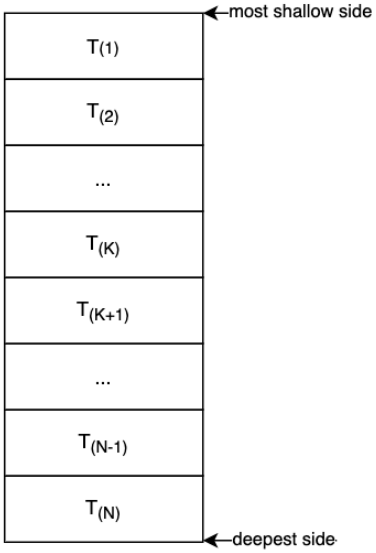
PART 1: A BRIEF EXPLANATION OF DNS, DOMAIN NAME SYSTEM

DNS, or Domain Name System is a distributed system that all machines, which are connected to the almighty Internet, have access to define themselves and search for other machines to send a request, forward a request, or other actions that require the address information of the targeted machine or machines.

It is designed as a hierarchical and decentralized naming system to be accessed from multiple places without burdening the system by avoiding a centralized structure. It translates human readable domain names into IP addresses of the target machine(s). 1985 is the year that the Domain Name System has been established in the USA, and it has become an essential part of the Internet. Its design consists of multiple-multilayered authoritative name servers to hold and store each domain in the Internet, which brings out a fault-tolerative structure.

The IETF, who has built and maintained the DNS has also defined a proper protocol to enable the DNS to communicate with the machines that sent requests to unravel the IP address of the machine which is known by the client as a human-readable domain name. This structure has its own key points however, and unless they are satisfied, there would not be such a system that is beneficial to maintain. Efficiency is the point which has the utmost importance among all the key points, thus the IETF has designed DNS to have its own data types, communication protocols and data storage methods.

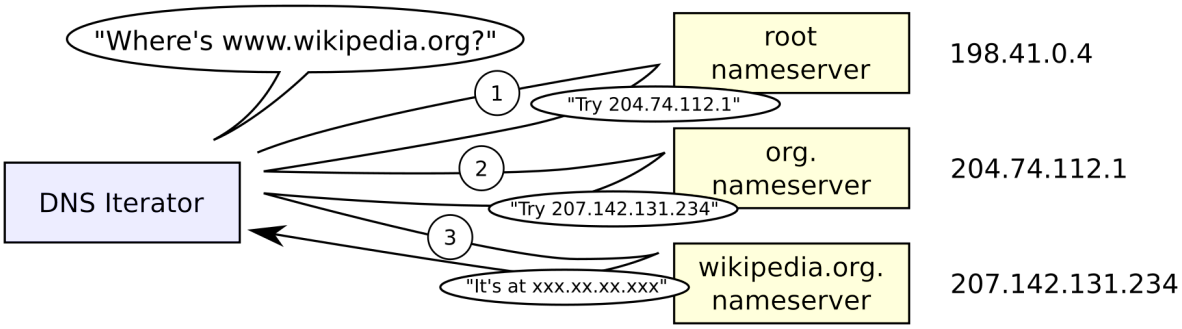


The underlying logic inside DNS is a recursive request mechanism. To be more concrete about the DNS, an example would be needed. Suppose that there's a human-readable domain name D, which consists of non-empty alphanumeric strings, which may include only the hyphen "-" additionally, concatenated with the dot sign "." and D is defined as $T_N.T_{N-1}.T_{N-2}....T_K....T_2.T_1$. By splitting the domain name D from the dots, we get N tokens that define a sub-domain. Then, putting those tokens into a stack, starting from the leftmost one, we have a stack structure which has T_N as its deepest token and T_1 as its most shallow token. It is okay to say that even though we've started to push the tokens from the leftmost side of domain name D, we've named each token by starting the leftmost side, since DNS works as a deductive procedure, which starts from the most general and finishes at the least general.

The procedure continues as follows: Starting from one of the initial servers that are initialized all around the world and considered as the entry points of the DNS structure, the first token inside the stack T_1 is popped and requested through the selected initial server, by asking "What is the IP address that holds information of the machine(s) which has(ve) a domain name T_1?". As a reply, either the requested IP is provided by the server or the request is forwarded to another initial server to find the IP address.

Let's assume that the request is handled without an error, thus we have the IP address that holds information related with the T_1 subdomain. Therefore, we pop another token from the stack, which is T_2, and re-apply the procedure with one change: The server that we request is changed from one of the initial servers to one of the servers that hold T_1 subdomain information. If such a domain exists, the machine that holds the information of that domain is returned as a reply from the requested server. Otherwise, the system tries other possible machines until there is none. If there's no IP address that holds the information of the subdomain T_2, the request is replied with an error.

This recursion continues until there's no token in the stack or an error occurs. From the start until the very end, the procedure aims to find the IP addresses of a domain name D defined as $T_N.T_{N-1}.T_{N-2}.\ldots.T_K.\ldots.T_2.T_1$ by iterating each subdomain T_K for all T_K be non-empty alphanumeric strings including the hyphen sign "-" that D consists.

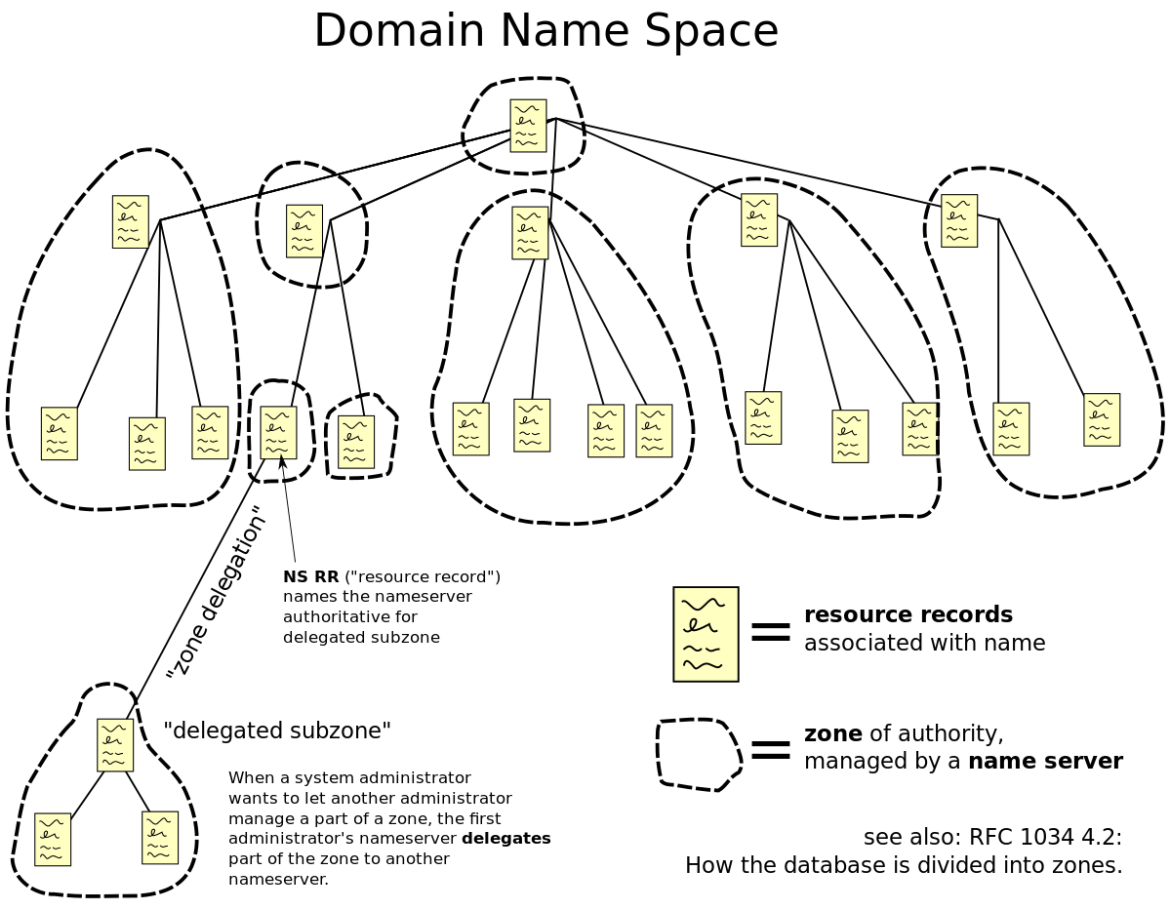


For a system that has so many distinct qualities, which is one of the backbones of the concurrent era's information retrieval procedures, there's no mistake in saying that it has distributed data all around the world. However, this does not imply that several of the servers have a fraction of its data as replicas of one or many other servers in the system. For the initial servers, which are the entry points of the system, it has to be replicas of one another since any downtime on one or many of those servers would lead to a disaster for all clients in the world. Therefore, we can infer that DNS has a hybrid data storing approach to function properly.

In addition to the data distribution methods, DNS has a way of scaling with fault tolerance. Each organization in the DNS manages has its own information and naming servers, thus any problem that disables an organization's naming server does not disable other organizations' naming servers.

In addition to these, DNS has a multiple machine-multilayered structure, thus each naming server at each layer may have different hardware specifications since at each layer, the number of requests that the machines have to respond may not be the same with a machine in another layer.

For the initial root naming servers, network capabilities and processing power should be maximum since all requests that are made by the clients from all over the world have to arrive at those entry points to be processed. But an institutional naming server inside DNS should not have that much network capabilities and processing power unless the institution has a lot of spare money. As for operating systems, all the naming servers inside DNS should be using Unix based operating systems, a lightweight one would fit the requirements. A GUI is unnecessary to have in a naming server, thus Unix based command-line based (CLI) operating systems are sufficient to manage the naming servers.



PART 2: SLURM, SIMPLE LINUX UTILITY for RESOURCE MANAGEMENT

SLURM, as in short of Simple Linux Utility for Resource Management, is a decentralized job dispatcher for high-end computers, supercomputers or computer clusters. It is published as an open source and free job scheduler for Unix-based kernels, especially Linux. 60% of the TOP500 computers all around the world use SLURM as their job scheduler.

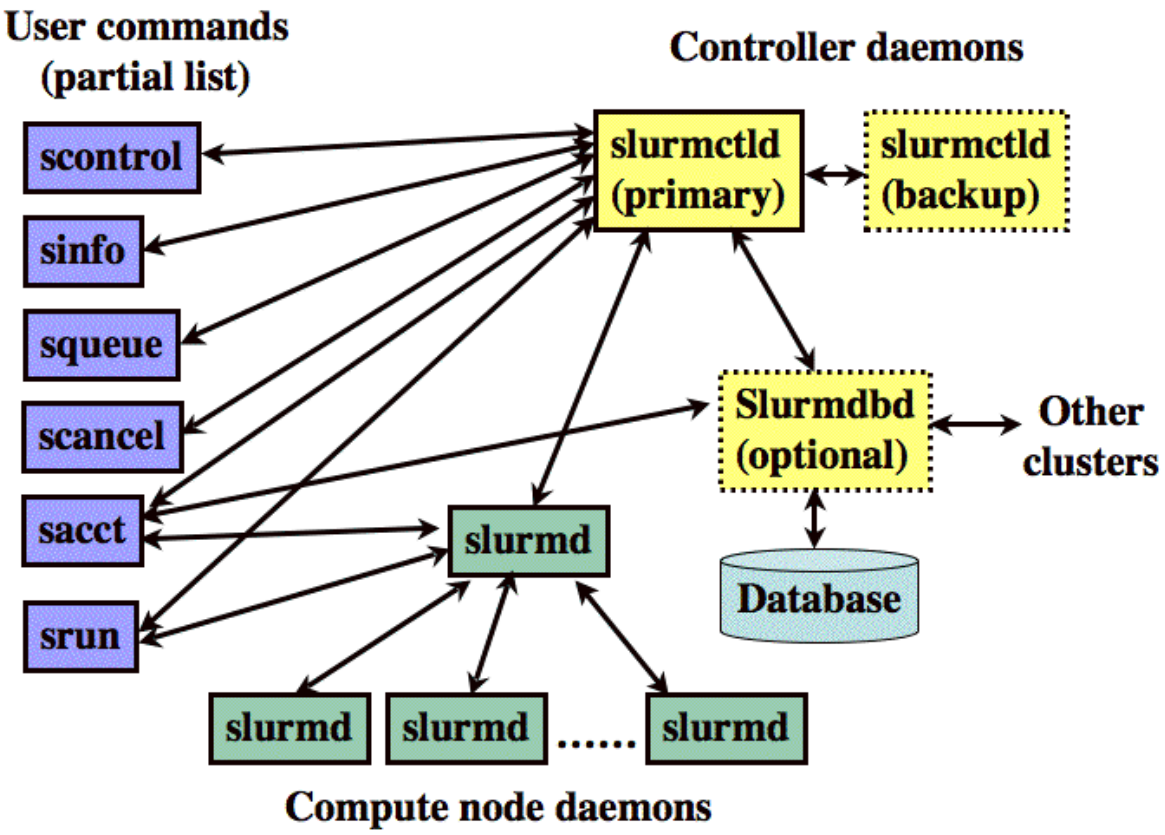
It aims to provide three critical functionality, which are:

1 - Allocation of resources with exclusive and/or non-exclusive access to users for a duration of time, which leads to performing any work.

2 - Providing a framework for launching, executing, and supervising jobs, typically a parallel job for several nodes in the system, such as Message Passing Interface(MPI)

3 - Resolving the resource allocation by a queue of demanding workers

Slurm tries to utilize the best fit algorithm for the job scheduling, by deriving a hybrid version of Hilbert Curve Scheduling and fat tree network topology. Therefore, it aims to maximize the benefits of locality of the tasks, assigned to computers which works as parallels.



Its structure is shown above, which has a primary unit called slurmctld, which is the controller daemon unit of the system, that receives the commands from users and invokes the necessary action to handle the requests. In case of a failure of the primary unit, its backup is here to take over the jobs from the primary control daemon unit. In addition to that, it has a database daemon for necessary parallel jobs on databases, such as bank transactions. Slurmdbd, which is a slurm database daemon, is responsible for such a task.On the other hand, the incoming job requests are dispatched to a slurmd, which is a slurm daemon that is connected to all other slurm daemons, all of which are the computer

nodes in the whole system. They have power and resources to handle the incoming job requests and combine their result into a single response message, which goes through slurmctld, to the end user.

It can be said that Slurm has a fault tolerant structure, since the primary control daemon has a backup in case of a failure, therefore dispatching jobs won't be disrupted. Also, any other daemon failure won't result in a catastrophic failure since any daemon node can be omitted from the system at any time, due to its decentralized structure. A user may command a computing node to run, but if it's unable to handle a job, any available computing node can be used to complete the task.

Also, it has database access from the database control daemon of Slurm, which is utilization of one or many databases at once. Furthermore, by using MSLURM, which is multiple environment slurm management, multiple databases can be controlled and accessed via multiple jobs at once, by using multiple read/write locks.

In overall, it has distributed data in its structure. We can say that the job scheduling instructions set by the users are inside the primary control daemon, and also some of it is inside the backup daemon, therefore there's a part of data that is a replica. However, in the computing nodes, there's non-overlapping data which is related to the ongoing job on each distinct node. Even if two computing nodes have the same job instructions, they may have different configurations and conditions, therefore it is safe to say that SLURM includes both replica data and non-overlapping data in its structure.

As a decentralized system that schedules jobs for supercomputers and computer clusters, scalability is the key of everything. It is easy to add or remove a node from the system, since every node has its own resources and environment to run a job. Therefore, any computing node that has a Unix-based operating system on it may be merged with the SLURM infrastructure, which leads us to a solution of the scalability issue.

As for the nodes, the physical infrastructure or hardwares is not necessary to be the same, since every node has its own purposes to be utilized. But, as I've explained above, Unix based operating systems inside the nodes is a must. There's no single operating system for the system, yet it has to have a Unix-based instruction set to be controlled completely. As the system's older name is an abbreviation of "Single Linux Utility for Resource Management", it would be nice that its nodes should have Linux as its operating system, but it is not restrained to only Linux.

RESOURCES:
1 - For DNS: https://en.wikipedia.org/wiki/Domain_Name_System
2 - For SLURM: https://en.wikipedia.org/wiki/Slurm_Workload_Manager,
https://slurm.schedmd.com/pdfs/slurm.sc08.bof.pdf,
https://slurm.schedmd.com/mslurm/mslurm_overview.pdf