BDLE
Transactions
à large échelle

HN 2014

Intro

- Contexte
  - Application transactionnelle à large échelle
    - Traitement de transactions "online" appelé OLTP
      - Transaction prédéfinie par une procédure paramétrée
        - Ex: gérer un panier d'achat
      - Mais on ne connait pas à l'avance les paramètres de la transaction
        - Ex: qui achète quels produits
    - Taille données proportionnelle au nbre d'utilisateurs : exple TPC-C
    - M utilisateurs dont N sont actifs simultanément
    - Besoin de traiter 10^3 à 10^6 transactions par minute
- Infrastructure
  - Besoin d'une solution à faible coût matériel, logiciel
- Demande non figée
  - De + en + d'utilisateurs
  - Possible fluctuation du nb d'utilisateurs (dans les 2 directions)

Intro

- Propriétés d'un SGBD
  - Transaction ACID
    - Atomicité d'une séquence d'opérations, Cohérence stricte, Isolation des utilisateurs, Durabilité face aux pannes.
- Système centralisé
  - Gère les transactions
- Problèmes
  - Passage à l'échelle
    - Augmenter le nombre de requêtes traitées
    - Augmenter le nombre de transactions traitées (tpm)
  - Disponibilité
    - Pas d'interruption du service

Type de transactions

- Transaction longue
  - Une transaction modifie un ensemble quelconque de données
  - Modifie de préférence une portion de la base relativement petite
  - Une transaction peut durer longtemps (ex: interactive, calculs)
- Transaction locale
  - Une transaction ne peut modifier que les données stockées sur une seule machine
- Transaction restreinte
  - Une transaction ne peut modifier qu'un sous-groupe prédéfini de données locales.
- Transaction courte
  - Une transaction est une séquence d'accès (RW) ciblés sur peu de données, sans dialogue avec l'utilisateur, sans calcul complexe.

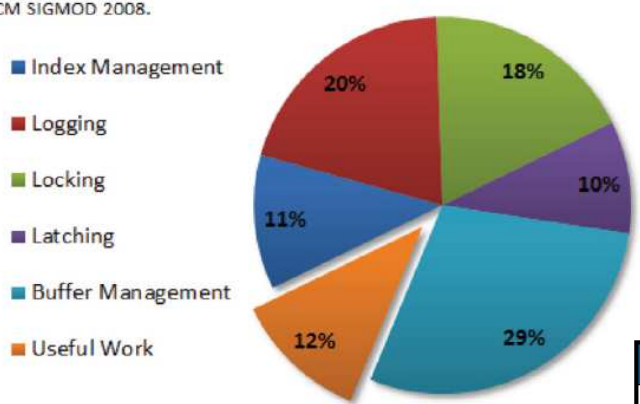
Types de solutions selon de type de transactions supportées

- SGBD Centralisé
  - Limite : Scale up (core, ram) mais pas de scale-out
  - Transactions longues
- SGBD parallèle
  - Données fragmentées
  - Agrégation logique de l'espace des données
  - Transactions longues
- Système partitionné (ex. DB Shards)
  - Données fragmentées
  - 1 SGBD indépendant par machine
  - Transactions locales
- Système **datastore non relationnels** dits NoSQL
  - Données fragmentées
  - Transactions restreintes à certaines données
    - Une seule donnée (ex. Cassandra)
    - Un seule groupe prédéfini de données (ex. KVStore)
- Systèmes émergents dits NewSQL (ex. VoltDB)
  - Données fragmentées
  - 1 SGBD simplifié par machine: mono utilisateur et très rapide (gain d'un facteur 8, voir illustration ci-après)
  - Transactions courtes mais non restreintes

VoltDB

General Purpose RDBMS Processing Profile

OLTP Through the Looking Glass, and What We Found There
Stavros Harizopoulos, Daniel Abadi, Samuel Madden, and Michael Stonebraker
ACM SIGMOD 2008.



	VoltDB	NoSQL	Traditional RDBMS
Scale-out architecture	✓	✓	
Built-in high availability	✓	✓	
Multi-master replication	✓	✓	
ACID compliant	✓		✓
SQL data language	✓		✓
Cross-partition joins	Automatic	In app code	In app code
Cost at Web scale	$	$$$	$$$$

Gestion distribuée de données

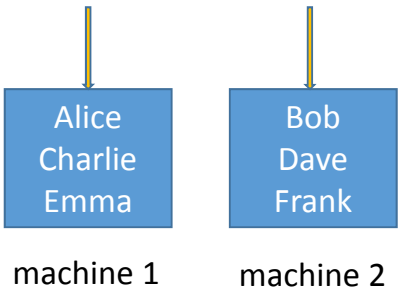
- Environnement : un système formé d'un réseau de machines
- Tolérance aux pannes
  - Données répliquées sur plusieurs machines
  - Théorème CAP: Quel accès aux données est possible en cas de panne du réseau ?
    - C: toujours lire l'état le plus récent d'une donnée répliquée
    - A: toujours répondre rapidement
    - P: toutes les machines ne sont pas joignables (panne réseau)
  - AP : lire une donnée immédiatement bien que les autres machines soient injoignables
    ⇒ la réponse est obsolète si la donnée a été modifiée sur une machine injoignable
  - CP : lire l'état le plus récent d'une donnée bien que les autres machines soient injoignables
    ⇒ Ne pas répondre immédiatement mais attendre de pouvoir contacter toutes les répliques
  - AC : lire rapidement l'état le plus récent
    ⇒ Toutes les machines doivent être joignables rapidement
- Compromis cohérence / latence
  - Répondre le plus vite possible en contactant une seule machine
  - OU répondre moins vite en contactant plus de machines

Scalable Datastore

- Système de stockage de données **non relationnelles**
- Distribué sur plusieurs machines
  - Conçu pour facilement à passer à l'échelle: Scale-out
- Tolérant aux pannes
- Priorité : latence faible
- Support restreint des transactions

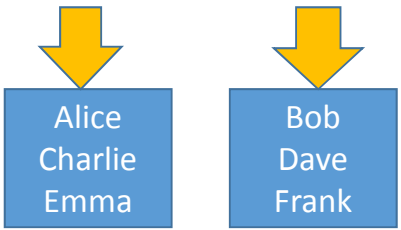
→ Système NoSQL transactionnel

Illustration du Scale-out



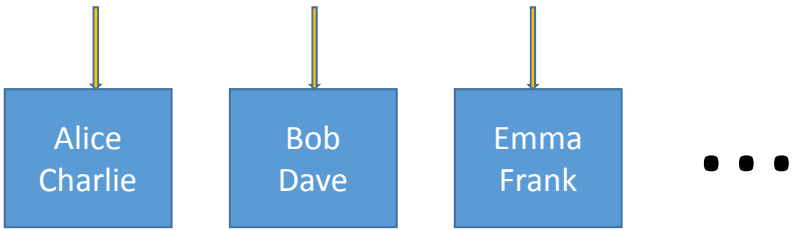
Scale-out: demande trop forte

Pb: transactions trop lentes car
demande > capacité de traitement



Scale-out: ajouter des partitions

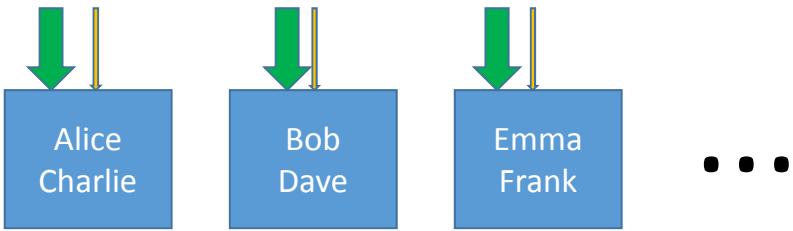
Ajouter des partitions "horizontalement"



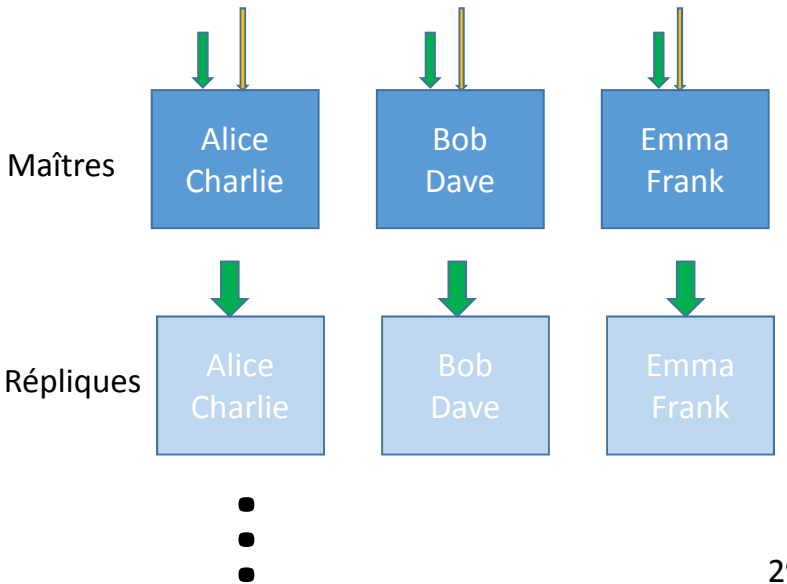
2e avantage: permet de gérer plus de données

Scale-out : lectures trop nombreuses

Pb: requête trop lentes



Scale-out : lectures trop nombreuses



Solution mono maitre
Un seul maître par donnée

2e avantage: tolère mieux les pannes

Ecriture: plusieurs protocoles

- Choisir le protocole le plus adapté aux exigences de l'application
- Le protocole d'écriture est spécifié par :

- Le **mode de propagation** vers les répliques

  Exprime le compromis : latence ./. cohérence

ET

- Le **niveau de durabilité** des écritures

  Exprime le compromis : latence ./. durabilité

Réplication avec propagation synchrone ou asynchrone

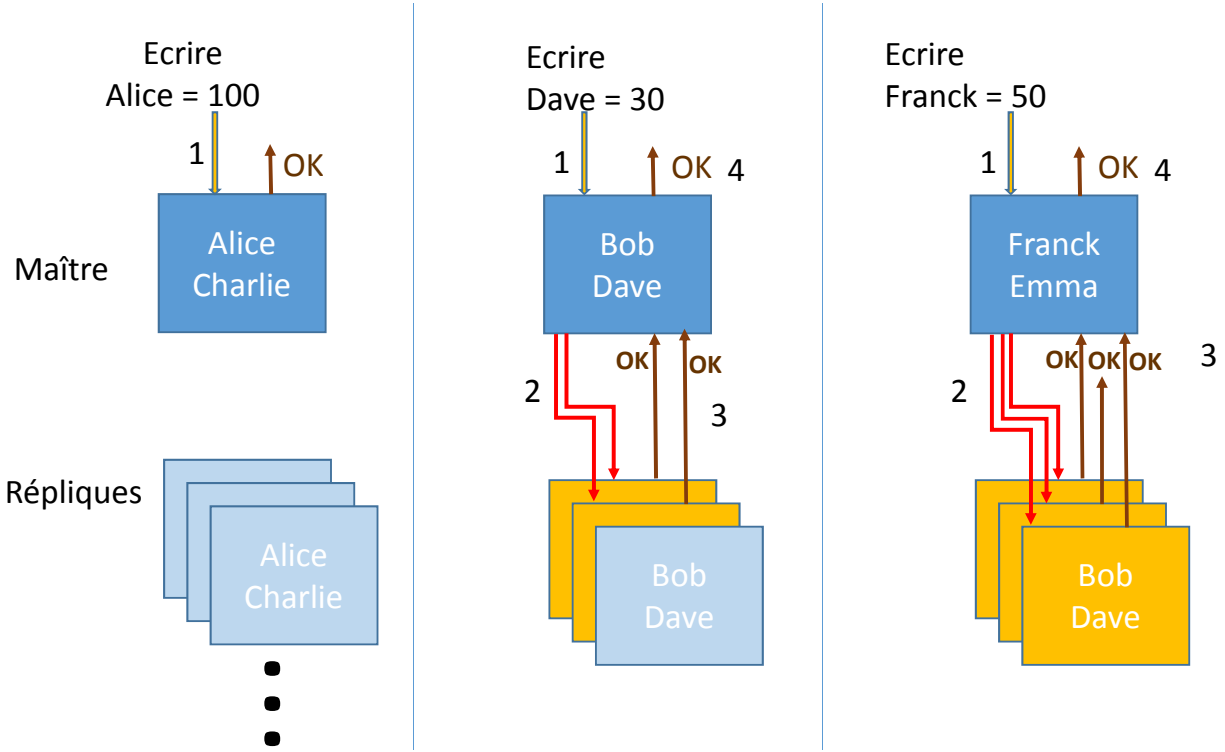
- Donnée de référence
  - Maître
- Plusieurs répliques
- Propagation des écritures vers les répliques
  - découplée ou non de la mise à jour du master
- Ecart possible entre les répliques
- Lecture d'une donnée qui n'est pas la plus récente.

Modes de propagation

Trois modes de propagation des écritures :

- Le maître ne propage pas
  - Il valide l'écriture localement puis répond à l'application
- Propager vers la majorité des répliques (n/2 + 1)
  - Attendre qu'une majorité de répliques valident avant de répondre à l'application
- Propager vers toutes les répliques
  - Attendre que toutes les répliques valident avant de répondre à l'application

Les 3 modes de propagation



Niveaux de durabilité des écritures

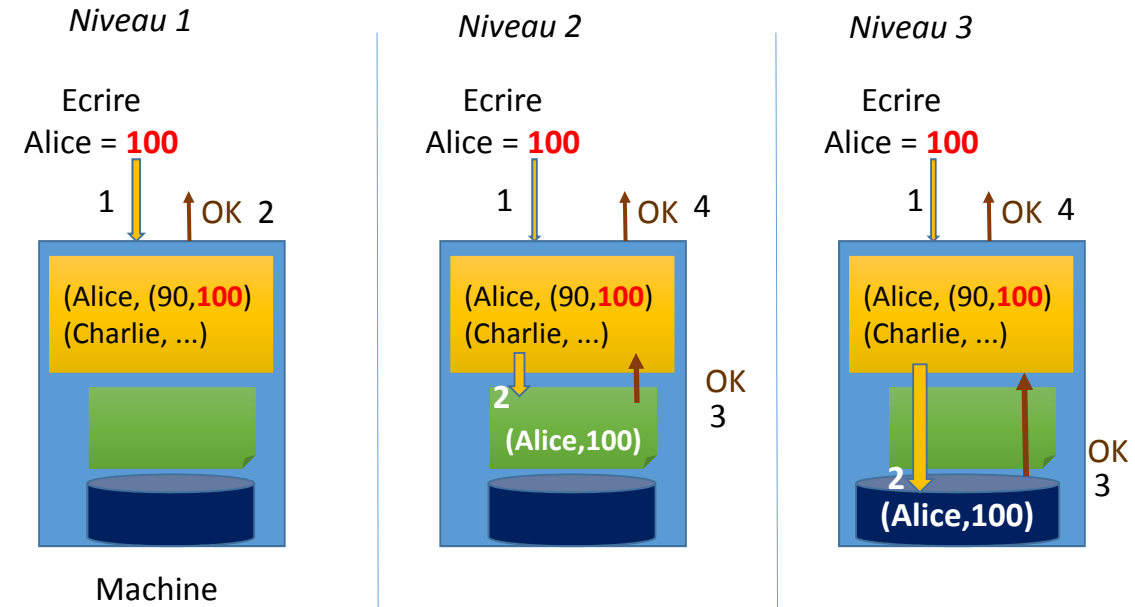
- Niveaux de durabilité croissante
  - Sur quel support écrire lors du commit ?
- Une écriture est durable si on dispose du journal pour restaurer la base
- Opération E = écrire 1 liste de paires
  - Compléter la base avec la nouvelle version des paires
  - Compléter le journal : ajout séquentiel (append)

Niveau 1 : Ne pas compléter le journal. Ecriture **non** durable
- Ajouter la nouvelle version des paires dans la base en mémoire
  - Modifier la Map<K, (V,version)> en RAM
- Perte possible de E sauf si d'autres répliques ont traité E également

Niveau 2 : Compléter le journal
- Niveau 1 + écrire dans le tampon (buffered write) associé au fichier du journal
- Tolère une panne logicielle du store mais pas de l'OS

Niveau 3 : Ecriture durable
- Niveau 1 + Niveau 2 + forcer à écrire le journal sur disque
- Tolère une panne de l'OS (reboot)

3 niveaux de durabilité



Donnée en RAM

Journal dans le buffer de l'OS

disque

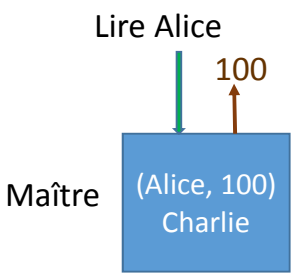
Ecriture flexible dans KVstore

- On peut préciser le protocole pour chaque écriture
  - put(K, V, mode)
- Le protocole est spécifié par
  - Le mode de propagation
  - Le niveau de durabilité du maître
  - Le niveau de durabilité des répliques
- Syntaxe pour le mode de propagation
  - NONE
  - SIMPLE_MAJORITY
  - ALL
- Syntaxe pour le niveau de durabilité
  - NO_SYNC
  - WRITE_NO_SYNC
  - SYNC

Niveaux de cohérence des lectures

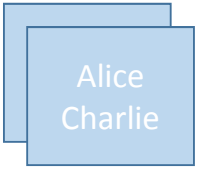
- DynamoDB
  - 2 niveaux
    - Stricte : dernière version
    - Quelconque : autre version (y compris la dernière)
  - V = get(clé, consistent)
    - avec consistent = true ou false
- Oracle NOSQL KVStore
  - Les niveaux de cohérence possibles sont :
    - Stricte
      - lire la dernière version: ABSOLUTE
    - Bornée
      - Lire une donnée dont la version >= N
      - Lire une donnée sur une machine dont le retard < A
        - Retard = date courante – date dernière mise à jour
    - Relâchée
      - Quelconque: Lire n'importe quelle réplique ou le master: NONE_REQUIRED
      - Faible: Lire n'importe quelle réplique sauf le master: NONE_REQUIRED_NO_MASTER
  - Durée tolérée: permet d'attendre qu'une version satisfaisante soit générée
  - V = get(clé, niveau, timeout)

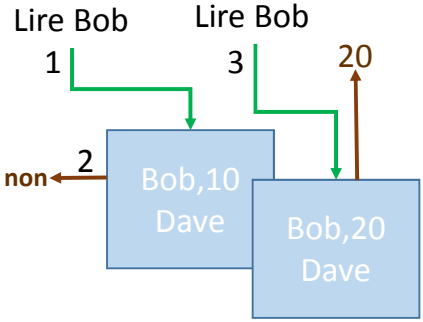
Les niveaux de cohérence des lectures



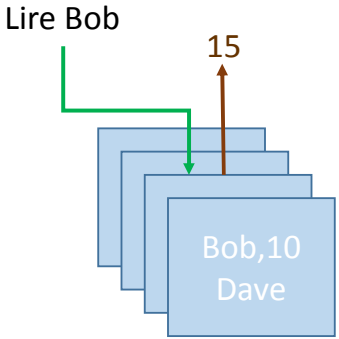
Maître
non accédé

Répliques





Plusieurs tentatives



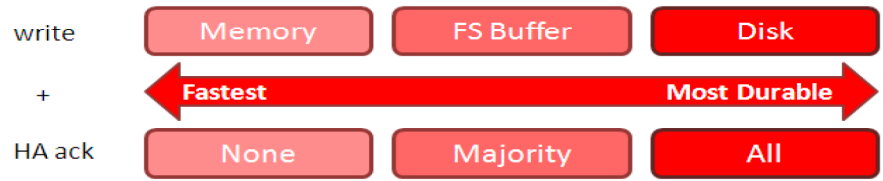
Lire la réplique la moins chargée

Exemple d'Oracle NOSQL

Transaction Durability and Read consistency

ACID Transactions – Configurability

- Configurable **D**urability Policy



- Configurable **C**onsistency Policy

ORACLE

6 | Copyright © 2012, Oracle and/or its affiliates. All rights reserved. |

Transactions locales

- Transaction locale : traitée sur une seule machine
  - Rapide car indépendant du nombre total de machines
  - Traitement possible en cas de partition du réseau
  - Tolérance aux pannes facilitée

- Problème: Spécifier la co-localité des données
  - donnée → machine

Spécifier la localité : dépend du modèle de données

- Table flexible
  - Sans schéma fixé à priori.
    - Un nuplet a une clé obligatoire
    - Un nuplet a un nombre quelconque d'attributs
  - On peut définir des groupes d'attributs
    - Une table a plusieurs groupes d'attributs
- Paire (Clé hiérarchique, valeur)
  - Clé formée d'une liste de termes
    - /a/b/c/
  - Structure hiérarchique (arbre)
    - Ex: /a/b1/c1/ /a/b1/c2/ /a/b2/c1/
  - Clé structurée en 2 composantes
    - Préfixe = Composante majeure
    - Suffixe = Composante mineure
    - Syntaxe : /a/b/ - /c/d/
- Donnée non modifiable
  - Modifier une donnée = ajouter une nouvelle version de la donnée

Localité selon le modèle des données

- Quelles données sont stockées sur la même machine ?
- Table flexible
  - (Clé , groupe d'attributs) → machine
- Clé hiérarchique
  - (composante majeure) → machine

Accès aux données : lecture

- Donnée:
  - (Clé hiérarchique, (valeur, version))

- Lecture ciblée d'une paire
  - (valeur, version) = get(clé)
- Lecture de plusieurs paires
  - Liste de (valeur, version) = multiget(K, intervalle, profondeur)
    - La composante majeure est un préfixe de K (lecture locale)
    - Intervalle et profondeurs précisent le sous arbre à lire

Isolation des accès

- Opérations
  - Lecture get(clé)
  - Ecriture Put(clé, valeur)

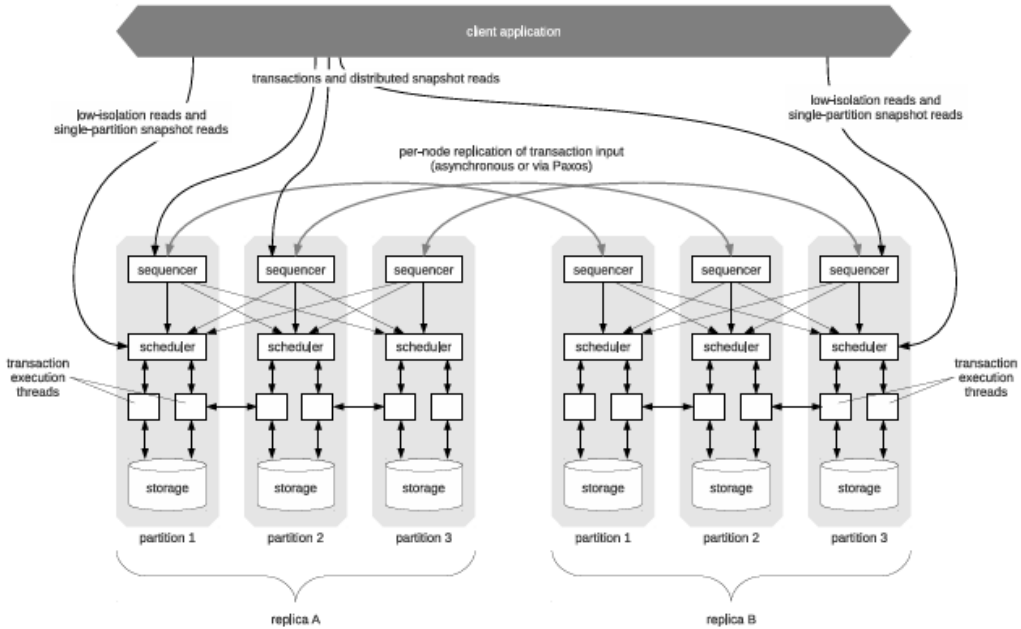
- Isoler la lecture-écriture d'une donnée
  - Suite de 2 opérations : (Lire A puis Ecrire A)
  - Contrôle optimiste
    - **Ecriture conditionnelle**
      - Ecrire A seulement s'il n'a pas été modifié entre temps
- Isoler plusieurs écritures
  - (Ecrire A, Ecrire B, ..., écrire Z)
  - Atomicité d'une séquence d'écritures conditionnelles
  - Co-localité des données A,B,...,Z

- Etude de systèmes existants

- Fiche de lecture sur
  - Calvin
    - http://cs-www.cs.yale.edu/homes/dna/papers/calvin-sigmod12.pdf
    - lire la section 3.2: Scheduler and concurrency control
  - TAO
    - https://www.cs.cmu.edu/~pavlo/courses/fall2013/static/papers/11730-atc13-bronson.pdf
    - Lire les sections
      - 4.4 Leaders and Followers
      - 4.5 Scaling Geographically

Yale's Calvin

Scalable transaction layer over shared nothing storage system
Sequencing, scheduling and storage layers
Deterministic locking, Logging transaction inputs, …



Facebook Tao

- Distributed data store for the social graph
- Eventual consistency
- Replaces Memcache as cache manager on top of MySQL
- FB data not easily partitionable
- Impossible to generate views presented to users ahead of time
- Request rates on some items could spike significantly

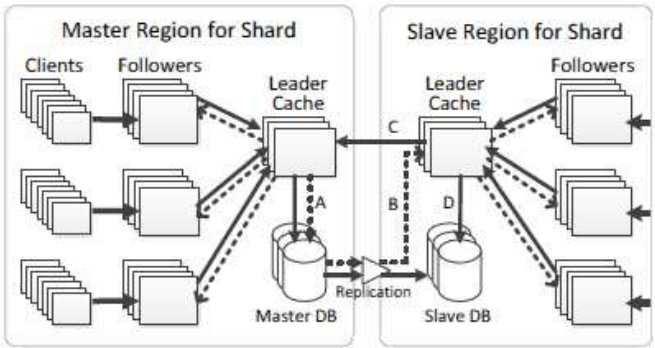


Figure 2: Multi-region TAO configuration. The master region sends read misses, writes, and embedded consistency messages to the master database (A). Consistency messages are delivered to the slave leader (B) as the replication stream updates the slave database. Slave leader sends writes to the master leader (C) and read misses to the replica DB (D). The choice of master and slave is made separately for each shard.

- Divers

Google Spanner

- Globally distributed semi-relational system, SQL-based query lang
- Transactional, relies on

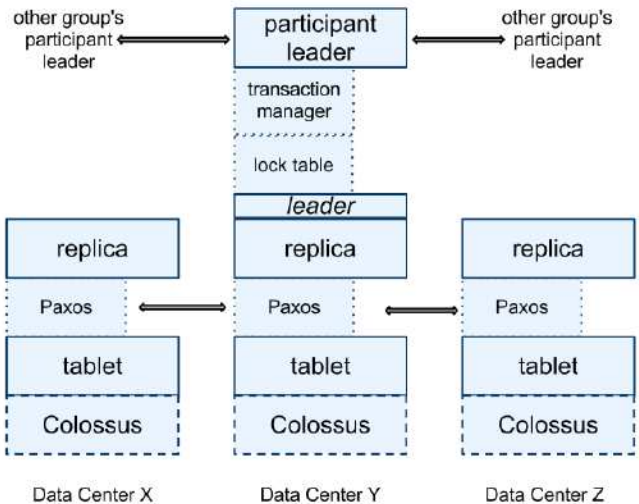


Figure 2: Spanserver software stack.

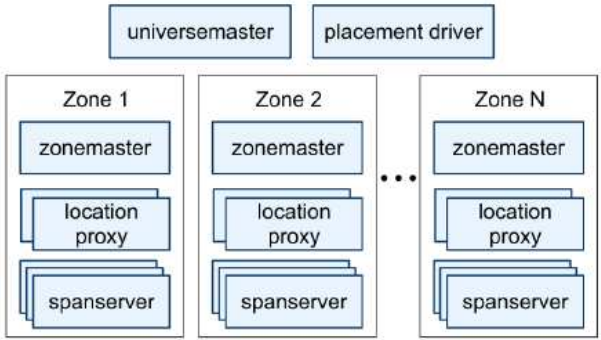


Figure 1: Spanner server organization.