# CBD - Teste 2018

**- Objetivos parallel dbms:**

BD's paralelas conseguem melhorar a velocidade de processamento e de I/O através do uso de múltiplos CPUs e discos em paralelo. Essa carga pode ser espalhada por múltiplas máquinas. Oferece Fault Tolerance e High Availability caso a aplicação precisar de trabalhar continuamente mesmo que várias máquinas parem podemos usar múltiplas máquinas para fornecer redundância. Redução de latência pode ser obtida porque ao termos vários user e máquinas espalhados o user poderá ser servido pela que estiver geograficamente mais perto dele.

**- Vantagens e desvantagens intraquery e interquery:**

**Interquery:** Execução paralela de múltiplas queries geradas por transações concorrentes. Usado para aumentar o transactional throughput (usado para fazer scale up de transaction processing system para super grande número de transactions por segundo). Forma mais fácil de paralelismo para suportar numa shared-memory parallel database. Mais complicado em shared-disk ou shared-nothing. Locking e Logging tem de ser coordenado com mensagens entre os processadores. InterQuery oferece Alto rendimento (High Throughput).

**Intraquery:** Execução de uma única query em paralelo em múltiplos processadores/disks. O mesmo operador é executado por muitos processadores, cada um a trabalhar num subset de dados. Para acelerar long-running queries. 2 formas complementares de intraquery: **Intra-Operation** (paralelizar a execução de cada individual operation da query) e **Inter-Operator** (executar as diferentes operações da query expression em paralelo). A primeira escala melhor com paralelismo crescente porque geralmente o número de tuplos processados por cada operation é superior ao número de operations numa query. IntraQuery oferece Low Response Time.

**- Replicação leaderless:**

Em Leaderless Replication não existe nenhum leader, qualquer réplica aceita writes dos clientes. Existem implementações em que o cliente manda diretamente os seus writes a várias réplicas. Noutras, o coordinator node faz isto pelo cliente (o coordinator não enforça nenhuma ordenação particular dos writes, apenas age como proxy enviando os writes dos clientes aos nodes).

Quando um node fica offline é simplesmente ignorado, quando voltar a ficar online o cliente pode começar a ler a partir dele. Qualquer write que lhe falte (enquanto esteve off) não vão estar no node (return Stale values). Contudo, cliente pode saber qual é a up-to-date com o Version number.

Para recuperar missing writes temos 2 mecanismos: **Read Repair** (Cliente lê a partir de vários nodes em paralelo, se deteta Stales responses escreve os valores mais recentes na réplica outdated; funciona bem para valores freq. lidos) e **Anti-Antropy Process** (Datastores têm background processes que estão à procura de diferenças dos dados nas réplicas e resolvem os problemas). Usando a primeira valores raramente lidos podem estar atrás por várias réplicas (leva a **reduced durability**).

Quorum Reading e Writing. Para obter permissão de várias réplicas antes de write ou read. **Read Quorum - w + r » n.**

**Sloppy Quorum**: Writes e Reads continuam a precisar de w e r respostas ok mas essas podem incluir nodes que não estejam entre os designados n home nodes para um valor. Escreve para estes nodes. Assim que o node home voltar os writes feitos para os nodes temporariamente aceitos são enviados para ele (**Hinted Handoff**).

**- skewed e hotspot - Partitioning of data:**

Particionar data: espalhar dados e query loads pelos nodes de forma uniforme.

**Skewed:** Algumas partições vão ter mais dados ou queries que outras.

**HotSpot:** Uma partição tem uma carga desproporcionalmente maior que as outras.

Para evitar HotSpots a forma mais simples é atribuir records a nodes de forma aleatória (dados acabam com uma distribuição uniforme). Uma desvantagem é que temos de fazer queries a todos os nodes em paralelos para lermos itens pois não sabemos onde eles estão

**- partitioning by key range / partitioning by hash of key - comparar:**

**Partitioning por Key Range:** Consiste em atribuir ranges contínuos de chaves a cada partição. Com isto sabemos as fronteiras entre ranges, conseguimos escolher as fronteiras automaticamente ou manualmente, somos capazes de determinar que partição contém uma dada key e somos capazes de realizar requests diretamente ao node apropriado. Ranges das keys podem não estar espaçados de forma uniforme. Dentro de cada partição mantêm-se as keys ordenadas. Assim, os range scans vão ser mais fáceis. Uma chave pode ser um index concatenado para dar fetch a vários related records numa query. Contudo, certos padrões podem levar a hotspots.

**Partitioning por hash da key:** Usam-se hash functions para determinar a partição de uma key. Reduz o risco de skew e hotspots. Uma boa hash torna dados skewed numa distribuição uniforme. Os problemas deste método são a perda de eficiência das range queries e a perda de keys ordenadas.

**- Vantagens e desvantagens da arquitetura shared nothing:**

**Shared Nothing:** Cada node (máquina ou VM) usa CPUs, RAM e Discos que trabalham de forma independente. As **vantagens** são: Melhor preço/performance ratio, extensibilidade, availability e redução da latência. As **desvantagens** são a complexidade e a dificuldade de load balancing.

**- replicação síncrona e assíncrona**

**Replicação Síncrona:** O leader espera pela confirmação de write do follower antes de reportar sucesso para o user e de tornar visíveis as alterações para outros clientes.

**Replicação Assíncrona:** O leader envia a mensagem mas não espera pela resposta do follower.

**Vantagens rep. Síncrona:** Followers têm updated copy consistente dos dados e se o líder falhar continuam dispoíveis nos nodes.

**Desvantagens rep. Síncrona:** O write pode não ser processado se um follower síncrono não responder, o líder é obrigado a bloquear todos os writes até que a réplica síncrona esteja disponível e é imprático ter todos os seguidores síncronos (usar apenas 1 síncrono e o resto assíncrono)

O enfraquecimento da durabilidade de replicação assíncrona é quase inevitável quando temos muitos followers ou se estiverem distribuídos.

**- replicação single leader**

Uma das réplicas é designada de Leader e as restantes de Followers. O algoritmo de single leader funciona assim:

1. O Cliente realiza uma write query na BD. Esta é enviada para o Leader.
2. Leader escreve novos dados no seu local storage.
3. Leader manda os dados que foram alterados para os Followers. (Usando replication log ou change stream).
4. Cada Follower pega no log recebido do Leader e atualiza a sua própria cópia local da BD (writes são processados na mesma ordem que o Leader os processou)

**- hadoop:**

fodias-te