The **Mass Storage Control Protocol** (**MSCP**) was a [protocol](http://en.wikipedia.org/wiki/Protocol_%28computing%29) designed by [Digital Equipment Corporation](http://en.wikipedia.org/wiki/Digital_Equipment_Corporation) of [Maynard](http://en.wikipedia.org/wiki/Maynard%2C_Massachusetts), [Massachusetts](http://en.wikipedia.org/wiki/Massachusetts) for the purposes of controlling their high-end [mass storage](http://en.wikipedia.org/wiki/Mass_storage) options.

First implemented in the [HSC50](http://en.wikipedia.org/wiki/VMScluster) hierarchical storage controller, the protocol quickly spread throughout the entire line of mass storage controllers built by DEC. The **UDA50** was an implementation of MSCP built on a [Unibus](http://en.wikipedia.org/wiki/Unibus) card; other implementations (for example, the **RQDX**) stretched down to the [Q-bus](http://en.wikipedia.org/wiki/Q-bus) and small, 5 megabyte disk drives and even [diskettes](http://en.wikipedia.org/wiki/Diskette).

Designed to minimize the amount of [CPU](http://en.wikipedia.org/wiki/Central_processing_unit) involvement, the protocol depended upon two queues. Into one queue were placed [packets](http://en.wikipedia.org/wiki/Packet_%28information_technology%29) which fully described the commands to be executed by the mass storage subsystem. To initiate an I/O request, the CPU had only to create a small data structure in memory, append it to a "send" queue, and if that was the first packet in the send queue, wake the MSCP controller. After the command was executed, an appropriate status packet would be placed into the second queue to be read by the CPU.

Interrupts to the CPU (a costly operation) were not needed so long as further command packets remained in the command queue and the response queue was not in danger of over-flowing. I/O-space reads and writes to the MSCP controller, a less-expensive but still-costly operation, were similarly minimized.

Because MSCP packets were deliberately designed to resemble the packets exchanged on the [VMScluster](http://en.wikipedia.org/wiki/VMScluster) interconnects, it was a very inexpensive operation to ship storage requests around a VMScluster for remote execution; this greatly facilitated the creation of large-scale VMSclusters. The dependence upon in-memory packets and the minimization of interrupts and I/O-space reads and writes greatly facilitated remote operations.

In 1986, DEC added VAXclustering support to their [MicroVAX](http://en.wikipedia.org/wiki/MicroVAX) minicomputers, running over [Ethernet](http://en.wikipedia.org/wiki/Ethernet) instead of special-purpose hardware. While not giving the [high-availability](http://en.wikipedia.org/wiki/High-availability_cluster) advantages of the CI hardware, these *Local Area VAXclusters* provided an attractive expansion path for buyers of low-end minicomputers.

Later versions of OpenVMS (V5.0 and later) supported "mixed interconnect" VAXclusters (using both CI and Ethernet), and VAXclustering over DSSI ([Digital Systems and Storage Interconnect](http://en.wikipedia.org/wiki/Digital_Storage_Systems_Interconnect)) and [FDDI](http://en.wikipedia.org/wiki/FDDI), among other transports. Eventually, as high-bandwidth wide area networking became available, clustering was extended to allow satellite data links and long-distance terrestrial links. This allowed the creation of *disaster-tolerant clusters*; by locating the single VAXcluster in several diverse geographical areas, the cluster could survive infrastructure failures and natural disasters.

VAXclustering was greatly aided by the introduction of [terminal servers](http://en.wikipedia.org/wiki/Terminal_server) using the [LAT](http://en.wikipedia.org/wiki/Local_Area_Transport) protocol. By allowing ordinary serial terminals to access the host nodes via Ethernet, it became possible for any terminal to rapidly and easily connect to any host node. This made it much simpler to accomplish [fail over](http://en.wikipedia.org/wiki/Fail_over) of the user terminals from one node of the cluster to another.

Eventually, VAXclusters reached the point where the cluster as a whole essentially never went down. *Rolling upgrades* even allowed the system operators to upgrade the OpenVMS system software, shutting down, upgrading, and rebooting individual nodes while the cluster as a whole continued processing. Cluster uptimes are frequently measured in years with the current longest uptime being at least sixteen years.[1]

As mentioned above, OpenVMS now also runs on [Alpha](http://en.wikipedia.org/wiki/DEC_Alpha) and [IA-64](http://en.wikipedia.org/wiki/IA-64) systems, so the term *VAXcluster* has been replaced by *VMScluster*. With [Gigabit Ethernet](http://en.wikipedia.org/wiki/Gigabit_Ethernet) now common and [10-gigabit Ethernet](http://en.wikipedia.org/wiki/10-gigabit_Ethernet) being introduced, standard networking cables and cards are quite sufficient to support VMSclustering.