Performance

Every server application company in the world touts performance and very few delivers. Performance must be designed from the ground up. Performance cannot be created in thin air, be bought, through marketing, or achieved through over-engineering. There is a reason why most products cannot achieved claimed performance and the reason is because performance is hard work and many companies do not dedicate the time and effort to do what is necessary in research and development.

LiteSpeed is founded by a team with over 18 years of experience in low-level, low latency, highly scalable network applications: products such as a HA (high availability) server that must process real-time communication data within of one of the world's largest telecommunication giant.

One cannot speak of performance without mentioning scalability. Here is the definition of scalability as it relates to LiteSpeed and its customers.

Scalability (n): *the ability to maintain measurable and tolerant performance as demand/input increase dramatically with minimal addition to cost of operation.*

True overall performance is a blend of raw throughput and scalability. Do not be fooled. Raw performance and scalability are not mutually inclusive. One can be fast, but lack the critical scalability characteristics. For today's most demanding web operators, it is not how fast a server is when given 5 concurrent requests, but 500 and 1000 concurrent requests. Without scalability, raw throughput is just an unimportant fine-print.

When one think of fast and scalable server products, one might think of software that use multi-threading (MT), a design that spawn many threads or multi-processing (MP), a design that spawn many processes to handle multiple workloads. However, one might be surprised to know that of the fastest and most scalable server products developed today, many are based on the principle of an asynchronous event-drive (AE) architecture relying for the most part, a single thread/process.

There is not enough space or time to explain the entire concept but we will do our best to give the basics behind an asynchronous event-driven (AE) software architecture.

If the notion that multi-threading is not the most efficient way to scale a difficult idea to digest, one can simply take a quick peak around a typical data-center installation. Some of the most scalable products deployed in data-centers are firewalls, load balancers, cache proxies, and core routers. More likely than not, they are all based on the principle of AE-driven architecture and not of MT/MP design. Firewalls, load balancer, and such “hardware” products are nothing more than software products enclosed in a hardware box and even embedded ASICS are just software translated into silicone for speed.

LiteSpeed is based on the same software architecture trusted by many of the world's most respected companies and is neither black magic nor black smoke: marketing creation.

A fair question one might ask is how can a single threaded network application perform faster than a MT/MP based one. The answers is simply one word: efficiency, the ability to minimize all possible

operational overhead.

Context Switches

Each core within most modern CPUs can truly process one thread at any given moment. The emerging n-core products can operate n-threads at any single time where n is always a very small number. What this means is that MT /MP concept is a bit of a misnomer with overhead. MT/MP programming principle is designed to simplify how programmers understand and design concurrent workloads. In reality, each core within a CPU is only able to handle one thread with the Operating System kernel switching all the running threads in and out of the CPU so fast as to give one the perception that all threads are running in parallel when in reality all the threads are given time slices of available CPU cores, not that they are all running inside the CPU cores at the same time: the distinction marks a huge difference.

The act of moving threads in and out of the CPU is called a "context switch" and context switching levies overhead as each context switch results in moving (reading/writing) memory in and out of the CPU registers. Performance degrade when the number of contexts switches increase to a point as to cause extra latency and could lead into the worst case scenario where the CPU is spending more time performing context switches than actual computational work. Most modern systems can perform a large number of context switches with no performance degradation but the non-linear increase in the number of context switches required versus the linear increase in the number of threads/processes impose a scaling problem with MT/MP design when thousands of threads/processes are required to process thousands of concurrent requests.

Memory Efficiency

An Asynchronous Event-driven (AE) design has tremendous efficiency as compared to a MT/MP design. For a MT, multi-threaded design, a stack is is required for each thread and for MP, multi-processed design, both a stack and a heap is required for each process. Stack and heap are memory areas allocated by the Operating System to facilitate the function of running code. An AE design does not require such overhead in memory as it is the most part, a single thread/process.

Although such memory inefficiency does not automatically make MT/MT design slow, they are another performance roadblock to the design when massive scaling is required. Even the smallest design issues that perform well on small input can become scalability nightmares with very large concurrent input set.

As most system administrators know, memory is relatively cheap but one cannot waste memory even in a modern high-end server. When memory usage exceeds capacity operating systems activates swapping, disk based memory access, which result in dramatic performance lose.

Locking

Shared resource locking within MT/MP design is often the first lesson and performance warning on modern concurrent processing.

A MT/MP design dictates that each individual workload is offloaded to a special thread or process that is separate from the parent process. In most real-world concurrent workloads, the child threads/processes must communicate with each other or with the parent thread/process. To avoid having two threads or processes accessing the same shared memory or other resources, locking is used.

When a thread within a MT system perform a write lock on a shared resource, all other threads that wish to use the same resource must idle and wait for the write lock to be released: wasting valuable CPU cycles, adding extra latency, and artificially lengthening request queues. Often is the case when a thread is performing a slow write operation within a write lock, performance and scalability is literally thrown out the window as all other threads attempting write on the same resource is locked out.

A MP system has an even larger overhead when it comes to locking. Not only are the same MT locking issue present but an MP design requires an slower type of locking mechanism known as IPC which is normally performed through shared memory where in most cases a process must first acquire an semaphore to synchronize access to shared memory or resource. Once again, this is not necessarily a performance problem but another scalability roadblock.

A properly designed AE based software does not need locking since there are no additional threads or processes to create the need to share or communicate with each other: there is only one thread/process.

There is no black magic to an efficient design. The shortest distance to two points is a straight line. For LiteSpeed, efficiency means a design that removes the need for unnecessary context switches, memory allocation, and lock contentions present in a MT/MP based architecture.

If an architectural change can lead to a more scalable foundation, it is a valid question to ask why isn't everyone doing it. Like everything in life, nothing comes without a cost. The AE architecture is inherently much more difficult to design and implement correctly and as result, is not needed or attempted in but the most demanding environments.

In the end, companies can discuss scalable architecture all day long but there is something even more important: implementation. Asynchronous event-driven (AE) architecture is not proprietary knowledge nor is LiteSpeed Technologies the only company applying its principles to web severing technology. As result of LiteSpeed team's 18 year in-depth experience in the field however, we firmly believe our point to point implementation rivals and exceeds industry competitors as demonstrated by multiple benchmarks available on www.litespeedtech.com. Just because two web servers are built using the same principles doesn't make them equals in final implementation. Theory and design is one thing, final implementation and performance tuning is another beast altogether.

LiteSpeed Technologies is not about creating a good enough product with average performance characteristics. We dedicated the time in research, based our products on the best architecture, and created a fully optimized solution to achieve head turning performance.

The final word of approval ultimately remain in the hands of satisfied customers. Give LiteSpeed a try and discover why over 300,000 internet domains are currently powered by LiteSpeed web server, according to Netcraft June 2006 Survey.

.