1. 内核、库和应用级

检查点可以由操作系统在内核级实现，操作系统在内核级透明地设立检查点并重新开始进程。这对用户来说是理想的。然而，尤其对于并行程序，大多数操作系统并不支持检查点。在用户空间，以一种较不透明地方式链接用户代码和检查点库。检查点和重启操作由运行时支持所掌控。这种方法使用广泛，因为它不需要修改用户程序。

现在的问题是目前大多数检查点库是静态的，这就意味着应用程序的源代码（或至少对象代码）必须是可得到的。如果应用程序是可执行代码的形式，它则不能正常工作。另外一种方法需要用户（或编译器）在应用程序中插入检查点函数。因此，应用程序必须被修改，透明度也就不能保证了。然而，它的优点是用户可以指定在哪个位置设立检查点。这有利于减小检查点的开销，因为检查点会消耗一定的时间和存储。

2. 检查点开销

在一个程序的执行过程中，它的状态可能保存很多次。这被表示为保存检查点所需时间。存储开销指的是检查点需要的额外内存和磁盘空间。时间和存储开销取决于检查点文件的大小。开销可能是巨大的，尤其当应用程序需要一个大的内存空间时。已经有许多技术被推荐，用来降低这些开销。

3. 选择最优检查点间隔

两个检查点之间的时间间隔称为检查点间隔。时间间隔增大可以降低检查点的时间开销。然而，这意味着失效后更长的计算时间。Wong 和 Franklin 推导出图 12-14 所示最优检查点间隔的表达式：

$$\text{最优检查点间隔} = \sqrt{(\text{MTTF} \times t_c)} / h$$

这里 MTTF 是系统的平均失效时间。MTTF 反映了保存一个检查点的时间开销，h 是在系统故障前的检查点时间间隔内，进行正常计算的平均百分比。参数 h 处于某个范围内。系统恢复之后，需要花费 h×(检查点间隔)的时间来重新计算。

图 1 两个检查点间的时间参数

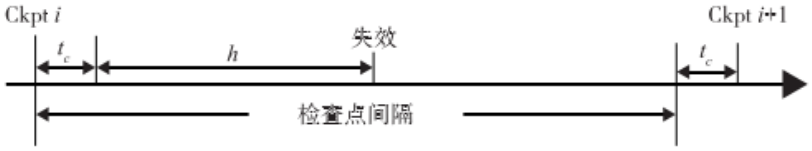

图 1

4. 增量检查点

相对于每个检查点保存全状态，增量检查点机制只保存与之前检查点相比发生改变的状态。然而，必须关注之前的检查点文件。在全状态检查点中，只需在磁盘上维护一个检查点文件，之后的检查点文件可以简单地覆盖此文件。在增量检查点中，之前的文件仍需要被维护，因为一个状态可能横跨许多文件。因此，总存储需求较大。

5. 分支检查点

大多数检查点机制是阻塞的，因为当设置检查点时，正常的计算被停止。如果有足够的内存，可以通过内存中程序状态的复制并唤起另一个异步线程同时执行检查点程序，以减少检查点的开销。一个简单的方法是使用 UNIX fork()系统调用计算重复检查点。分支子进程复制父进程的地址空间并设置检查点，与此同时，父进程继续执行。由于检查点程序是磁盘-I/O 密集的，重叠操作可以实现，进一步的优化可使用写时优化机制。

6. 用户指导检查点

如果用户插入代码（如库或系统调用）告知系统何时保存、保存什么以及不保存什么，检查点开销有时能够大幅度降低。检查点的准确内容应该是什么？它应该包含足够的信息帮

助系统恢复。进程状态包括其数据状态和控制状态。在 UNIX 进程中，这些状态存储在其地址空间，包括文本（代码）、数据、堆栈段和进程描述符。保存和恢复全状态的代价是昂贵的，有时甚至是不可能的。

例如，进程 ID 及其父进程 ID 是不可恢复的，在许多应用中它们也不需要被保存。大多数检查点系统只保存部分状态。例如，通常不保存代码段，因为在多数应用程序中其不发生改变。什么类型的应用能够被设置检查点呢？目前检查点机制需要程序是多机通用的（well behaved），精确的定义在不同的方案中有所不同。在最低程度上，通用程序应该不需要不可恢复的状态信息，例如进程的 ID 数值。

7. 并行程序检查点

现在来看并行程序检查点。通常并行程序的状态远多于串行程序的状态，因为它包括了独立进程的状态集合，以及网络通信状态。并行同时也会带来多种时间和一致性问题。

图 2 描述了一个三进程并行程序的检查点。标记为 x、y 和 z 的箭头表示进程间的点对点通信。三条分别标记为 a、b 和 c 的粗实线表示三个全局快照（或简称快照），这里的全局快照指检查点的集合（表示为点），每个检查点来自一个进程。此外，一些通信状态也可能需要保存。快照线与进程时间线的交点表示该进程设置（局部）检查点的位置。因此，程序快照 c 由三个局部检查点组成：s、t、u 分别是进程 P、Q 和 R 的检查点，以及保存的通信状态 y。

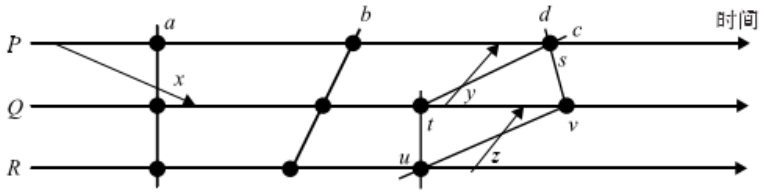

图 2 并行程序中的一致检查点和非一致检查点

8. 一致快照

如果一个进程在检查点处没有接收到消息，而这个消息并没有由其他进程发出，那么全局性快照称为一致的。在图形中，这相当于没有从右至左穿过快照线的箭头。因此，快照 a 是一致的，因为箭头 x 是从左向右的。但是快照 c 是不一致的，因为 y 是从右到左的。为了确保一致性，在两个检查点之间不应该有任何锯齿路径（sawtooth path）。例如，检查点 u 和 s 不属于一个一致性的全局快照。更苛刻的一致性要求需要没有箭头穿过快照，这样只有快照 b 是一致的，如图 2 所示。

9. 协作检查点和独立检查点

并行程序的检查点机制可分为两种类型。在协作检查点（也称为一致检查点）中，并行程序冻结，并且所有进程在同一时间设置检查点。在独立检查点中，进程彼此独立设置检查点。这两种类型可以通过不同方式相结合。协作检查点难以实现，并且需要巨大的开销。独立检查点则具有较小的开销，可以利用串行程序现有的检查点机制。