Centiman事务处理系统 松散耦合架构并且尽可能避免同步，正常情况下唯一的同步是在最终判断事务是提交还是abort的时候，处理器收集所有的验证器的结果并组合，然后给出该事务是abort还是commit

每个事务有读集写集，每个验证器存在写集缓冲区，知道所有提交成功的事务的写集

sharded validation，在一组验证器上并行运行，验证器和处理器是可扩展的

watermark到底是什么?
我的理解就是一个时间戳，作为一个基准值吧，在watermark之前的事务全部已经完成

两大挑战
1.松散耦合的设计带来spuious aborts，就是验证器并不知道验证的全局结果，所以单个验证器就可能以为这个事务是可以提交的，就可能因此导致后续的验证出错，检测出错误的冲突
2.在普通OCC，我们可以允许事务绕过验证，只要保证读取了数据库的一致性快照，Centiman不要求事务的写操作 原子的写到数据库中，所以很难确保读取最近的一致性快照

watermarks允许验证器知道要abort的事务，并且精确维护这些事务的状态，并且异步传播给系统关于哪些事务已提交并已写入数据库的消息

Centiman结构
1.datastore 键值存储，但是除了key-value还得要一个timestamp，相当于加了个版本号
2.transaction processing subsystem（processors和validators）
3.a global master 监视系统性能，协调弹性扩展，处理故障恢复
4.发出事务的clients

sharded validation
Figure2 validate(i,RS(i),WS(i)) i是时间戳，RS(i)是执行到i时间戳时收集到的读集，WS是写集
写的是按时间顺序进行验证有无读写冲突，读集中有没有出现写集中写了的数据，出现了就直接abort，没出现则更新写集
有点没搞清楚
有可能会导致spurious aborts，解决方案就是通过broadcast commit decision给validators，如果同步执行这个操作，会完全消除spurious aborts但是会导致阻塞，所以

我们使用watermarks异步传播
Figure3和Figure4没仔细看，大概看了一下
Figure3就是加入了watermark，只需要验证Max（v,w）之后的有没有冲突就可以
Figure4就是检查所有[v,w]有没有交集，正确性我没看懂，为什么检查有没有交集就可以了

watermark验证

property3.1.如果记录r在时间t有watermark w（watermark是时间戳吗），那么在时间t，时间戳i<w的所有事务要么已经提交并更新r，要么再也不会对r有任何操作
Definition3.2：对于时间戳i来说，当未来不会install任何时间戳j<=i的记录，并且没有任何时间戳k>i的记录存在于数据库，我们说这个datastore处于快照i
Definition3.3：当所有的读操作都看到处于快照i的datastore相同的值和版本，就叫一个事务读取快照i

每个验证器维护一个从过去一直到现在的sliding time window
每个处理器维护一个本地processor watermark Wp，代表处理器中时间戳小于Wp的事务都已经完成，可能已经提交并install所有的写，也可能已经abort
每个处理器定期重新计算本地watermark，并且缓存其他处理器的本地watermark信息，并定期更新

故障恢复
每个处理器节点维护一个预写日志，这样就可以重做写操作 没细看

总的来说就是提出一个watermark

处理器被实现成一个线程

IP + port绑定了验证器处理器以及存储器，可以是多机多线程也可以是单机多线程
watermark的取值在处理器commit事务的时候添加更新的，而且watermark是一直不断添加的，就是事务一旦确定可以提交了，就更新添加这条记录的watermark
感觉咱们好像用watermark来判断
老师，您问的问题是watermark怎么解决的spuious abort是吗，我可能是ppt上没写清楚spuious abort的定义，就是假设事务i通过了验证器A的验证，但没有通过验证器B的验证，那么验证器A就会进行添加事务i的写集到验证器的WriteSetsA里面，就会导致一个实

际上被abort的事务将写集写到验证器的WriteSetsA里面，也就是WriteSetA被污染了，然后当对后续事务j进行验证时，验证器A可能会检测到i的写集和j的读集之间有重叠，造成验证失败。这是一个spuious abort的定义。然后watermark他是一个时间戳，在这个时间戳之前的事务是已经结束了，要么成功提交，把验证器的writeSet更新，要么就是已经abort，所以watermark加入就可以知道上面那种情况事务i的写集是没有用的，因为watermark是在时间戳i之后了，不会和事务j产生影响，这就避免了这种spuious abort