**ASYNC I/O**

In computer programming, the async/await pattern is a syntactic feature of many programming languages that allows an asynchronous, non-blocking function to be structured in a way similar to an ordinary synchronous function. It is semantically related to the concept of a coroutine and is often implemented using similar techniques, and is primarily intended to provide opportunities for the program to execute other code while waiting for a long-running, asynchronous task to complete, usually represented by promises or similar data structures. The feature is found in C# 5.0, Python 3.5, Hack, Dart, Kotlin 1.1, and JavaScript, with some experimental work in extensions, beta versions, and particular implementations of Scala[1], Rust[2], and C++

**JavaScript:**

It’s important to first understand what asynchronous JavaScript is, and why we would want to make asynchronous calls. Asynchronous calls refer to calls that are moved off of JavaScript’s execution stack and do some work elsewhere. These are calls to an API. In Node’s case, they are calls to a C++ API in Node. Once the work is done, there is a function put in the event queue. Then when JavaScript’s execution stack is empty, the event loop pulls the function from the queue and pushes it onto the execution stack.

**LINUX:**

Linux database should use direct i/o. Direct i/o offer best performance for mix of sequential and random request that characterize database workloads. Compared to buffered i/o , direct i/o provide better random read , better cpu utilization and equivalent scan performance . And if the number of outstanding request is high enough , direct i/o approaches the performance of buffered i/o for random writes.

**Python:**

Asyncio is used as a foundation for multiple Python asynchronous frameworks that provide high-performance network and web-servers, database connection libraries, distributed task queues, etc.

Async IO is a concurrent programming design that has received dedicated support in Python.

* **Asynchronous IO (async IO)**: a language-agnostic paradigm (model) that has implementations across a host of programming languages
* **async/await**: two new Python keywords that are used to define coroutines
* **asyncio**: the Python package that provides a foundation and API for running and managing coroutines

**Multithreading**:

In most situations there is no need for asynchronous I/O, since its effects can be achieved with the use of threads, with each thread doing synchronous I/O. However, in a few situations, threads cannot achieve what asynchronous I/O can.