DISTRIBUTED TRANSACTIONS

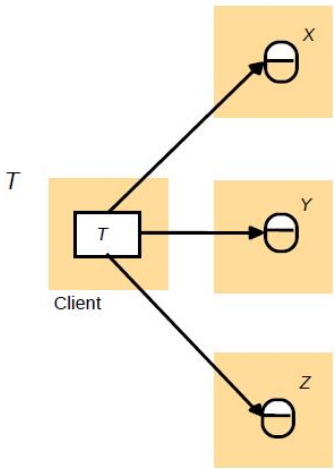
Distributed Transactions

- Proses transaksi (flat / nested) yang mengakses object yang dikelola oleh beberapa server
- Diperlukan sebuah coordinator untuk memastikan konsep atomicity
- Menggunakan protokol yang sudah disepakati sejak awal (two phase commit protocol)
- Memerlukan concurrency control

Flat Transaction (1)

- Sebuah client melakukan request pada lebih dari satu server
- Sebuah transaksi diselesaikan dahulu sebelum meminta ke server berikutnya (sequential)
- Jika menggunakan konsep locking, maka sebuah transaksi hanya bisa menunggu satu obyek saja

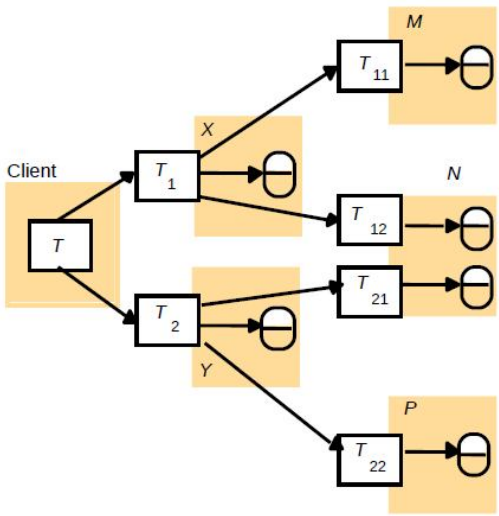
Flat Transaction (2)



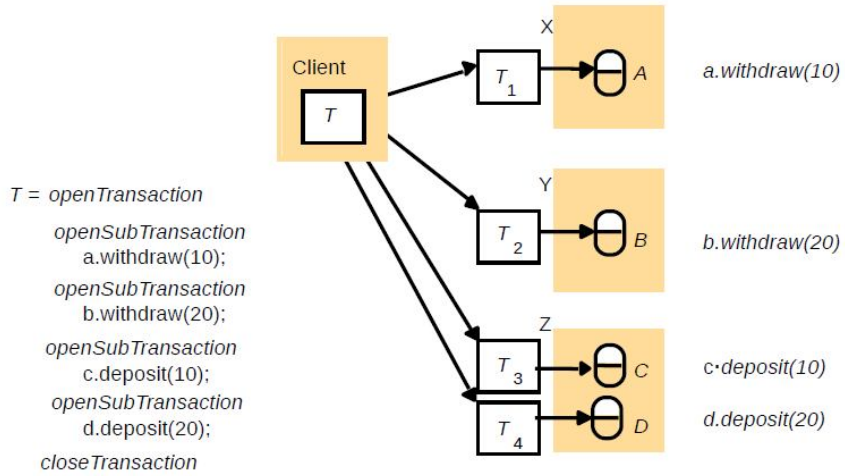
Nested Transactions (1)

- Top level transaction bisa membuat sub transaksi
- Sub transaksi pada level yang sama bisa berjalan secara concurrent (parallel)
- Mampu memanfaatkan utilitas prosessor (pada kasus multi processor)

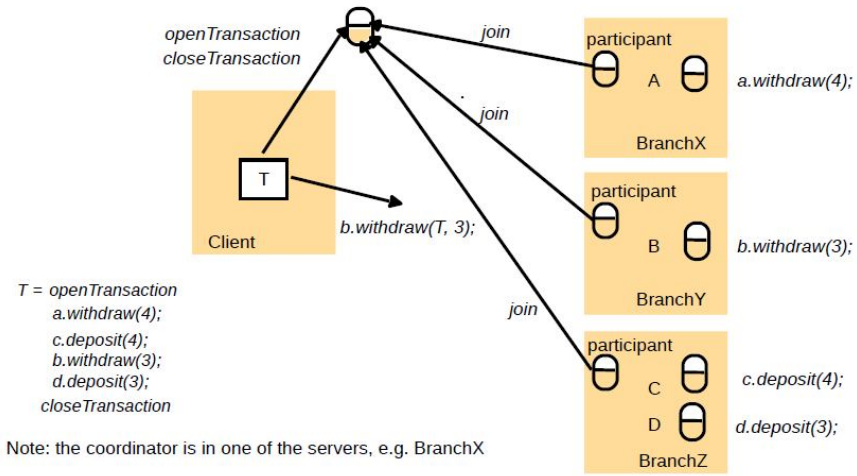
Nested Transactions (2)



CONTOH TRANSAKSI BANK (1)



CONTOH TRANSAKSI BANK (2)



Note: the coordinator is in one of the servers, e.g. BranchX

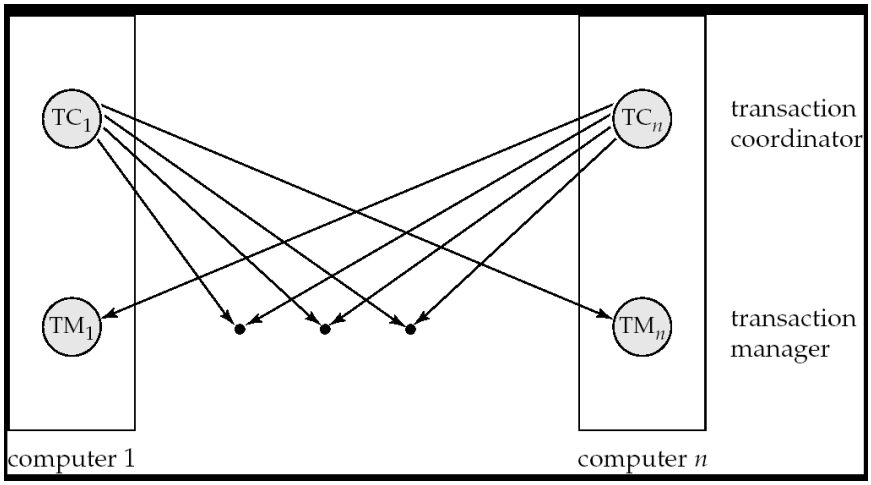
PROSES TRANSAKSI

- Client mengirimkan request *openTransaction* pada coordinator, pada sembarang server
- Coordinator yang dituju mengembalikan identifier transaction yang unik pada client
- Client bekerjasama dengan coordinator dengan commit protocol

Coordinator (1)

- Sebuah host/system yang menjadi “mandor” untuk menyelesaikan sebuah transaksi terdistribusi
- Bertanggung jawab untuk mengambil keputusan commit/abort
- Selama terjadi transaksi, coordinator mencatat daftar referensi dari partisipan
- Coordinator tahu semua daftar peserta

Coordinator (2)



Atomic Commit Protocols

- Dikembangkan awal 1970an
- Bertujuan untuk memastikan bahwa setiap perubahan terjadi pada semua system atau tidak sama sekali (all or nothing)

  Terdapat tiga jenis
  - One phase commit protocol
  - Two phase commit protocol
  - Three phase commit protocol
- Berakhir setelah transaksi di commit/abort

One Phase Commit Protocol

- Coordinator mengirimkan pesan commit atau abort pada semua partisipan
- Proses diulang terus menerus sampe semua sudah membalas
- Masalah : Tidak mungkin melakukan abort setelah ada permintaan untuk commit
- Solusi : Two Phase Commit

Two Phase Commit Protocol (1)

- Fase 1 (voting)
  - Coordinator mengirimkan request *canCommit pada* setiap partisipan
  - Partisipan memilih yes/no dan mengirim balik pada coordinator. Jika yes, maka menyimpan obyek pada penyimpanan permanen. Jika tidak, abort

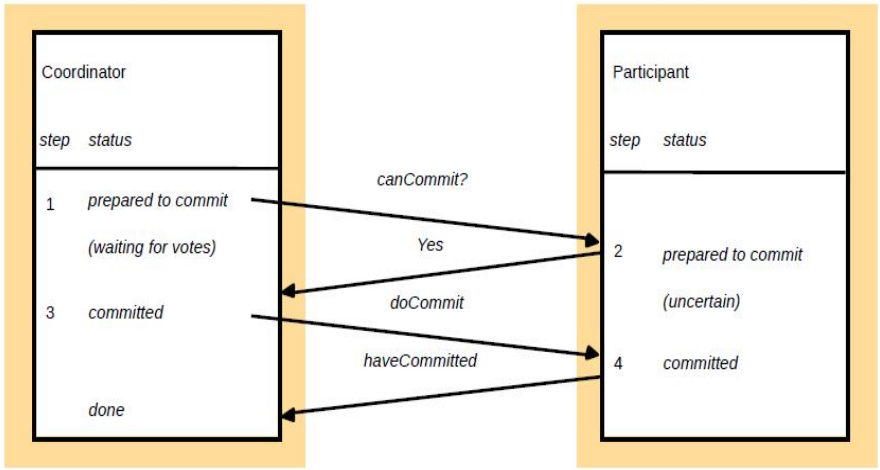
Two Phase Commit Protocol (2)

- Fase 2
  – Coordinator mengumpulkan hasil voting
  – Jika semua setuju, coordinator memutuskan commit dan mengirimkan *doCommit pada semua* partisipan
  – Selain itu, coordinator memutuskan abort dan mengirimkan *do Abort pada semua partisipan yang* memilih yes
  – Partisipan yang memilih yes menunggu doCommit/ *doAbort dari coordinator*

Two Phase Commit Protocol (3)

- Setelah menerima salah satu dari pesan diatas, partisipan menjalankan perintah sesuai dengan pesan yang diterima
- Jika dilakukan perintah commit, maka partisipan mengirimkan pesan *haveCommitted* pada coordinator sebagai konfirmasi bahwa proses sudah dilaksanakan

Two Phase Commit Protocol (4)



Masalah pada Two Phase Commit

- Susah memastikan semua partisipan sudah melakukan vote dan mendapatkan hasil yang sama
- Jika proses mengalami kegagalan (terjadi network partitioning), maka tidak akan didapatkan konsensus, karena partisipan yang lain akan saling menunggu (blocking)
- Solusi : three phase commit

Three Phase Commit

- Mencoba mengatasi masalah blocking
- Menggunakan asumsi tidak lebih dari *k site fail* (k adalah angka yang sudah disetujui)
- Coordinator memastikan bahwa paling tidak ke site lain tahu bahwa coordinator akan melakukan commit
- Jika coordinator fail, site yang lain melakukan election coordinator baru dan melihat status terakhir dan menentukan aksi (commit/abort)

Masalah pada Three Phase Commit

- Implementasinya susah
- Harus memastikan bahwa state harus tetap konsisten meskipun terdapat perbedaan hasil (transaksi dicommit di satu site dan abord di site yang lain sebagai akibat dari network partitioning)
- Terlalu banyak overhead

Concurrency Control

- Setiap server mengelola sekumpulan obyek dan bertanggung jawab untuk memastikan tetap konsisten ketika diakses oleh transaksi concurrent
- Jenis
  - Locking
  - Timestamp ordering concurrency control
  - Optimistic concurrency control

Distributed Locking (1)

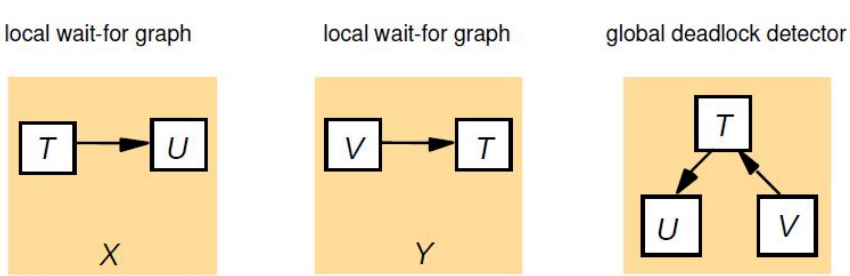
- Lock pada obyek dikendalikan secara lokal oleh local lock manager
  - Memberikan akses untuk lock
  - Membuat transaksi yang meminta request menunggu
- Tidak bisa melepas lock sampai ada kepastian abort/commit pada semua server
- Bisa menimbulkan distributed deadlock

Distributed Locking (2)

U		V		W	
d.deposit(10)	lock D				
		b.deposit(10)	lock B at Y		
a.deposit(20)	lock A at X				
				c.deposit(30)	lock C at Z
b.withdraw(30)	wait at Y				
		c.withdraw(20)	wait at Z		
				a.withdraw(20)	wait at X

Phantom Deadlock (1)

- Deadlock yang terdeteksi tetapi bukan merupakan sebuah deadlock
- Terjadi karena adanya delay pada saat pengiriman informasi deadlock pada jaringan



Phantom Deadlock (2)

- Transaksi U melepas obyek pada server X
- Transaksi U meminta V pada server Y
- Asumsi
- – Server Y diterima terlebih dahulu dibandingkan server X
- T--> U --> V --> T
- T --> U sudah tidak ada lagi

Transaction Recovery

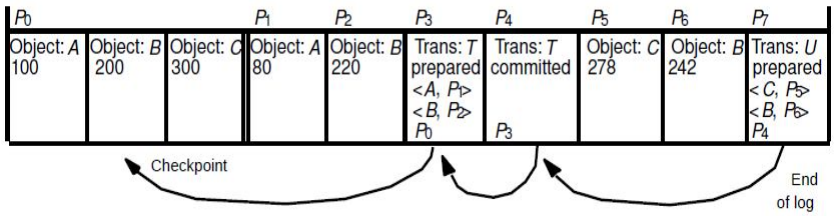
- Proses pemulihan dengan menggunakan versi terakhir dari obyek yang sudah dicommit dari media penyimpanan permanen
- Dilakukan oleh recovery manager
  - – Menyimpan obyek pada storage permanen
  - – Restore obyek server setelah crash
  - – Mengorganisasi ulang file recovery untuk meningkatkan performa dari recovery
  - – Reclaim storage space

Intention List

- Berisi daftar obyek aktif yang diakses oleh client selama transaksi (termasuk yang mengalami perubahan selama transaksi)
- Digunakan untuk melihat obyek mana yang terkena efek perubahan setelah commit
- Jika commit, maka nilai baru disimpan pada storage. Jika abort, data dari intention list digunakan untuk membatalkan perubahan dan melakukan rollback ke data lama

Logging

- Semua transaksi yang commit dicatat pada log
- Urutan entry menggambarkan urutan transaksi
- Setelah crash, semua transaksi yang tidak memiliki status *committed akan dibatalkan*



SEKIAN