Sistemas Distribuidos

Sincronización, Concurrencia y Transacciones

Transacciones Distribuidas

Transacciones Distribuidas

Transacciones que afectan de forma atómica a objetos residentes en varios servidores.

Uso principal: transacciones distribuidas

- Cuando termina la transacción (*end-transaction*):
  - Si todos los procesadores implicados están de acuerdo, se "compromete"
  - Si algún procesador quiere abortarla o está caído, se "aborta"

Protocolo clásico *two-phase-commit* (2PC)

- Proceso que ejecuta transacción actúa de coordinador
- Requiere *almacenamiento estable*: ("nunca" pierde la infor.)
  - Uso de dos discos: se escribe primero en uno y luego en otro

Two-Phase Commit

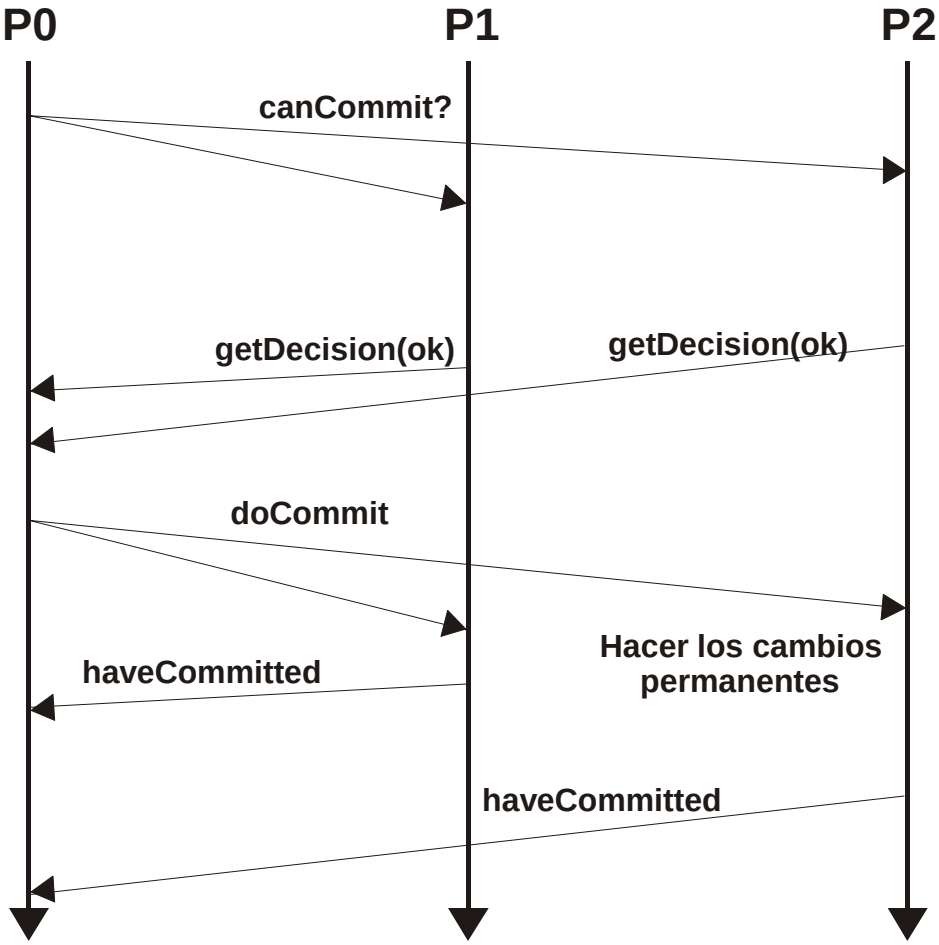
Mensajes intercambiados en *two-phase commit*:

- **`canCommit?()`**: El coordinador consulta a los servidores.
- **`doCommit()`**: El coordinador solicita a los servidores el procesamiento de las modificaciones.
- **`doAbort()`**: El coordinador indica a los servidores que la operación se aborta.
- **`haveCommitted()`**: El servidor indica que ha completado la operación.
- **`getDecision()`**: El servidor indica si puede realizar la acción.

Two-Phase Commit

Coordinador (Monitor Transaccional):

- Escribir **`canCommit?()`** en mem. estable.
- Mandar a subordinados. **`canCommit?()`** Recoger las respuestas **`getDecision()`**
- Si todos *ok* => **`doCommit()`**
- Si alguno *abort* o no responde=>**`doAbort()`**
- Escribir resolución en mem. estable
- Mandar resolución



Víctor Robles
Francisco Rosales
Fernando Pérez
José María Peña

Two-Phase Commit

Subordinados (Servidores/objetos transaccionales):

Recibir **`canCommit?()`**

Decidir respuesta y grabar en mem.estable

Mandar respuesta: **`getDecision()`**

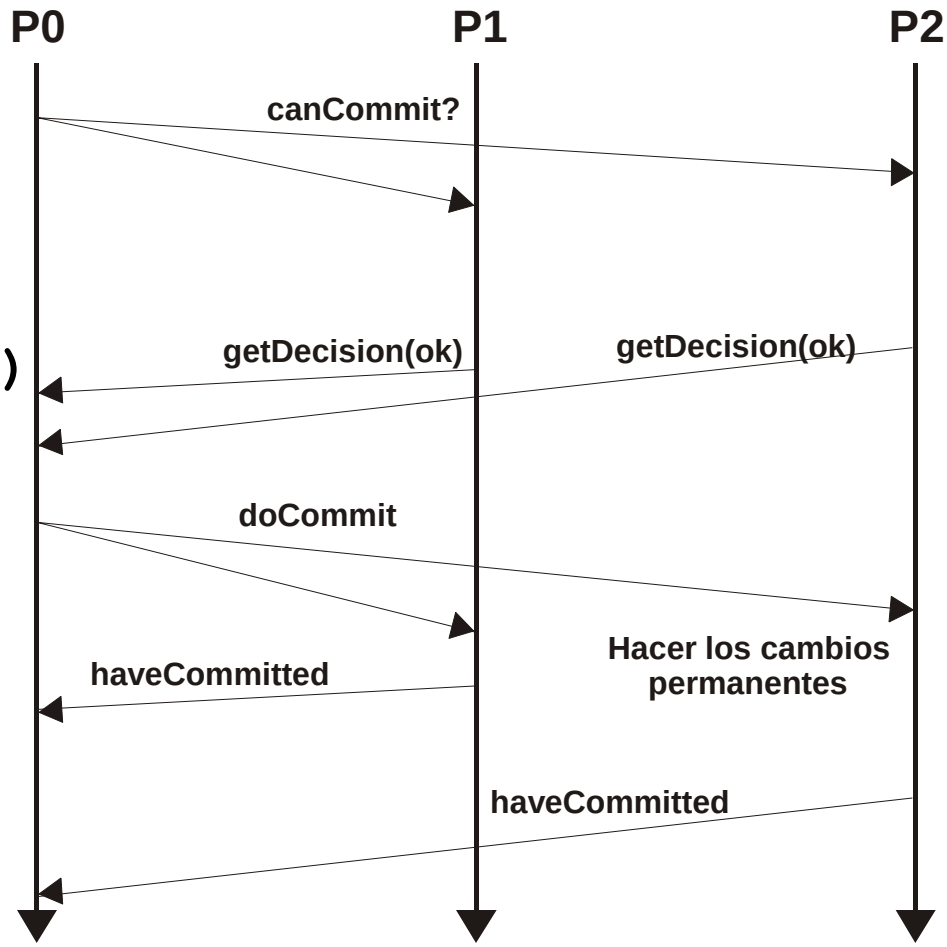
Recibir resolución

Escribir resolución en mem. estable

Llevar a cabo resolución:

`doCommit()` => hacer cambios permanentes

`doAbort()` => deshacer cambios



Fallos en 2PC

- Buena tolerancia a fallos
  - Recuperación después de caída: consulta mem. estable
- Recuperación después de caída de un subordinado:
  - Si encuentra en mem. estable la respuesta pero no la resolución:
    - pregunta a coordinador cuál ha sido la resolución
  - Si encuentra en mem. estable la resolución:
    - la lleva a cabo
- Recuperación después de caída de coordinador:
  - Si encuentra en mem. estable **`canCommit?()`** pero no resolución:
    - manda a los subordinados mensajes **`canCommit?()`**
  - Si encuentra en mem. estable la resolución:
    - manda a los subordinados mensajes con la resolución

Three-Phase Commit

Existe una variante del 2PC denominada *Three-Phase Commit*

Fases:

- El coordinador transmite **`canCommit?()`** a todos los servidores.
- Los servidores responden con **`getDecision()`** al coordinador.
- El coordinador recolecta las respuestas y manda:
  - **`preCommit()`** : Si todos aceptan.
  - **`doAbort()`** : Si todos aceptan.
- Los servidores con un asentimiento.
- Cuando todos los asentimientos han sido recibidos entonces transmite **`doCommit()`**

- Es no bloqueante y más robusta ante fallos que el 2PC

Víctor Robles
Francisco Rosales
Fernando Pérez
José María Peña

Monitor Transaccional

Se denomina Monitor Transaccional al elemento que coordina la articulación de transacciones. En términos generales:

- Genera el ID de transacción cuando se arranca, y
- Es el encargado de, al final de la transacción, verificar si la transacción es correcta o no (articula el 2PC o equivalente).

Las tecnologías para dar soporte a sistemas de control transaccional se dividen en:

- Estándares de interfaces y servicios.
- Implementaciones concretas (productos software y *frameworks*).

Estándares y Servicios

- Alternativas:
  - X/Open DTP (Distributed Transaction Processing):
    - Parte de la especificación X/Open XA (Extended Architecture).
    - Soporta propiedades ACID base y protocolo 2PC.
  - COSS Transactions de CORBA:
    - Especificaciones de servicios de transaccion de objectos basados en la arquitectura CORBA.
    - Compatible X/Open XA.
  - JTS (Java Transaction Service) / JTA (Java Transaction API):
    - JTS es una especificación y JTA son interfaces de programación.
    - Soporte Java para interacción con X/Open XA (JTA) o con el servicio de transacciones CORBA/OMG (JTS).
    - Se necesitan implementaciones concretas de ambos: JBoss TS o Bitronix JTA).

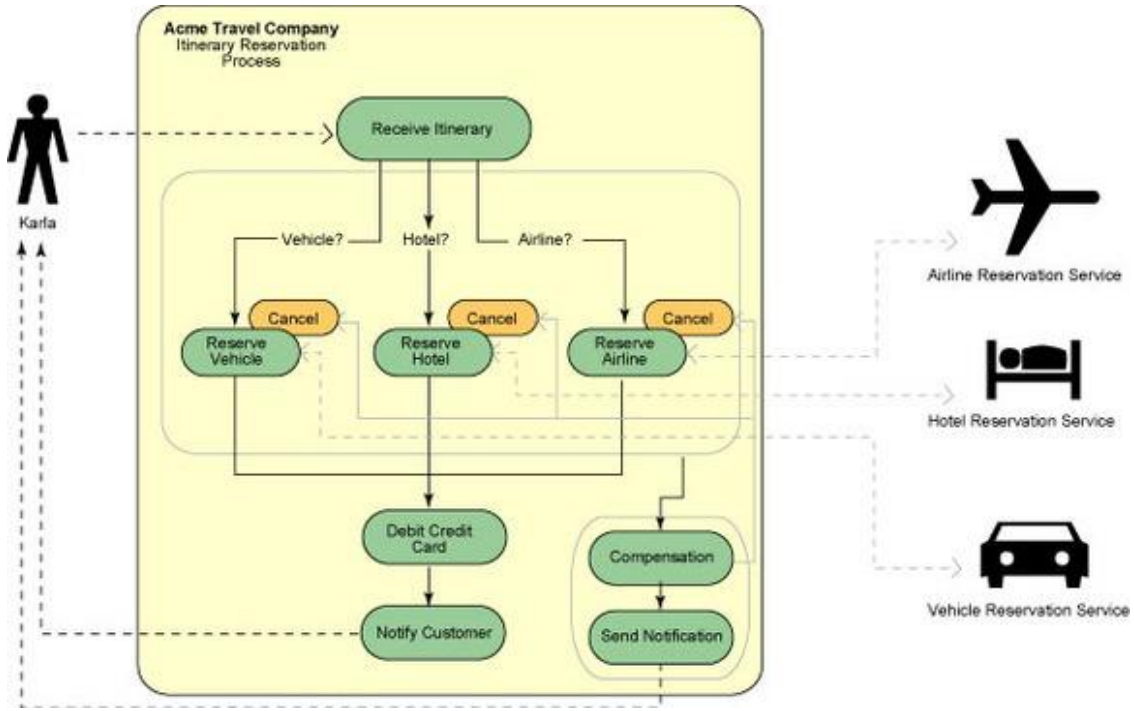
Productos y Frameworks

- Alternativas:
  - MTS (Microsoft Transaction Server):
    - Soportado nativamente en versiones server de MS Windows.
    - Se basa en un Ejecutivo MTS y una serie de servidores con soporte transaccional (los objetos transaccional).
  - IMS TM (Information Management System – Transaction Manager):
    - Gestor de transacciones de IBM.
    - Parte de la arquitectura IMS.
  - CICS (Customer Information Control System):
    - También IBM pero más ligado con z/OS o z/VSE (Mainframes).
    - El más importante en el mercado de grandes servicios.
  - Tuxedo:
    - Servidor de aplicaciones (actualmente propiedad de Oracle).
    - Incluye muchas más funcionalidades (colas, seguridad, equilibrado de carga, …)

Transacciones de Larga Duración

- Modelos de transacciones caracterizados por:
  - Desde el comienzo de la transacción hasta su finalización (con éxito o fallo) pasan intervalos de tiempo largos (horas o días).
  - Algunos pasos requieren intervención manual (diferida), para determinar validez o no de determinadas operaciones.
- La granularidad de las tareas:
  - El acceso completo a un subservicio (se encuentra muy limitado, por cuestiones de restricción u operativas).
  - Eso implica que los objetos transaccionales (el elemento básico al cual se accede en exclusiva, se bloquea o en el que se da una colisión) es significativamente grande.

Escenario de Ejemplo



Cliente que quiere reservar una serie de servicios de viajes (vuelo, hotel coche).

Cada uno de los servicios los operan empresas diferentes (por lo tanto no se puede acceder a bloquear temporalmente ciertos recursos).

El usuario hace una petición, el sistema reserva (simple o múltiple) y espera una confirmación final por parte del usuario.

[Sne02] Snell, James. Automating business processes and transactions in Web services. IBM developerWorks, Aug 2002.
http://www.ibm.com/developerworks/library/ws-autobp/

Características e Implementación

- Es un escenario de gestión de procesos de negocio, típicamente implementado con BPEL y servicios web (WS).
- WS para control de transacciones:
  - WS-Coordination.
  - WS-Transaction.
- El servicio de viajes realiza reservas reales sobre los otros servicios y, en el caso de ser descartadas, se aplica una compensación al hacer el *rollback*.