

# Distributed MDSplus Database Performance with Linux Clusters

D.H. Minor and J.R. Burruss

*General Atomics, P.O. Box 85608, San Diego, California 92186-5608, USA*

---

## Abstract

The staff at the DIII-D National Fusion Facility, operated for the USDOE by General Atomics, are investigating the use of grid computing and Linux technology to improve performance in our core data management services. We are in the process of converting much of our functionality to cluster-based and grid-enabled software. One of the most important pieces is a new distributed version of the MDSplus scientific data management system that is presently used to support fusion research in over 30 countries worldwide. To improve data handling performance, the staff is investigating the use of Linux clusters for both data clients and servers. The new distributed capability will result in better load balancing between these clients and servers, and more efficient use of network resources resulting in improved support of the data analysis needs of the scientific staff.

*Keywords:* MDSplus, Distributed Client, Thin Client

---

## 1. Introduction

MDSplus [1] is a data acquisition and storage system. Developed jointly by the Massachusetts Institute of Technology, the Center for Nuclear Research (Padua, Italy), and the Los Alamos National Lab, MDSplus is the most widely used system for data management in the magnetic fusion energy program. MDSplus stores data from an experiment in a single, self-descriptive, hierarchical database.

MDSplus allows for access to data through either a traditional thin client or a new distributed client. The traditional MDSplus thin client method puts more load on the server, because the server performs all of the expression evaluation, data compression and decompression. In fact the client does nothing but send and receive messages to and from the server. In contrast, the distributed client performs the evaluation, data compression and decompression instead of querying the server.

At General Atomics, we use MDSplus both for data storage and as a proxy for data retrieval from a locally created database named PTDATA [2]. Our intention is to use clustered Linux workstations to test the performance of distributed MDSplus for normal

data retrieval, data writing, and for proxy data retrieval of PTDATA data. As faster, clustered Linux clients become available, it makes sense to offload work traditionally done on a single server to the new clusters of machines. We want to load the distributed MDSplus client onto these clusters and investigate how much it improves performance. We also want to investigate whether it is possible to distribute data across multiple server clusters, further decreasing the load on any one server machine, and removing another potential performance bottleneck.

As a database, MDSplus is hierarchical. Data are organized into trees, each of which contain nodes; much like a filesystem with directories and subdirectories, trees contain nodes and subnodes. Trees can be linked to other trees through subnodes, but the relationship is always hierarchical. Nodes can provide structure, or can contain data. Node data exists as expressions; the expression language of MDSplus is known as Tree-Data Interface (TDI).

Figure 1 illustrates the traditional thin client MDSplus approach. Under thin client, multiple client machines connect to a single MDSplus data server. Lightweight processes on the client machines (repre-

sented here by the small dots) connect to the server, where corresponding server processes are created (large dots). The server processes are CPU- and memory-intensive processes which do all of the query processing and expression evaluation. These server processes persist until the client closes its connection.

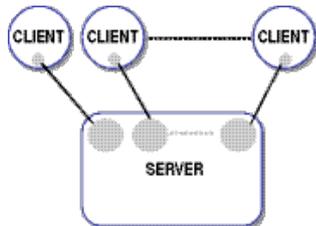


Fig. 1. Each client connects to a single server under thin client MDSplus.

Thin client is well suited for system configurations where client machines are old, slow, and on slower networks than the server machines. This has traditionally been the situation at many magnetic fusion energy research institutions such as DIII-D, where client machines have less RAM, slow, single CPUs, and are on slower networks than the server machines. Under the thin client, expressions are evaluated by the server; the results of these expressions are sent over the network to the client, which does very little computational work. Moreover, machines running the thin client do not need to have complete working versions of MDSplus software, but instead only the bare minimum client libraries needed to connect to the server and read data. This can be helpful in situations where disk space as well as configuration and maintenance time are at a premium.

Figure 2 illustrates the new distributed client approach to MDSplus. Under distributed client, each client process does its own expression evaluation (represented by the large dots), and can read data from multiple servers (small dots). This takes more CPU power from each client and puts less load on each individual server. This is better suited for faster, more capable client machines.

With distributed MDSplus, the expression itself is sent to the client, and the client does its own expression evaluation. This difference in expression evaluation can significantly impact overall performance in a number of ways. First, expression evaluation consumes computational resources. If this load

can be distributed over multiple machines the end user should see better performance.

Second, when the result of the expression is much larger than the expression itself, the transfer of the large expression result is less efficient than the transfer of the short expression. A specific example of this is the node expression which is used to retrieve data from PTDATA. As an expression, this takes only a few bytes. Once evaluated, this takes up anywhere from 8 KB to hundreds of MB. Using thin client, data must go from the PTDATA server, to the MDSplus server, then on to the client. Using distributed client, the data would travel directly from PTDATA to the client, thus avoiding the extra hop.

## 2. Test Infrastructure

We tested distributed MDSplus using Linux workstations already installed in several clusters at General Atomics. In particular we utilized machines configured in a Load Sharing Facility (LSF) cluster. General Atomics makes extensive use of LSF clusters onsite and we hope to take advantage of the existing infrastructure in the test.

For client machines we used 12 dual-processor workstations. These workstations are identical and each have dual 2.66 GHz Intel Xeon processors and 2 GB of RAM in each node. These workstations are part of a large computational cluster.

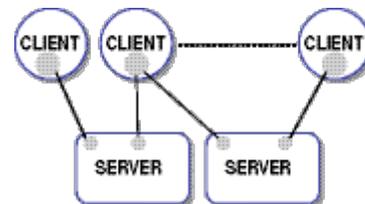


Fig. 2. Clients read data from multiple servers under distributed MDSplus.

We identified three dual-processor Linux workstations to serve as MDSplus data servers. These three machines are identical, and have dual 3.06 GHz Intel Xeon processors. They have 4 GB of RAM. These machines are also part of an LSF cluster. Both the data servers and clients use gigabit Ethernet. All of our tests were done on our LAN, because all of the current MDSplus activity in our organization is local.

All of the machines are running RedHat Enterprise Linux, version 3. They are fully patched

and upgraded. The clients are running MDSplus version 1.5-0. The servers are running 1.6-2.

### 3. Testing Procedures

We tested several database- and system-level configurations to find what bearing they have on the two versions of MDSplus. Each set of tests were run at multiple times during the day and night to achieve as much consistency as possible. We performed five tests which represent common usage of the MDSplus system and two tests which are common system “tweaks” recommended for Linux systems running databases. For this round of tests we only did read tests to the MDSplus database. This represents a normal activity for the DIII-D scientists.

1. Baseline: comparison of thin client and distributed client in a normal environment.
2. Math results: comparison when math is being done within node expressions.
3. Indirection: comparison when there is multi-referencing within a tree.
4. PTDATA: comparison when there is interaction with PTDATA.
5. Moderate Database Loads: comparison of performance with 6 simultaneous reads
6. TCP adjustments: comparison when various TCP parameters are changed
7. RAMDisk: comparison of performance when RAMDisks are used.

Results for all tests are shown in Table 1.

#### 3.1. Baseline

The baseline tests were run during normal production hours, but not during Fusion operations. This was done because it represents a middle ground – average daily use of the machines without large spikes of demand or idle time. Our tests consisted of one-at-a-time reads of floating point data. We used 8 MB data sizes – these are also fairly average in size. In the simple case of a single client reading two million values of compressed floating-point data over a local area network from a numeric node in a tree located on direct-attached storage, there is almost no measurable difference between thin client and distributed client performance.

#### 3.2. Moderate Load (Simultaneous Clients)

However, when six clients simultaneously read data, differences between thin and distributed

MDSplus become apparent. Under this scenario, distributed MDSplus outperforms thin client by approximately 20%. This difference is most likely due to the fact that, since thin client puts all of the work on the server while distributed client keeps the work on the client, distributed client keeps server loads lower relative to thin client, a difference that becomes apparent under moderate loads.

#### 3.3. Math Expressions

The first test involved using math within node expressions. This is a very common procedure for our researchers. Under this condition, the distributed client performed better than the thin client. This is not surprising given that the computation is being handled on all of the different machines, rather than on just the servers. The process becomes in many ways “parallel,” as the computational load is shared among the different machines. The actual TDI expression used was  $\_data = [..], \sin(\_data) + \cos(\_data) + \tan(\_data)$ , where  $[..]$  is the data array of two million floating point values. Distributed performed 23% better than thin client in this test.

#### 3.4. Indirection

In the next test we introduced multi-referencing within the trees. The data node contained not raw data, but a node reference; the subject of this node reference contained another node reference, and so on, leading to nine levels of indirection. Distributed performed 10% better than thin client in this test.

#### 3.5. Compression

Next, we examined the effect of turning off compression on the clients. This produced some very interesting results – when compression was turned off, distributed client performance dropped measurably, while thin client performance stayed the same. The drop off for distributed client can be explained by its heavier reliance on the network – since data is being shipped back and forth between the clients and the servers, the uncompressed data moved at a much slower rate, this affecting the performance of the system as a whole.

#### 3.6. PTDATA

In the special case of the PTDATA proxy – that is, reading data from a numeric node containing a `ptdata2()` TDI function, which points to data in the PTDATA

database to be read — distributed MDSplus outperformed thin client by 16%. This is because under thin client, the MDSplus server first retrieves the data from PTDATA, then sends that data to the client, resulting in an extra network hop as the data passes from the originating database, through the MDSplus server, to the client. Under distributed client, the `ptdata2()` TDI expression is sent to the client, which then directly reads the data from the PTDATA database, thus avoiding the extra network hop.

### 3.7. System Change – TCP parameter changes

One thing that is commonly done to systems running database software is to modify the network settings. This allows the system to have the maximum number of reads and writes occur during a given time. We increased the maximum TCP send and receive buffers and the memory reserved for TCP buffers also.

These settings are all commonly recommended settings for database-serving systems. Under these settings distributed client outperformed thin yet again. Interestingly however, these settings actually caused performance to degrade in both distributed and thin client. We will need to conduct further testing to determine what caused this result.

Table 1

Results of all tests

TEST Results	TPT <sub>READ</sub> (MB/s)	
	THIN	DIST
1-Client	8.5	8.6
6-Client	2.8	3.3
Math	2.0	2.5
Indirect	3.0	3.3
Uncompressed	8.3	4.6
TCP Windows	2.4	3.1

## 4. Discussion of System Testing

Clearly the distributed client came out on top of nearly all of the tests. The only one where it did not best the thin client was when compression was turned off. This would be an unusual thing to do, and would likely only be done if there were some uncommon circumstances which demanded it. The improved performance of the distributed client has important ramifications for future MDSplus deployment at DIII-D: be-

cause the clustered computing model is now prevalent (and is projected to be so for the foreseeable future), it makes sense for us to move away from the thin client, and its reliance on monolithic server architecture. As more and more fast Linux boxes are added to our current infrastructure, the overall performance of MDSplus, using distributed client, should only improve.

## 5. Obstacles, Oddities and Future Directions

For budgetary reasons, we changed the study from an initial focus on MDSplus server performance to client performance, using a number of distributed clients running on clustered workstations. This proved to be a blessing in disguise, as it allowed us to test using a real-world infrastructure and giving us results that we can use for future planning.

Running tests on real-world equipment involves its own set of problems of course. Even though our results are enlightening for our particular environment, it was difficult at times to eliminate extraneous factors. The servers and clients we were using were simultaneously being used by researchers in the course of their daily work. This means that load (cpu and network) on the machines was not always consistent. In order to get around this as much as possible we ran multiple tests for every scenario to filter out the noise as much as possible.

There are several areas of testing that need to be done for the distributed client. The tests for this paper were all read tests. The next set of test will be write tests. When running the DIII-D experiment, there is a large amount of data being written into MDSplus. A similar battery of tests should be done simulating this procedure.

It would also be enlightening to run tests using a WAN. This would be useful to see its implications for collaborators. It also would provide useful information for future, large-scale fusion facilities which might have computing resources spread across multiple locations.

### Acknowledgment

This work supported by the U.S. DOE under DE-FG02-03ER83842 and DE-FC02-04ER54698.

### References

- [1] T.W. Fredian and J.A. Stillerman, MDSplus: Current Developments and Future Directions, *Fusion Engin. Design* **60** (2002) 229.
- [2] B.B. McHarg, Access to DIII-D Data Located in Multiple Files and Multiple Locations, *Proc. 15th Symp. on Fusion Engineering*, 1993, p. 123.