Appunti NOSQL - Federico Torrielli

- Aggregate Aware (NOSQL) - Aggregate Ignorant (SQL)
- In NOSQL la "transazione" è atomica solo all'interno dell'aggregato
- **Skinny Rows**: poche colonne utilizzate, molto lunghe
- **Fat Rows**: molte colonne utilizzate, corte o lunghe
- **Materialized Views**: quando dobbiamo fare molte volte una query, tipo una select, non ha senso prenderle da database tutte le volte. La soluzione è cachare il risultato di una query in una view materializzata per renderla disponbile tutte le volte che chiamiamo la stessa query (o qualcosa di molto simile, con il minimo aggiustamento). NOSQL non ha la capacità di avere delle view, ma può materializzare i propri aggregati attraverso la computazione map-reduce.
- **Strategie per mantenere una materialized view**: o facciamo un update della view tutte le volte che aggiorniamo il singolo dato oppure effettuiamo dei batch jobs ripetitivi ad intervalli (*cron*) per evitare di sovraccaricare il DB
- Replication vs Sharding e Load Balancing
- **Master-Slave Replication** vs **Peer-to-Peer Replication** (+ Peers == + Resilience) e (+ Peers == - Consistency)
- Conflitti **Write-Write** sono comuni in P2P Replication:
  1. Possiamo coordinare tutti i peer, garantendo la consistency, ma aumentando notevolmente il network traffic
  2. Possiamo accettare che ogni tanto ci sia un conflitto
  3. Possiamo usare un approccio pessimistico: **Write Locks**
  4. Possiamo usare un approccio ottimistico: **Conditional Update** –> ogni client che fa un update testa prima il proprio risultato in un workspace privato
- Gli ultimi due non possono funzionare in una situazione P2P, dobbiamo infatti avere a che fare con la **Sequential Consistency**
- **Sequential Consistency**: tutti i nodi applicano gli aggiornamenti nello stesso ordine
- Possiamo evitare situazioni di inconsistenza facendo solo *transazioni nello stesso aggregato*
- **Finestra di Inconsistenza**: lasso di tempo dove un client può effettuare una read inconsistente, solitamente $<$ 1sec.
- **Eventual Consistency**: quando il db non rileva updates su un aggregato, *eventualmente* decide in un secondo momento di aggiornare i suoi replica set
- **Read-Your-Writes Consistency**: situazione in cui si effettua un refresh di una pagina dopo aver scritto un commento sulla stessa ma l'update non si è ancora propagato su tutti i nodi e quindi, collegandosi da un altro computer, si pensi che il commento si sia perso per sempre.
- **Sticky Session**: un modo per effettuare la consistenza di sessione "appiccicando" la propria sessione web ad un solo nodo del cluster
- **Version Stamps**: ogni documento preso ha un version stamp "allegato",

e ad ogni richiesta dello stesso documento basta controllare che il version stamp sia l'ultimo disponibile dal nodo.

- **CAP Theorem**: Non puoi avere più di due tra –> **Consistency, Availability e Partition Tolerance** | In teoria puoi avere AP ma avere anche la eventual consistency!
- ACID vs **BASE**: Basically Available, Soft State e Eventual Consistency
- **Write Quorum**: $W > N/2$ | Ovvero, i nodi che partecipano alla write devono essere più della metà dei nodi in replica
- **Read Quorum**: $R + W > N$ | Ovvero, i nodi che partecipano alla read e alla write devono sempre essere maggiori ai nodi in replica
- **Optimistic Offline Lock**: forma di update condizionale che implica una doppia read "scaramantica" per vedere che il valore non sia cambiato prima di presentarlo, funziona meglio se si ha un *version stamp*
- Tipi di **Version Stamp**:
  1. Counter monotono (e se i numeri diventano troppo grossi?)
  2. GUID (non posso compararli nel tempo)
  3. Hash della risorsa (non posso compararli nel tempo)
  4. Timestamp (occhio al sync del tempo!)
- **Vector Stamps**: *version stamps* ma per sistemi *P2P*. Teniamo un vettore di stamp dove ci scriviamo il valore dello stamp che ha ogni nodo e quando comunichiamo tra nodi, facciamo un *sync* dei loro vector stamps.