nežádá. Jinak je požadavek zařazen do fronty.
  - Ukončení zpracování požadavku – poslání zprávy **ack** procesům, kterým bylo poslání předtím odepřeno (a odstranění požadavků z fronty)
  - Povolení zpracování požadavku – požadavek je potvrzen ostatními procesy

Algoritmus Maekawa

- Vzájemné vyloučení zajištěno vytvořením hlasovacího quora
- Vyžaduje souhlas $(2\sqrt{N})-1$ procesů

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Algoritmy založené na předávání pověření

- Suzuki a Kasami – broadcast
- LeLann – logický kruh
- Raymond – stromová struktura, rozšíření pro sdílení K identických zdrojů

Algoritmus vzájemného vyloučení Suzuki-Kasami

- Metoda využívá distribuci pověření mezi jednotlivými procesy
- Každý uzel udržuje celočíselné pole největších sekvenčních čísel obdržených od jednotlivých procesů
- Proces posílá ostatním pověření, které obsahuje
  - frontu požadavků a
  - vektor LN posledních požadavků jednotlivých procesů

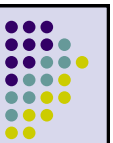
Algoritmus vzájemného vyloučení Suzuki-Kasami

- Pokud chce proces vstoupit do kritické sekce a nemá pověření
  - Zvýší vlastní TS ve vektoru TS a pošle ostatním požadavek (i, TS_i)
- Pokud proces přijme požadavek od i-tého procesu
  - Nastaví položku vektoru TS[i] na maximum
  - Jestliže má pověření a TS[i]=LN[i]+1, pošle pověření do P_i

Algoritmus vzájemného vyloučení Suzuki-Kasami

- Pokud P_i opouští kritickou sekci
  - Nastaví LN[i] v pověření na vlastní vektor časových značek
  - Frontu v pověření nastaví tak, že do ní zahrne všechny procesy, které v ní nejsou a mají časovou značku požadavku o 1 vyšší než je časová značka v pověření
  - Pověření se pošle prvnímu ve frontě (ten se z fronty odstraní)
  - Fronta je posílána v pověření

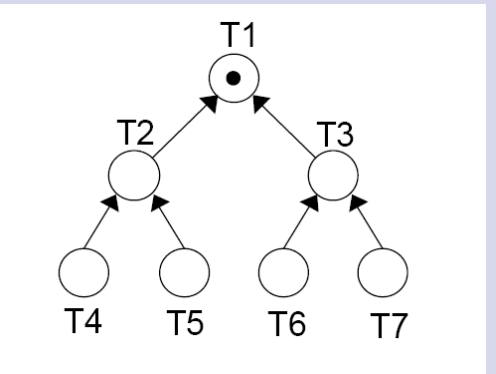
LeLann algoritmus



- Procesy jsou uspořádány do logického kruhu
- Na začátku má pověření proces 0
  - Pověření je posíláno v kruhu (přenos zpráv dvoubodovými spoji)
  - Po obdržení pověření může proces vstoupit do kritické sekce
  - Po opuštění CS předává pověření dál
- Problém s detekcí ztráty pověření
- Jednoduché rozšíření kruhu

Raymodův algoritmus

- Procesy jsou uspořádány do stromu



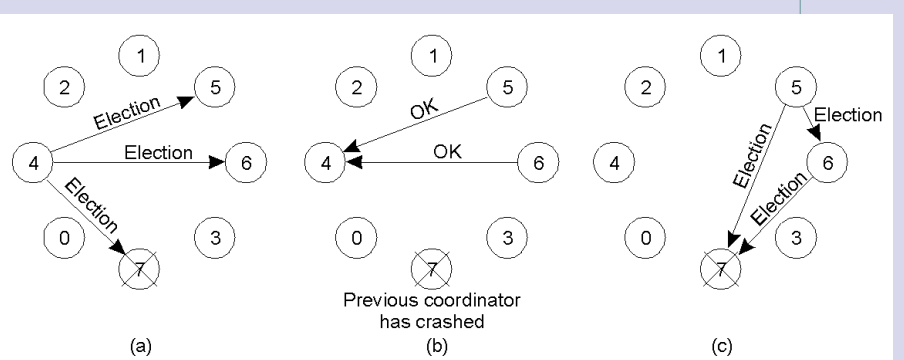
Raymodův algoritmus

- Pracuje na bázi předávání pověření
- Proces udržuje frontu požadavků na pověření od procesů nižších úrovní
- Požaduje-li vstup do CS a fronta je prázdná, posílá požadavek nadřazenému a řadí se do fronty
- Požaduje-li podřízený proces vstup do CS, předá mu pověření nebo jej zařadí do fronty
- Obdrží-li pověření, předá ho prvnímu ve frontě

Algoritmy výběru 1 z N

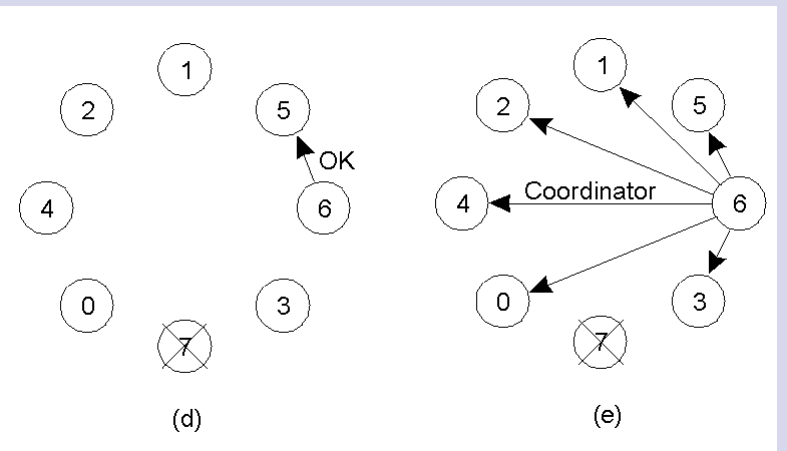
- úplný graf – Bully algoritmus
- kruhová topologie
  - logický kruh – předávání dle seznamu, rekonstrukce kruhu
  - fyzický kruh – předávání dle sousedství uzlů
- stromová topologie – logický strom

Algoritmus vhazování (The Bully Algorithm) (1)

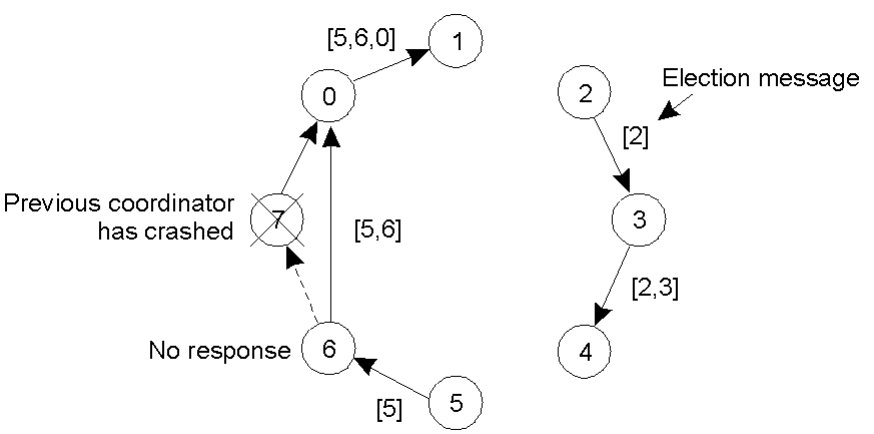


- Algoritmus výběru vhazováním
- Proces 4 má výběr
- Procesy 5 a 6 odpovídají, že se má proces 4 zastavit
- Nyní drží výběr 5 i 6.

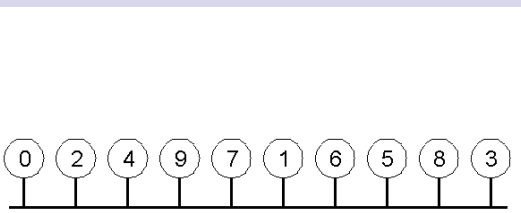
Algoritmus vhazování (The Bully Algorithm) (3)



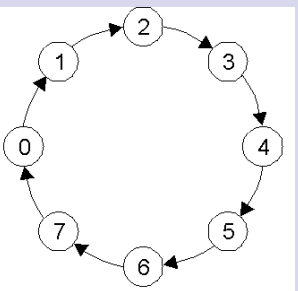
Algoritmus výběru v kruhu



Algoritmus předávání pověření v kruhu (Token Ring)



(a)



(b)

a) Neuspořádaná skupina procesů v síti.
b) Logický kruh vytvořený programově.

Algoritmy detekce ukončení

- asynchronní systémy
- synchronní systémy
  - Dijkstra-Scholten – difuzní výpočty
    - jeden iniciátor
    - uspořádání do stromu, iniciátor je kořen
    - detekce ukončení podle počtu ukončených podřízených
  - Shavit-Francez
    - více iniciátorů
    - všechny procesy se účastní vlny
    - proces který není iniciátorem pokračuje ve vlně
    - pokud strom kolabuje, iniciátory pokračují ve vlně

Algoritmy shody

- Shoda na hodnotě
- Interaktivní konzistentnost – shoda na vektoru hodnot
- Problém Byzantinských generálů

Shoda

- N procesů se chtějí dohodnout na hodnotě
  - Např. Na synchronizované akci (go/abort)
- Koncensus může být dosažen i za přítomnosti poruch
  - Porucha procesu – krach,svévolná porucha
  - Komunikační porucha – ztracené nebo zničené zprávy

Synchronní a asynchronní systémy

- Synchronní systémy
  - omezené zpoždění přenosu zpráv (latency)
  - procesy mají synchronizované hodiny
  - časy zpracování jsou omezené
  - zhroucení (crash failure) procesu může být přesně detekováno
- Asynchronní systémy - bez omezujících předpokladů
  - zhroucení (crash failure) procesu nemůže být přesně detekováno
- Failstop
  - podobné jako asynchronní, ale zhroucení je přesně detekováno

Algoritmus shody

- Algoritmus shody
  - Všechny procesy P_i začínají ve stavu „nerozhodnutý“
  - Každý P_i navrne hodnotu V_i z množiny D a pošle ji ostatním procesům
  - Shody je dosaženo pokud se všechny nechybující procesy shodnou na téže hodnotě d
  - Každý nechybující proces P_i nastaví svou rozhodovací proměnnou na d a změní svůj stav na „rozhodnutý“

Algoritmus shody - princip

- Algoritmus probíhá ve více kolech (round)
- Jedno kolo zahrnuje
  - poslání zprávy do skupiny procesů
  - příjem zpráv z předchozího kola
  - lokální zpracování (rozhodnutí, zastavení)
- Jednoduché, neefektivní v pomalých sítích

Požadavky na shodu

- Ukončení – každý proces nastaví vybranou (rozhodnutou) hodnotu
  - Toto není možné při krachu procesoru v asynchronních systémech
- Souhlas (dohoda) – výběr hodnoty je shodný pro všechny korektní procesy
  - Svévolné (Byzantinské) poruchy mohou způsobit nekonzistentnost a brání dohodě
- Integrita (celistvost) – pokud všechny korektní procesy P_i navrhnou tutéž hodnotu d, pak dohodnutá hodnota je d
- Shoda může zahrnovat fázi návrhu a fázi dohody

Interaktivní konzistentnost

- Interaktivní konzistentnost je speciální případ shody, kde se procesy dohodnou na vektoru hodnot, pro každý proces na jedné hodnotě
- Ukončení – každý korektní proces nastaví svůj rozhodovací vektor
- Souhlas – rozhodovací vektor je stejný pro všechny korektní procesy
- Integrita – jestliže P_i je korektní, pak se všechny korektní procesy dohodnou na V_i jako na *i-tém* prvku rozhodovacího vektoru
- Hodnota Vi pro chybující proces může být ignorována nebo dohodnuta shodou (konsensus)

Shoda v synchronních systémech

- Pro systém se *2k+1* procesy, kde maximálně k procesů může být chybných potřebuje algoritmus *k+1* kroků (s timeoutem) a s využitím základního multicastu.
- Každý proces P_i má seznam $V\text{-}i_1$, $V\text{-}i_2$, … přenesených hodnot v krocích *1, 2, …*

$$
\begin{aligned}
&\textit{Initially } V\text{-}i_0 = \{\}\ ;\ V\text{-}i_1 = \{v_i\} \\
&\quad \textit{for round} = 1 \textit{ to } k+1 \textit{ do } \{ \\
&\qquad \textit{multicast } (V\text{-}i_{round} - V\text{-}i_{round-1}) \\
&\qquad V\text{-}i_{round+1} = V\text{-}i_{round} \\
&\qquad \textit{for each } V\text{-}j_{round} \textit{ received} \\
&\qquad\quad V\text{-}i_{round+1} = V\text{-}i_{round+1} \cup V\text{-}j_{round} \\
&\qquad \} \\
&\quad \} \\
&\quad d\text{-}i = \textit{minimum } V\text{-}i_{k+2}
\end{aligned}
$$

Byzantinští generálové

- Tři nebo více generálů se potřebuje vzájemně seznámit se svými záměry
- Jeden nebo více generálů může být zrádce, který dává chybné informace
- K řešení tohoto problému používají protokol
  - Každý z generálů posílá svou informaci ostatním (předpokládáme spolehlivou komunikaci)
  - Jakmile generál shromáždí všechny hodnoty, pošle vektor hodnot ostatním generálům
  - S využitím hlasovacího mechanizmu může každý generál získat správné hodnoty

Byzantinští generálové

- Problém není řešitelný pro $N \leq 3$
- Pro k zrádců je třeba 2k+1 loajálních generálů
  - Problém nemůže být řešen pokud chybný proces lže konzistentně
  - Lze použít zabezpečení zprávy (šifrování, digitální podpis)

•

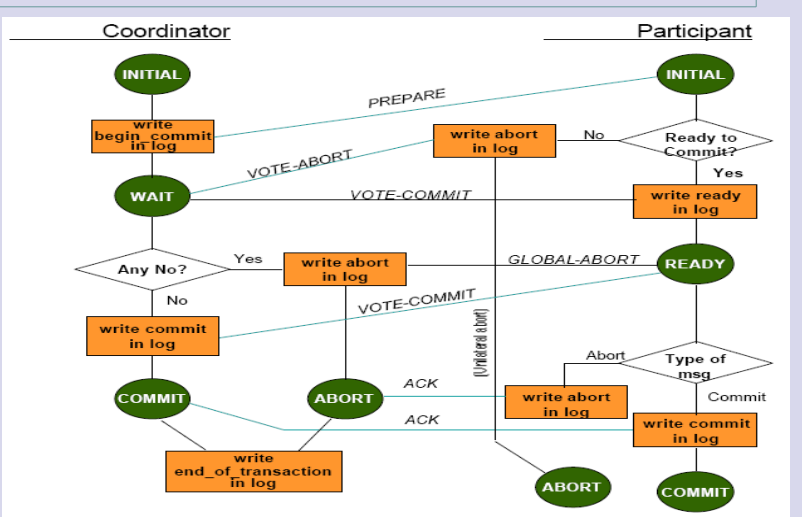
2-fázový commit (2PC)

- **Koordinátor** : Koordinátor je proces, který vyvolá transakci a řídí její průběh
- **Účastník (participant)** : proces, který se účastní provedení transakce v jiném uzlu

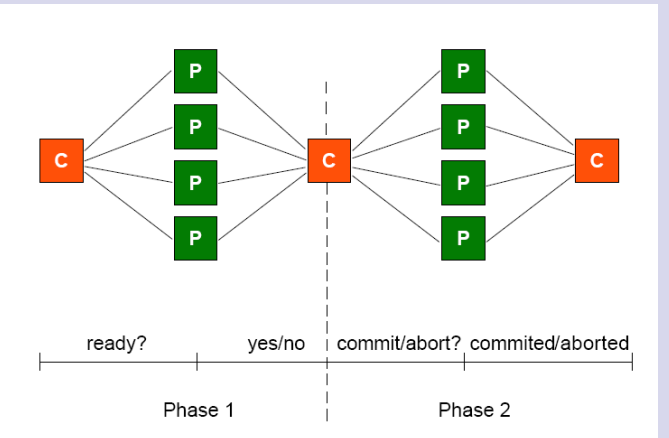
1. *Phase 1*: Koordinátor pošle ostatním účastníkům požadavek – zprávu účastníci volí mezi provedením (commit) a zrušením (abort)
2. *Phase 2*: Všichni provedou akci

3. Pravidla provedení: Koordinátor provede transakci pokud všichni účastníci volí její provedení

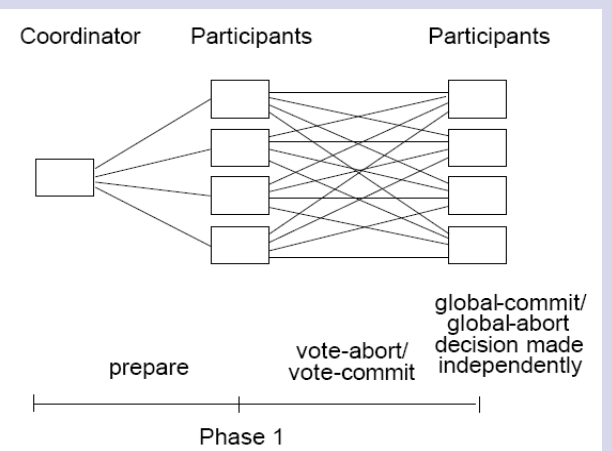
Akce 2PC protokolu



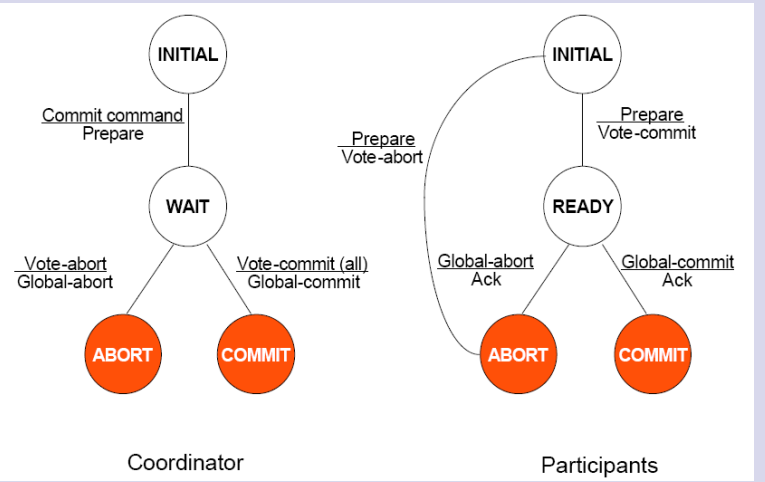
Centralizovaný 2PC



Distribuovaný 2PC



Stavové přechody 2PC



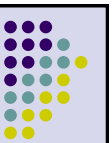
Problémy s 2PC

- blokování
  - Ready znamená, že účastník (participant) čeká na koordinátora
  - Pokud koordinátor skončí chybou, čeká participant na obnovu
  - Blokování redukuje dostupnost
- Není možná nezávislá obnova
- Hledáme protokol pro obnovu, který by neblokoval při výpadku účastníka nebo koordinátora
- Takovým protokolem je – 3PC

3-fázový commit (3PC)

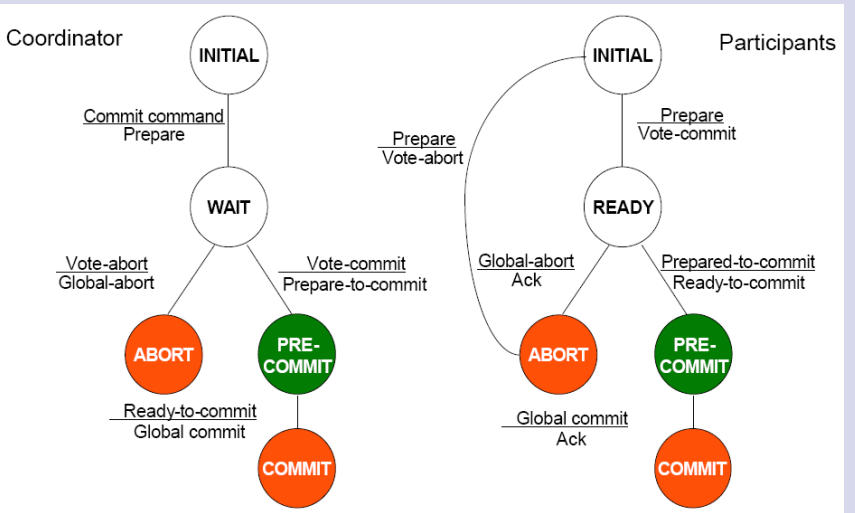
- 3PC je neblokující.
- Protokol ukončení (commit) je neblokující jestliže
  - Je synchronní během jednoho přechodu
  - Jeho diagram přechodů zahrnuje
    - Žádný stav, který by sousedil s oběma commit i abort stavy
    - Žádné nespáchatelné (non commit) stavy, sousedící s commit stavem
- sousedství: možnost přejít z jednoho stavu do druhého jedním přechodem
- spáchatelný: všechny strany volily commit transakci

3-fázový commit (3PC)



- Fáze hlasování, fáze rozhodování, fáze provedení (commit).
- Fáze hlasování
  - Koordinátor posílá požadavek hlasování
  - Všichni odpoví commit nebo abort
- Fáze rozhodování
  - Po příjmu všech odpovědí rozhodne koordinátor o výsledku – commit nebo abort
  - Koordinátor posílá abort nebo prepare-to-commit
  - Všichni odpovídají ACK
- Fáze provedení (commit)
  - Po získání všech ACK posílá koordinátor commit
  - Všichni potvrzují ACK

Stavy ve 3PC



3PC - komunikace