在选择异步多线程库时，学习成本、性能和优缺点都是重要的考虑因素。以下是针对上述提到的库的简要评估：

**C++标准库**

* **std::future 和 std::async**
  + **学习成本**：相对较低。这些特性是C++11标准的一部分，因此只需要熟悉C++和基本的并发编程概念即可。
  + **性能**：性能通常与直接使用线程和互斥锁相当，但在某些情况下，由于任务调度和线程管理的开销，可能不是最优的。
  + **优点**：简单、直接，与C++标准库的其他部分集成良好。
  + **缺点**：对于复杂的并发模式（如工作窃取、并行算法等）支持有限。

**第三方库**

* **Boost.Asio**
  + **学习成本**：中等。Boost库本身很庞大，但Asio部分相对独立且文档良好。需要理解异步I/O的概念和Boost库的惯用法。
  + **性能**：性能优异，特别适合网络编程和I/O密集型任务。
  + **优点**：跨平台、可扩展、支持各种I/O操作和协议。
  + **缺点**：Boost库整体较大，可能只使用Asio部分也需要包含整个库。
* **CppRestSDK**
  + **学习成本**：中等。专注于HTTP客户端，因此需要理解HTTP协议和RESTful API设计。
  + **性能**：对于HTTP客户端来说，性能良好，但可能不是最优的底层并发库。
  + **优点**：简单易用，支持现代HTTP特性和JSON处理。
  + **缺点**：专注于HTTP客户端，不提供通用的并发工具。
* **Intel TBB**
  + **学习成本**：中等偏高。TBB提供了复杂的并行算法和任务调度器，需要一定的时间来熟悉其API和并发模式。
  + **性能**：性能优异，特别是在数据并行和任务并行方面。TBB经过高度优化，可以充分利用多核处理器。
  + **优点**：功能强大、灵活，支持各种并行模式和数据结构。
  + **缺点**：学习曲线较陡峭，可能需要一些时间来掌握其最佳用法。
* **Qt Concurrent**
  + **学习成本**：取决于你对Qt框架的熟悉程度。如果你已经在使用Qt进行GUI开发，那么学习Qt Concurrent将相对容易。
  + **性能**：在Qt应用程序中，Qt Concurrent提供了方便的并发工具，性能通常与直接使用线程相当。
  + **优点**：与Qt框架集成良好，易于在Qt应用程序中使用。
  + **缺点**：如果你不使用Qt进行GUI开发，那么可能需要额外的学习成本来熟悉Qt框架。

**总结**

* 对于简单的异步任务，std::future 和 std::async 是学习成本最低的选择，并且与C++标准库集成良好。
* 对于网络编程和I/O密集型任务，Boost.Asio 是一个性能优异且功能丰富的选择。
* 如果你需要处理HTTP请求或构建RESTful API，CppRestSDK 是一个简单易用的库。
* 对于复杂的并行模式和数据处理，Intel TBB 是一个功能强大且经过优化的库。
* 如果你在Qt框架中开发应用程序，Qt Concurrent 是一个方便的选择。