1. 研究问题是什么？为什么这是个有意义的问题？为什么这是个有挑战性的问题？
2. CBDS上对组件进行更新时的一致性问题.
3. 因为这个问题很常见,处理不当会导致系统发生错误.
4. 首先需要保证正确性,其次需要及时进行更新并尽量减少对系统的干扰.而这两者之间又有某种trade-off的关系.
5. 为什么已有工作在这个研究问题上不足？在哪方面、哪种情况下不足？
   1. 已有方法有quiescence-based和tranquility-based两种
   2. quiescence-based approach可以保证正确性,但是由于对条件假设过于悲观,需要block所有可能相关的事务,导致性能大幅下降.
   3. tranquility-based approach只要求block某个组件的临近节点的所有事物,可以有效提高性能,但是由于假设太弱,不能保证正确性.
6. 为什么提出的新方法能有效地解决该研究问题？究竟是什么内涵使得这种新方法区别与已有的工作？表现在哪方面？
   1. 新方法(version-consistent)可以保证正确性.
   2. 新方法区不同于之前的工作(基于静态信息),基于动态信息,在组件不处于quiescence状态时也可以进行更新.
7. 新方法是怎么运作的？它有什么假设？为什么这些假设可行？
   1. version-consistent是指在一个root transaction以及它所有subtransaction组成的过程中,对同一个组件的多次调用,都是在这个组件的同一个版本上.
   2. 首先提出了”freeness”的定义,即当一个组件在某个root transaction及其所有可能的后续subtransaction的调度过程中,它还没有被调用过,或者将来不会再被调用.
   3. 接着提出了一个命题,”当一个组件满足freeness的条件时,对它进行的更新操作version consistent.
8. 新方法的可用性和有效性如何？它的使用场景和使用条件是什么？它是否适用于实用场景？
   1. 新方法总体来说对系统没有什么额外的要求.
   2. 它共有3种strategy,其中WF(waiting for freeness)是不实用的,CV(concurrent version)需要系统的支持,BF(blocking for freness)则可以适用于通常的场景.
   3. 它典型的应用场景是在连续的transaction中经常性地需要进行reconfiguration
9. 新方法是如何与已有工作进行比较的？这样的比较是否有效？是否有隐藏的正确性威胁因素？
   1. 比较了新方法的不同strategy与旧方法(quiescence-based)在同样的情况下(接近现实状态)的性能,指标有两个,更新前的准备时间(delay)和工作时间的损失(loss).
   2. 这样的比较是有效的,因为首先新方法的正确性是经过数学证明的.因此只需要进行性能上的比较.
   3. 有,新方法是基于动态信息的,需要在动态信息的获取与维持上花费额外的开销,论文中就这一点并没有详细的说明.
10. 新方法是否仍有什么不足？在什么情况下表现出来？是否有改进的可能？为什么及如何做？
    1. 在各个节点的动态信息的获取与维持上有不小的开销,在整个系统的组件数量较多,并且transaction的粒度比较小的情况下会比较明显.论文中给出的解决方案是采用on-demand set up,也就是只在需要的时候才开始获取并维持这些信息,但是仍然没有解决这个问题.
    2. 在存在环形调用关系的组件之间论文中所给出的management algorithm可能会存在问题.如果两个组件在同一系列的事物中可能出现循环调用,并且调用次数可能只有在运行时刻才知道,就无法知道应该在哪一次调用后去除对应的future边.
    3. 没有考虑网络问题导致的信息丢失.可能两个节点间的通信失败会导致will-not-use的notification丢失,使得future边无法被正确移除,使整个update过程被block.在论文的实验部分中,所做的模拟都是单机模拟,并没有考虑这个问题.虽然这个问题不会导致正确性受到影响,但是系统的性能会受到很大的影响.