Horizontally Scalable Systems

The term *horizontally scalable* refers to systems whose capacity and throughput are increased by adding additional nodes. This is in distinction to *vertically scaled* systems, where adding capacity and throughput generally involves replacing smaller nodes with larger and more powerful ones.

It seems that most technological advancements involve building increasingly complex systems that adapt to increasingly subtle inputs. But we should also recognize that improved understandings and technologies often enable us to build systems that are simultaneously superior to, and in many ways, much simpler than their predecessors. Horizontally scaled systems are based on protocols developed for client-server distributed storage and distributed computing systems. But, unlike their more complex cousins, their evolution has guided by the principles of maximum independence (loose coupling) and and parallelism.

Not all problems can be solved with a horizontally scalable architecture. But where they work, they work very well. This makes horizontally scaled architectures well-worthy of study.

Goals of Horizontally Scalable Systems

These systems started with many of the same goals shared by most distributed systems: scalability of capacity and throughput, reliability, and availability. But whereas earlier distributed systems attempted to provide single-system services and semantics, horizontally scaled systems offer simplified (and decidedly non-Posix) semantics. Most of these simplifications have a few basic (and highly inter-related) goals:

- improve scalability and robustness by eliminating the need for synchronization and communication between parallel servers.
- improve flexibility and performance by enabling the (non-disruptive) addition or removal of servers at any time.
- stateless protocols that permit requests to be arbitrarily distributed and redistributed among those servers.
- turn servers into standardized components, easily deployed and requiring little or no configuration.
- service protocols that are designed to minimize the potential modes of failure and enable simple recoveries.
- service protcols that are optimized for throughput (vs latency) over very large (e.g. continental) distances.
- service protocols that exploit the numerous optimizations (e.g. web cache servers) and other features that have come to be standard in networks that serve heavy HTTP traffic.

And, as a (not unintentional) side-effect, these systems are much simpler to design, maintain, deploy, and manage than other (more complex) distributed systems.

Horizontally Scalable Hardware Platforms

All services in horizontally scalable systems are exchanged via network messages. A node in such a system is simpy a computer that implements the specified protocols. The *unit of deployment* in most horizontally scalable systems is the node; When a new service or more capacity is required, we add a new computer system to the server farm. The node also becomes the *unit of replacement*; If a service providing element fails, it is common to take down the failed node and install or bring up a new one.

The node instruction set, operating system, memory capacity, and storage technologies are merely implementation details ... of little importance to the system design. In fact, ignoring deployment issues, there should be no problem running a service where every one of the server nodes was a different instruction set, running a different operating system/version, and using different network interfaces and storage controllers.

This high degree of platform independence has made it very common to run horizontally scalable systems on virtual machines. Adding new virtual nodes is much simpler than adding new hardware nodes. When a new server is needed, we simply clone a standard image, and boot up a new virtual machine. Thirty seconds later, a new server is running. A service run on a virtual machine may be a little less efficient than it would be if run on physical hardware, but the flexibility and ease of deployment and capacity balancing greatly outweigh any reduced efficiency.

Note, however, that it is not the case that all virtual servers are equivalent to generic desktops. High end VM servers may include GPUs, NVRAM, flash memory, or other special hardware than can be allocated to VMs. With access to such acceleration hardware, a service running in a virtual machine may be able to rival the performance of (much more expensive) custom-built hardware.

If a horizontally scalable system is comprised of many servers, and new servers can be added at any time, it is essential that they all have appropriate network connectivity (for client-facing traffic, back-end servers, and peer-to-peer replication). It is not practical to rack new switches and connect new cables each time a new server is to be integrated into a particular service. Rather, large computing facilities are moving towards *Software Defined Networking* where all servers (virtual or real) are connected to large and versatile switches that can easily be reconfigured to create what ever virtual network topologies and capacities are required for the service to be provided.

Storage is a similar problem. New virtual machines need virtual disks for the operating system to boot off of, and to store the content they will manage. Large computing facilities are also moving towards *Software Defined Storage*, where intelligent storage controllers draw on pools of physical disks (or SSDs) to create virtual disks (LUNs) with the required capacity, redundancy, and performance. The new servers have no idea what the underlying storage is, but merely use remote disk access protocols (e.g. Fibre Chanel or iSCSI) to access configured storage.

Horizontally Scalable System Architectures

The individual servers in a horizontally scaled architecture are highly independent. They do not share CPUs, disks, network interfaces, or a single operating system instance. This means it is unlikely that any single failure would affect multiple servers. Even enclosure components (e.g. fans and power supplies) or networking components (e.g. switches and routers) that might be shared by multiple servers have redundance. Such systems are said to have no *single point of failure* or a *shared nothing architecture*. Both mean that there is no single (shared) component whose failure could affect multiple systems. Such design is key to reliability and availability.

Even if there are no single points of failure in the computing and storage components, all of the servers in a single room (or campus) could potentially be taken down by a regional disaster (power failure, flood, earthquake, etc). For this reason, highly available, and highly reliable systems often make copies and include back-up servers that are far away (e.g. in other cities). Far separated facilities that are unlikely to be affected by a single regional disaster are often referred to as being in distinct *availability zones*. If the servers in one availability zone go down, the work can be handed off to servers in a different availability zone. The ability to fail over to distant copies and serves is sometimes called *geographic disaster recovery*.

Horizontally Scalable Software Architectures

The classic horizontally scaled system is a farm of web-servers, all of which have access to/copies of the same content. As demand increases, more servers are brought on-line. When each server starts up, it is made known to a network switch that starts sending it requests.

- Since HTTP is a stateless protocol, the switch can route any request to any server. This routing may be as simple as Round-Robbin, or it may be load-balanced based on measured response times to recent requests.

- Parallel servers have no need to communicate with one-another, and the only configuration a web server requires is knowing where to find the content it is serving.
- If a web-server crashes, the switch will stop sending it new requests. If the operations in the service protocol are idempotent, the client will time-out and retransmit the request, which will automatically be routed to a different server.

Horizontally scalable systems do not have to be stateless. RESTful interfaces enable stateful interactions with all of the the same advantages. Even non-RESTful services can benefit from horizontal scaling. Consider the *shopping carts* supported by many product web sites. The front-end web-servers are indeed horizontally scaled and stateless. Shopping cart display and update operations are forwarded from the responding web-server to a back-end database server. The back-end database server may be highly stateful, and not at all horizontally scalable. But if 99.9% of all requests can be satisfied by the front line of web servers, it may be much easier to build a (perhaps vertically scaled) database server to handle the shopping cart requests.

Even (highly stateful) storage systems (including databases) can be designed to be horizontally scalable. Distributed Key/Value Stores often divide the key namespace into a independent subsets (usually called *shards*), each of which is assigned to a small group (e.g. 3) of servers. Assigning multiple servers to each shard enables us to maintain data availability even if some servers fail. Imposing a low cap on the number of servers involved in any particular transaction enables the protocols to scale to thousands of nodes with no degradataion in performance. When we need to add capacity, we do not add more servers for a particular shard; rather, we increase the number of shards into which the name space is divided, and assign the new shards to new servers. Similar techniques have been employed to improve the scalability of block storage, object storage, and even file storage services.

Cloud Model

The *cloud model* may be more a business model (renting services from an independent service provider, vs. buying equipment) than an architecture, but it is very well aligned with horizontally scalable architectures:

- All services are delivered via (well standardized) network protocols.
- The cloud model opaquely encapsulates the individual servers behind a single highly available IP address. This enables the network to distribute requests among the available servers.
- The resources presented by a cloud service are intended to be abstract/logical and thin-provisioned. Horizontally scaled architectures are well suited to implementing and delivering such services.
- The flexibility to continously add, remove, and rebalance resources in a horizontally scalable system, makes it possible for a cloud service provider to easily accommodate great fluctuations in demand.
- Public cloud services are (in most cases) unlikely to be co-located with the clients, and so the service protocols must be optimized for (throughput) efficiency over WAN-scale communications links.
- Horizontally scalable systems' ease of deployment/management yields economies of scale that make it possible for cloud service providers to offer their services at very competitive prices. Consider, for instance, the cost of maintaining redundant computing facilities in many different cities. This is normal operation for a large service provider, but would be prohibitively expensive for the typical company to do for itself.

But it is worth noting that the cloud model does not necessarily mean that all data access becomes remote. We always have a choice:

a. send the data to the computation (e.g. remote data access)
b. send the computation to the data (e.g. Map/Reduce)

Many cloud storage providers also also offer Virtual Machine hosting, and for applications that run in those remote virtual machines, all cloud data access will be local. For many applications, the amount of raw data processed is much greater than the amount of output that is produced. In such cases, it may be both faster and more economical to run the computation on a VM server that is co-located with the data it will process.