07 – OLTP Database 2

1. Latches

   A. Implementations

      i. TaS spinlock : efficient but non-scalable, not cache friendly
      ex) std::atomic<T>

      ii. OS Mutex : simple but non-scalable
      ex) std::mutex

      iii. Adaptive spinlock :
      if not acquired lock, then deschedule thread and store it at global parking lot. Threads checks parking lot before acquire lock
      ex) WTF::ParkingLot

      iv. Queue-based Spinlock : better performance than mutex, cache friendly
      ex) std::atomic<Latch*>

      v. Reader-Writer Spinlocks :
      allow concurrent read
      can be implemented on spinlocks

2. B+Trees

   A. Latch Coupling

      i. Search :
      acquire read lock at child, then release lock at parent

      ii. Insert/Deete :
      acquire write lock at child, if child is safe(no split or merge will happen), then release lock at ancestors

      better : acquire read lock at internal nodes(optimistic assume)
      if not, re-acquire write lock at parent nodes

      versioned : no read latch, version counters increases when updates.
      check whether version number is same when beginning of operation and end of operation

3. Judy Array

   A. Metadata are packed in 128 bit fat pointer called “Judy Pointer”
   Node Type, Population Count, Child key prefix, value, child pointer

   B. Node Types

      i. Linear Node : sparse populations
      maintain key array and child pointer array

      ii. Bitmap Node : typical populations
      maintain prefix bitmap and child pointers

      iii. Uncompressed Node : Dense Populations

4. ART

   A. 4 Node types (vary on # of childs)

5. MassTree

   A. Each nodes are B+Tree

   B. Optimized for long keys (able to use long digits)