pre-WS MDS and WS-MDS Performance Study
Ioan Raicu
01/05/06

Table of Contents

Table of Figures

Abstract

The focus of this report is the performance of Monitoring and Discovery System (MDS), an essential Globus Toolkit component that is very likely to be used in a mixed LAN and WAN environment. We specifically tested the scalability, performance, and fairness of the WS-MDS Index bundled with GT 3.9.5 [1], and GT 4.0.1 (in this report); we also performed a similar study on the pre-WS MDS Index bundled with GT 4.0.1 which is based on LDAP in order to compare the two implementations. To drive our empirical evaluation of pre-WS and WS MDS, we used DiPerF [3], a DIstributed PERformance testing Framework, whose design was aimed at simplifying and automating service performance evaluation. For the WAN experiments, we used PlanetLab to geographically distribute the WS-MDS clients throughout the world. For the LAN experiments, we used the TeraGrid, specifically the ANL site.

1 Contributing People

People involved in various parts of the experiments presented in this (1) report, in a (2) previous study on WS-MDS performance [1], and in (3) another previous study on WS-MDS [2] (in no particular order):

- Ioan Raicu (1, 2)
- Catalin Dumitrescu (1, 2)
- Neill Miller (1, 3)
- Xuehai Zhang (1)
- Jennifer Schopf (1, 2, 3)
- Ian Foster (1, 2, 3)
- Mike D'Arcy (3)
- Laura Pearlman (3)
- Carl Kesselman (3)

2 Testbeds and Performance Metrics

We used PlanetLab in a previous study of MDS4 WAN performance, and the TeraGrid for the performance in a LAN. In the newer study on MDS2 performance in a LAN, we also used the TG. Both the TG and PL are described in the next subsection.

2.1 PlanetLab and TeraGrid

PlanetLab: PlanetLab [4] is a geographically distributed platform for deploying, evaluating, and accessing planetary-scale network services. PlanetLab is a shared community effort by a large international group of researchers, each of whom gets access to one or more isolated "slices" of PlanetLab's global resources via a concept called distributed virtualization. PlanetLab's deployment is now at over 500 nodes (Linux-based PCs or servers connected to the PlanetLab overlay network) distributed around the world. Almost all nodes in PlanetLab are connected via 10 Mb/s network links (with 100Mb/s on several nodes), have processors speeds exceeding 1.0 GHz IA32 PIII class processor, and at least 512 MB RAM. Due to the large geographic distribution (the entire world) among PlanetLab nodes, network latencies and achieved bandwidth varies greatly from node to node. In order to capture this variation in network performance, Figure 1 displays the network performance between 268 nodes to 1 node at UChicago. It is very interesting to note the heavy dependency between high bandwidth / low latencies and low bandwidth / high latencies.

TeraGrid (TG) is an open scientific discovery infrastructure combining leadership class resources at eight partner sites to create an integrated, persistent computational resource. The deployment of TeraGrid brings over 40 teraflops of computing power and nearly 2 petabytes of rotating storage, and specialized data analysis and visualization resources into production, interconnected at 10-30 gigabits/second via a dedicated national network. The University of Chicago / Argonne National Laboratory (UC/ANL) site has 96 IA32 nodes and 62 IA64 nodes as part of the TG. We used 20 dedicated IA32 nodes to run the client workload and 1 dedicated IA32 node to run the MDS Index. Each node has dual 2.4GHz Xeon processors, 4GB RAM, and SuSE v8.1. The 21 nodes were all connected via 1Gb/s Ethernet network.

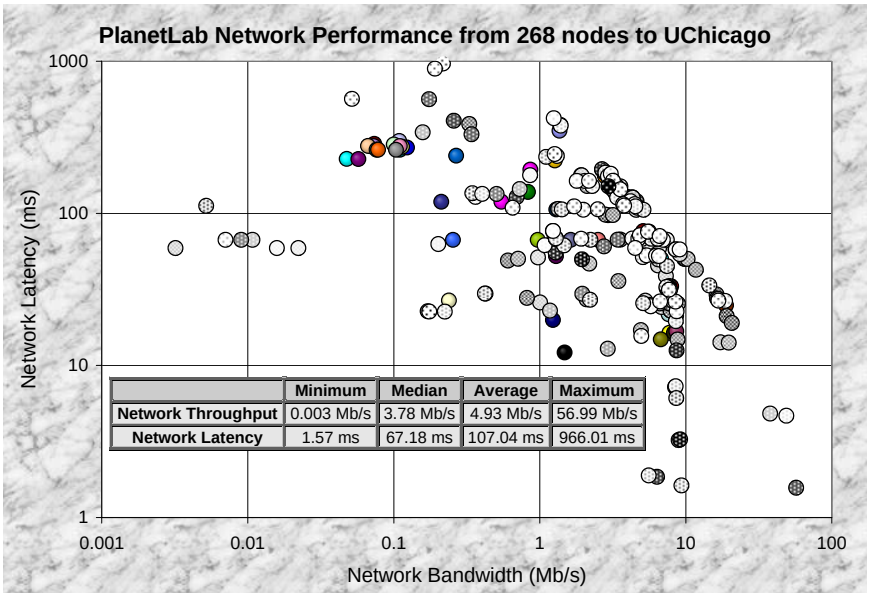

Figure 1: PlanetLab Network Performance from 268 nodes to a node at UChicago as measured by IPERF on April 13th, 2005; each circle denotes a node with the corresponding x-axis and y-axis values as its network characteristics, namely network latency and bandwidth, in log scale.

2.2 Metrics

The metrics collected (client view) by DiPerF which produced the results in this report are:

- ***Service response time*** or time to serve a query request, that is, the time from when a client issues a request to when the request is completed
- ***Service throughput:*** aggregate number of queries per second from the client view
- ***Load:*** number of concurrent service requests

2.3 Experiment Setup

In order to keep the experiments between MDS2 and MDS4 as similar as possible, we designed the MDS2 client code in JAVA, and configured the MDS2 client code to setup a connection, do repeated queries for a predefined amount of time, then close the connection. This inherently tests the performance of LDAP, but then again, MDS2 is heavily based on LDAP, so the LDAP's performance will probably govern the performance of MDS2.

This might be a good debatable issue on if we should maintain a connection for the life of the experiment, or if we should create the connection, perform 1 query, and close the connection (which might be closer to what would happen in a real deployment scenario). However, this would drastically reduce the achieved throughput and increase the response time for both MDS2 and MDS4. For example, in MDS2, the creation of the connection on an unloaded index takes on the order of about 100 ms, which in our experiments gets amortized over 10K+ queries per client; in MDS4, the creation of the connection can take more than 1000ms, which again gets amortized over 1K+ queries.

This is however not what was done in the old papers [5, 6] on MDS2 from 2003 and 2004 since for every query, a separate connection was being created and teared down. As a side effect, the old work reported lower throughput numbers and higher response times than the results we obtained and are presenting in this report.

The MDS2 Index was populated by N number of Information Providers (IP), where N had the value of 1, 10, 25, 50, 100. Each IP had 1 attribute which was set only once at the start of the index to the value returned by the Linux "date" utility (i.e. Fri Dec 16 13:28:59 CST 2005). We turned off caching, and set a maximum time to complete a query to 75 seconds (giving ample time to complete any given query even with 800 concurrent clients). In MDS4, we make N registrations (i.e. adding "date" entries to the Index), which is essentially the same as the N different IPs each having 1 attribute, "date".

2.4 Graph Details

Figure 2, Figure 3, Figure 6, and Figure 7 all depict the overview of the MDS2 and MDS4 performance using discrete sustained loads. Each discrete load was sustained for 10 minutes; the first two minutes of each sustained load was disregarded since the performance was more variable due many clients starting up; therefore, each point on the graph represents the average of 8 minutes worth of performance metrics, and the y-error bars represent the standard deviation in the performance metrics over the corresponding period of 8 minutes. The different index sizes we measured the performance of MDS were: 0, 10, 25, 50, 100. The discrete sustained loads we choose were: 1, 2, 3, 4, 5, 6, 7, 8, 16, 32, 64, 128, 256, 384, 512, 640, 768, 800. We stopped at 800 concurrent clients due to Linux configured limit of 1024 open file descriptors (FDs) per process (which essentially limited the GT4 container to 1024 open FDs). In order to raise this limit, we would have to have root privileges, which we did not have in the TG.

Figure 10, Figure 11, Figure 12, Figure 13, Figure 14, Figure 15, and Figure 16 all depict the performance of MDS2, showing all the raw data collected by DiPerF; these graphs are important in order to better understand Figure 7.

3 MDS4 (WS MDS) Results

The MDS4 (based on GT 4.0.1) experiments presented in this section are very similar to those done in September 2005. I wanted to get the most up-to-date results from the MDS4 Index, so I re-ran all the experiments on the MDS4 Index based on the latest GT 4.0.1 build from CVS (as of last week).

The throughput obtained by MDS4 in the TG (LAN) was quite impressive, very robust, and consistent. Notice the low error bars, and the flat throughput achieved once the service was saturated, despite an increasing number of concurrent clients. The results show that the MDS4 Index service can easily handle 100+ of concurrent clients without loss of efficiency.

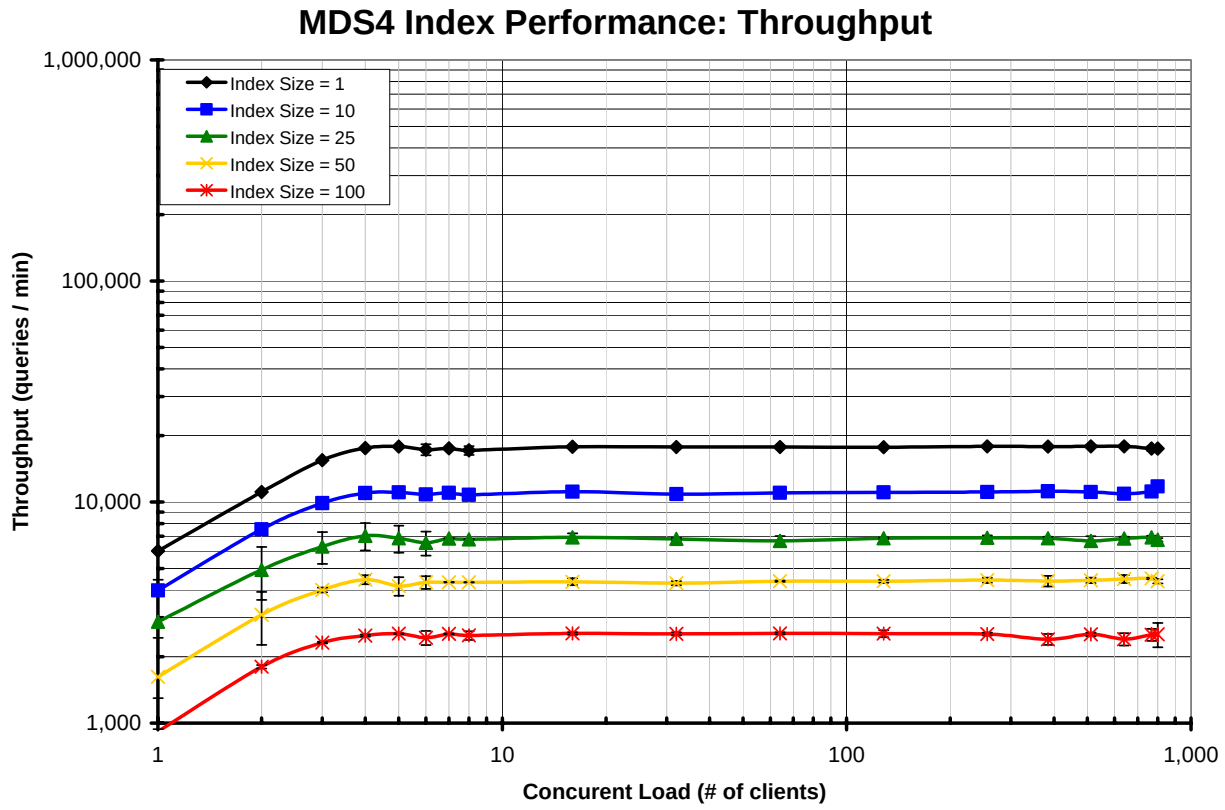

Figure 2: MDS4 Index Performance: Throughput; y-axis – throughput (queries/min) [log scale]; x-axis – concurrent load (# of clients) [log scale]

The response time performance was equally as impressive, with excellent stability and consistency. Once the service was saturated, the response time grew linearly with increasing number of concurrent clients.

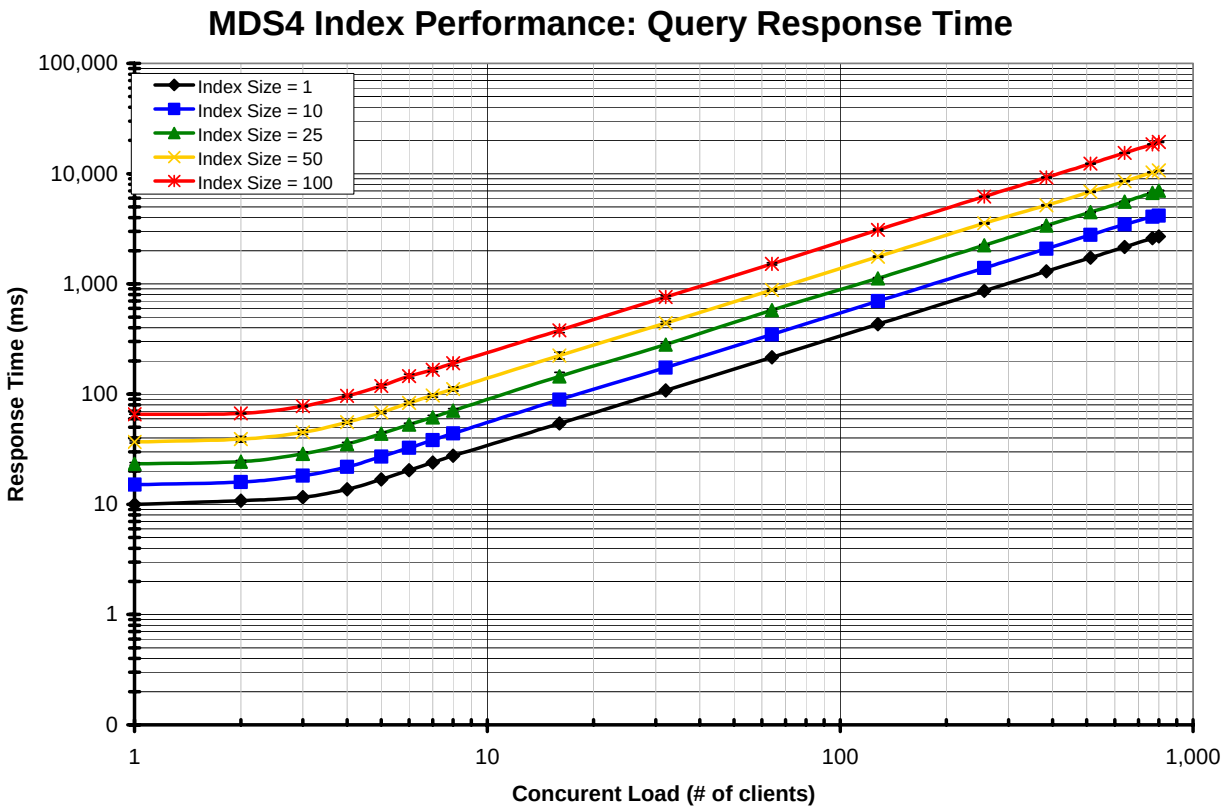

Figure 3: MDS4 Index Performance: Query Response Time; y-axis – response time (ms) [log scale]; x-axis – concurrent load (# of clients) [log scale]

Although from Figure 2 and Figure 3, there is not much visible difference from the old experiments, but when comparing the old results with the new ones, we saw that there was a 3~4% overall improvement in performance on the new code. Figure 4 and Figure 5 shows the speed-up of the new code from the old one. It seems that the performance speed-up varied from 0.9 (10% slower) to 1.17 (17% faster), with some experiments being faster, while others where a little slower. Overall, the response time was improved by about 4% while the throughput was improved about 3%.

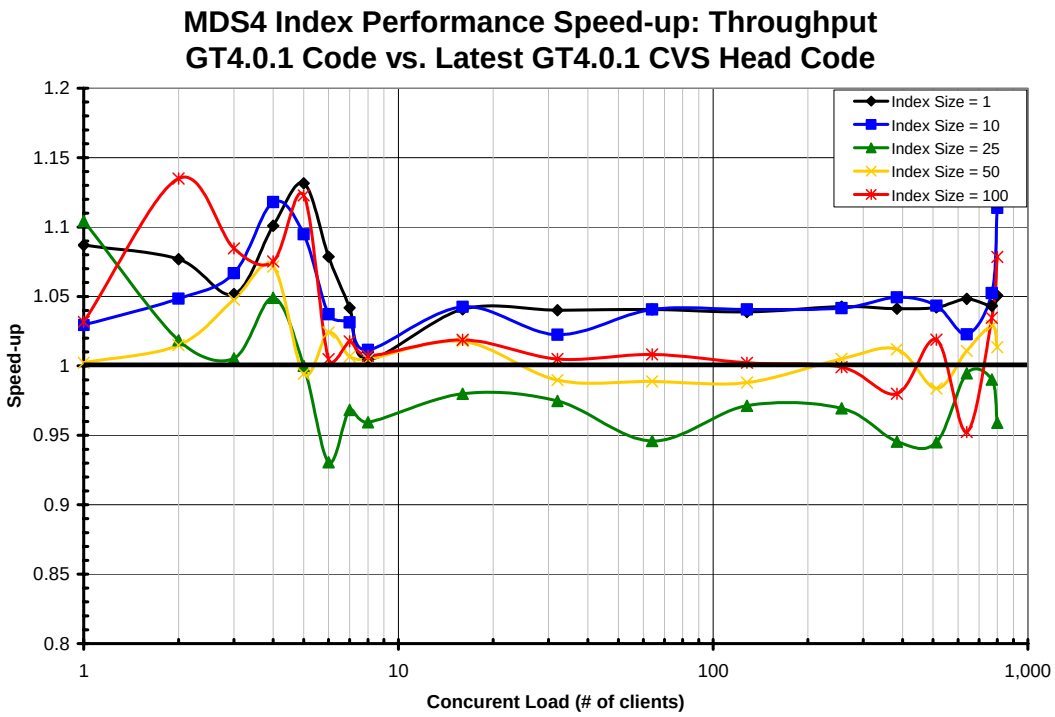

Figure 4: MDS4 (latest CVS Head code) Throughput Speedup over MDS4 (original GT 4.0.1) Performance; y-axis – Performance Speed-up (MDS4 [latest code] is Faster for values > 1 and MDS4 [original] is faster for values <1); x-axis – concurrent load (# of clients) [log scale]

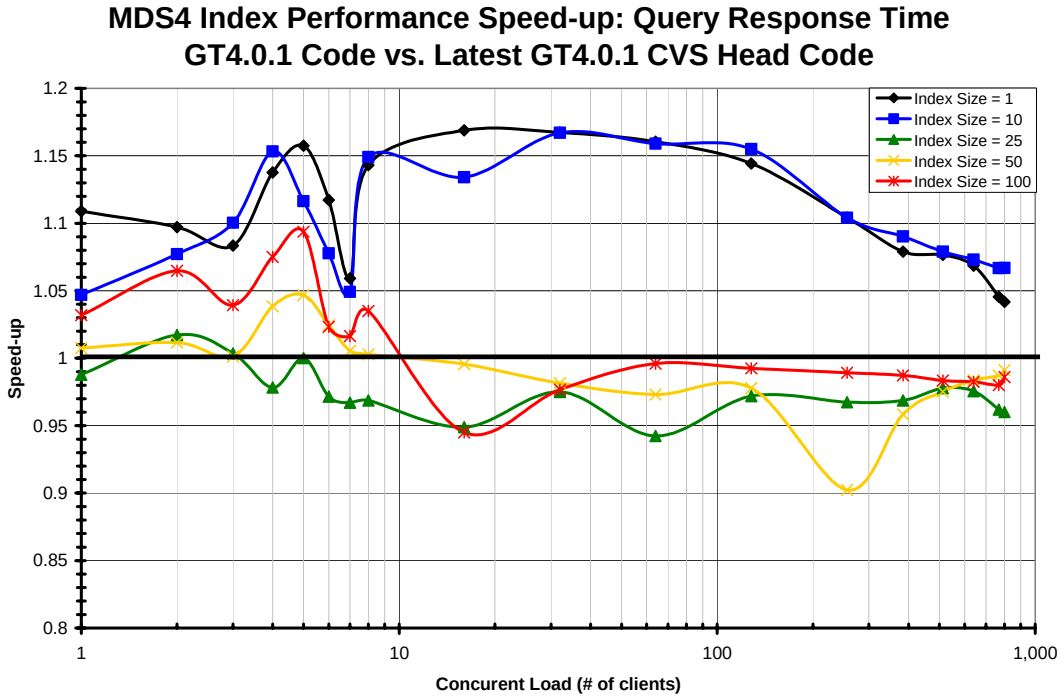

Figure 5: MDS4 (latest CVS Head code) Response Time Speedup over MDS4 (original GT 4.0.1) Performance; y-axis – Performance Speed-up (MDS4 [latest code] is Faster for values > 1 and MDS4 [original] is faster for values <1); x-axis – concurrent load (# of clients) [log scale]

4 MDS2 (pre-WS MDS) Results

We measured the throughput and response time for pre-WS MDS bundled with GT 4.0.1 on the same testbed we performed the similar study of MDS4 performance. The summary section shows the two graphs (throughput - Figure 6 and response time - Figure 7) that can be directly compared to the MDS4 results from Figure 2 and Figure 3. We follow up with a more in-depth analysis (Figure 10 through Figure 16) of the response time performance of MDS2 (depicted in Figure 7). We finally measure the speed-up (and sometimes slow-downs) of MDS2 over MDS4 and show the results in Figure 17 and Figure 18, along with our conclusions on MDS performance.

4.1 Summary of MDS2 Results

We measured the performance of MDS2 with and without caching enabled. The first section covers the performance when caching was enabled, in which we see a drastic performance increase over the performance of MDS4. The second section covers the partial results which we were able to obtain for MDS2 without caching. We were only able to test the performance for an index size of 1, while for greater index sizes, the experiments failed.

4.1.1 MDS2 with Caching

The throughput achieved by the MDS2 Index was considerably higher across the board when compared to that of MDS4. In general, we saw speed-ups in the range of 5 to 30 with MDS2 being faster than MDS4, but in a few instances, the MDS2 performance, specifically the response time, was magnitudes slower than that of MDS4.

We can see that MDS2 very quickly (between 1 to 3 concurrent clients) achieves its peak throughput of nearly 300K queries per minute when the index size was 1 and almost 13K with an index size of 100. In contrast, the MDS4 performance peaked at 17K queries per minute with an index size of 1, and 1.5K with an index size of 100. The consistency of the MDS2 Index in terms of achieved throughput seems to be just as good as MDS4's consistency.

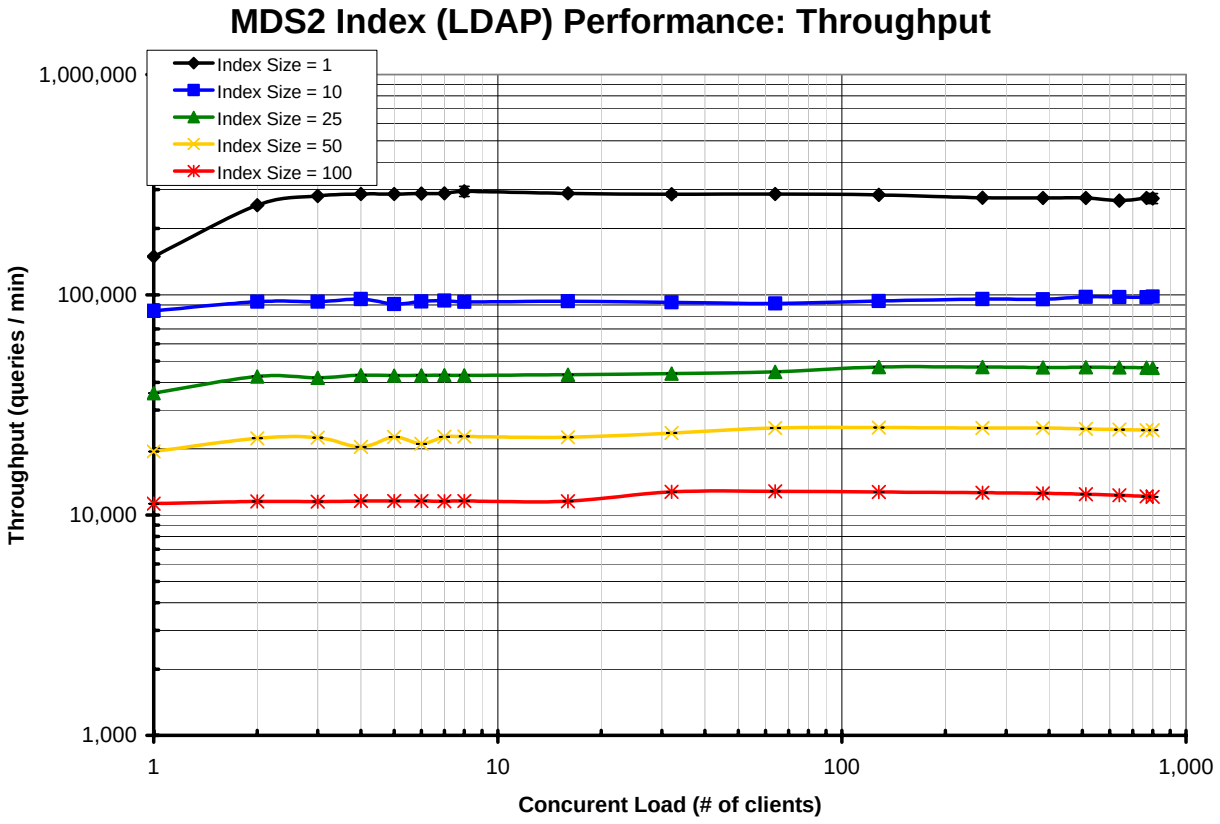

Figure 6: MDS2 Index Performance with caching: Throughput; y-axis – throughput (queries/min); x-axis – concurrent load (# of clients) [log scale]

If everything up to this point was predictable, consistent, and robust, the next experiment shows some serious flaw in MDS2 (really LDAP) internal management of its resources. Although it seems from Figure 7 that only larger index sizes have this problem, in fact the problem exists regardless of the index size; it manifests itself in different intensities, with larger index sizes showing worse behavior, as we will see in Figure 10 through Figure 16.

In the experiments we performed ranging from 1 to 800 clients and with index sizes between 1 and 100, the problem areas seem to be very bad between 2 to 16 concurrent clients, and the problems linger up to about 128 concurrent clients; the oddity of the entire experiment is that as the number of concurrent clients keeps rising above 128 concurrent clients, the behavior of MDS2 Index becomes more normal, or at least closer to the MDS4 experiments which had stable and consistent results. I re-ran these experiments a second time just to make sure it was not a fluke, but unfortunately I found almost identical results.

Perhaps the NetLoger analysis can shed some light on this, exactly where it is spending so much time. For example, with an index of size 100, and 16 concurrent clients, it takes more than 10 seconds to answer a single query, but with 32 concurrent clients, it takes less than 0.2 seconds. Looking at the raw data gives us some idea about what is happening, but not why it is happening. I believe that there are a few clients that are literally starving (very long response times, while the majority of the clients had good response times). I will discuss this in more depth in a following subsection (4.3).

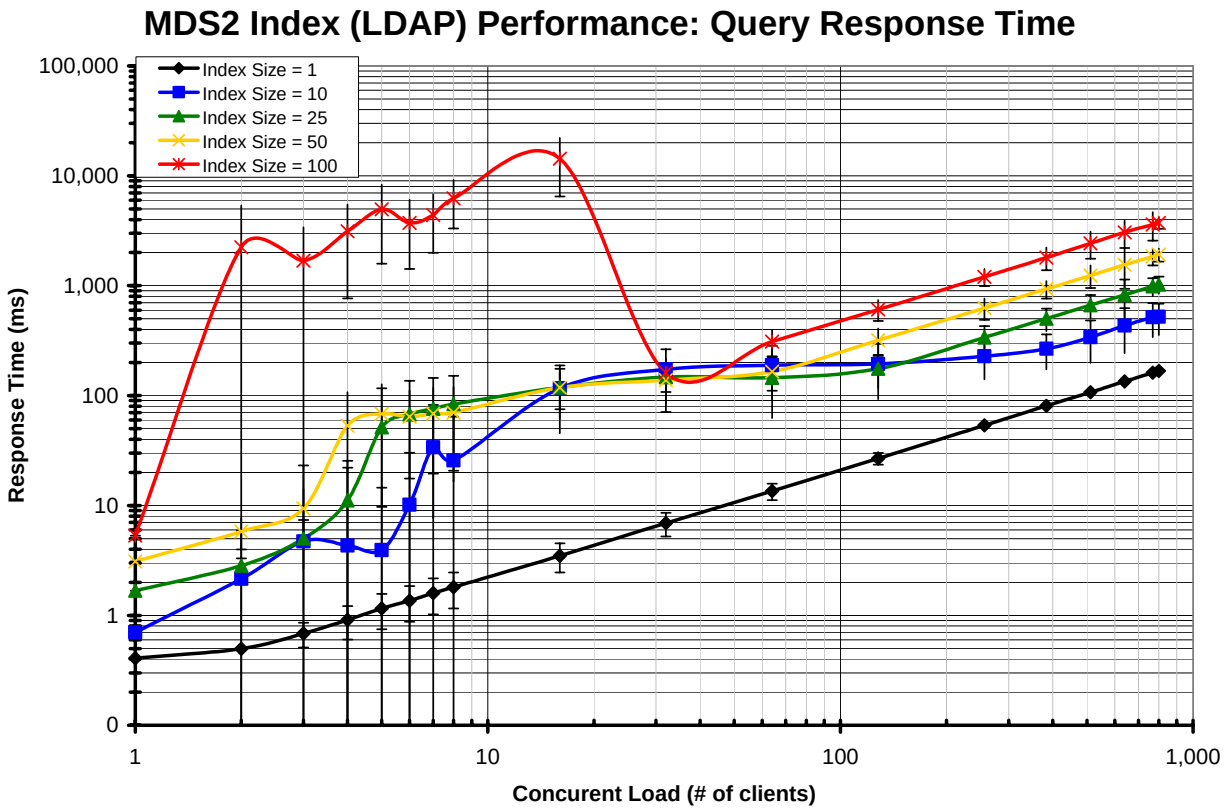

Figure 7: MDS2 Index Performance with caching: Query Response Time; y-axis – response time (ms) [log scale]; x-axis – concurrent load (# of clients) [log scale]

4.1.2 MDS2 without Caching

According to our limited results from Figure 8 and Figure 9, we see a tremendous slowdown in MDS2 without caching when compared to MDS2 with caching. MDS2 without caching achieves performance (both throughput and response time) of about 1% when compared to the performance of MDS2 with caching on an index size of 1. We were not able to test the performance of MDS2 without caching with index sizes of 10, 25, 50, and 100 due to failing experiments. It seemed that similar concurrency issues as those experienced in MDS2 with caching (Figure

7) manifested themselves even worse as only 1 concurrent client ever ran over the duration of each experiment which lasted over 3 hours. We found that by inserting some wait time between each successive MDS2 query helped alleviate the problem somewhat, but as the MDS2 Index became saturated, it would start ignoring new queries. Since we were not able to create an experiment that would saturate the MDS2 Index service (similar to the rest of the experiments) for index sizes of 10, 25, 50, and 100, we did not report those results; although, we did report the performance of MDS2 Index without caching for a single concurrent client accessing the MDS2 Index.

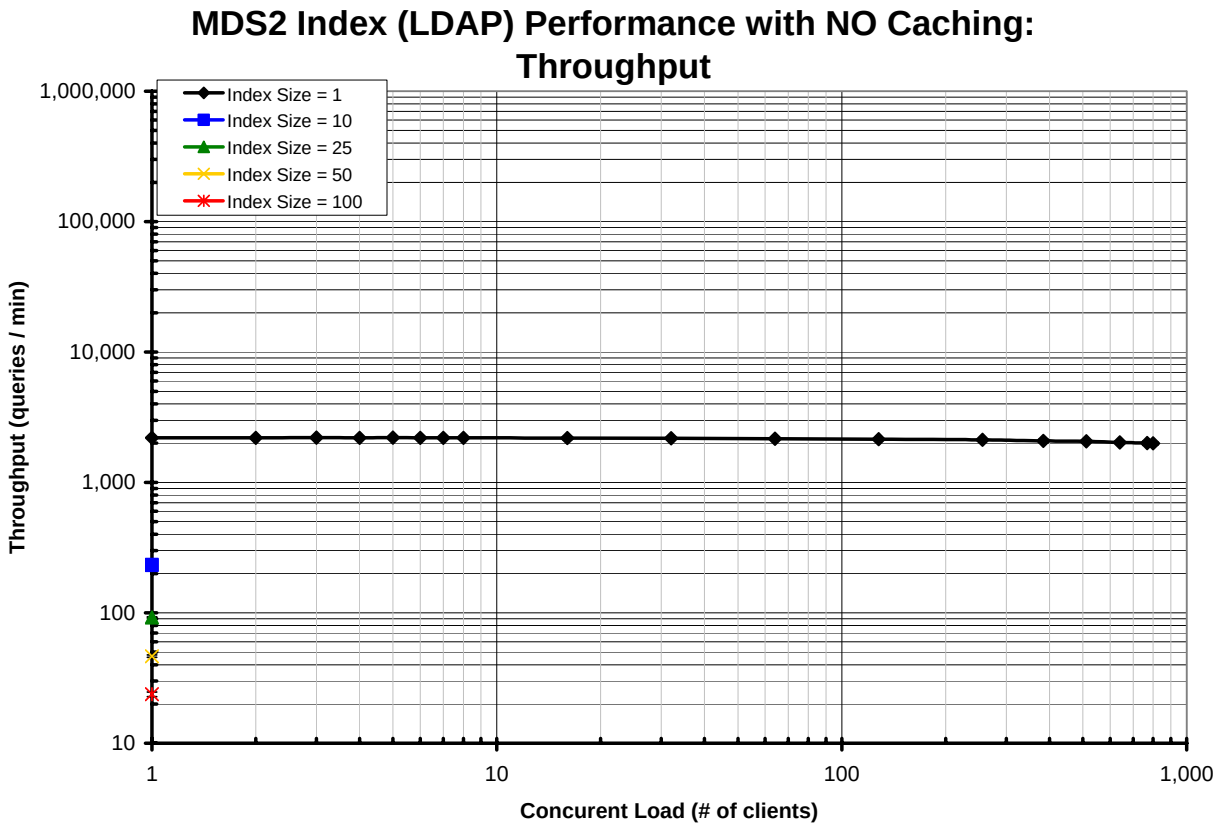

Figure 8: MDS2 Index Performance without caching: Throughput; y-axis – throughput (queries/min); x-axis – concurrent load (# of clients) [log scale]

It is interesting to see that the throughput decreases about 10% as the number of concurrent clients increases from 1 to 800. Also, pay attention to the x-axis as the log scale ranges from 10 to 1,000,000 (while all the rest of the throughput graphs range from 1,000 to 1,000,000). Overall, we saw an enormous drop in performance when the caching was disabled; furthermore, although MDS2 with caching outperforms MDS4 by a very big margin, MDS4 outperforms MDS2 without caching significantly as well.

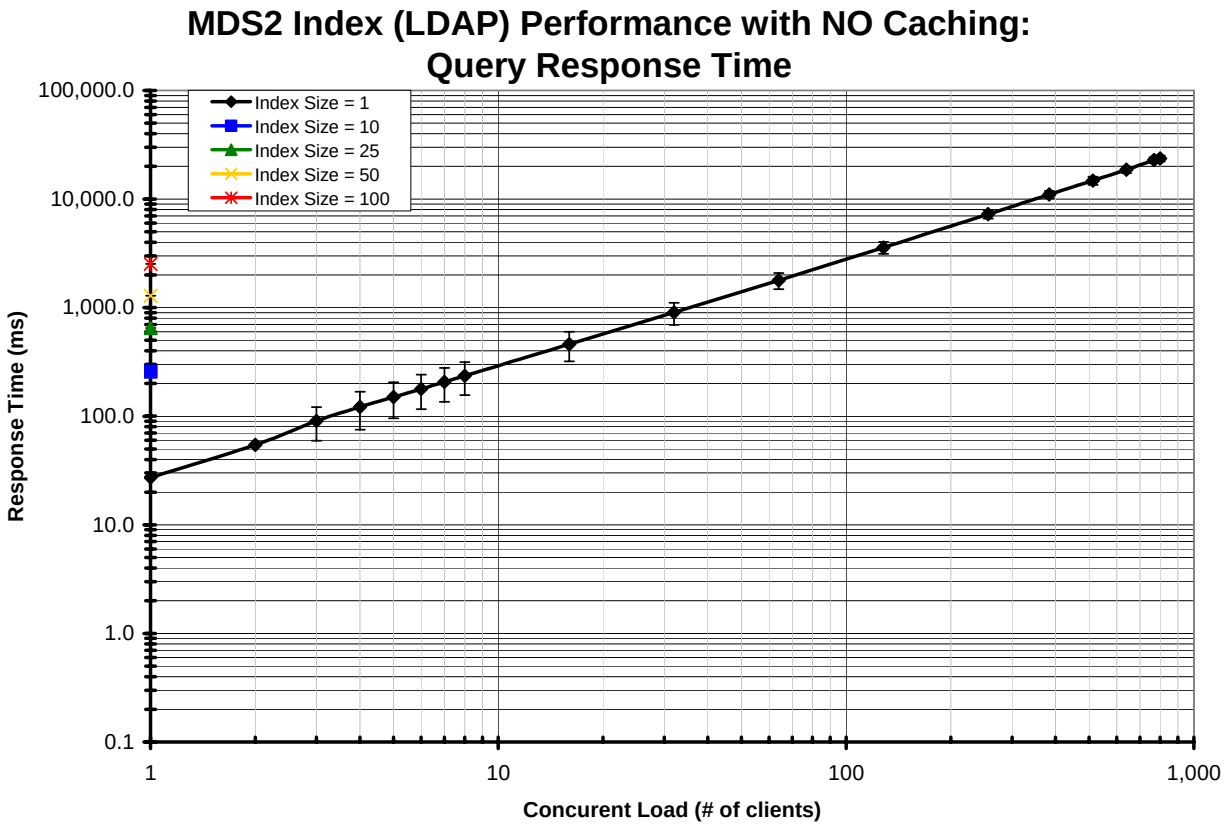

Figure 9: MDS2 Index Performance without caching: Query Response Time; y-axis – response time (ms) [log scale]; x-axis – concurrent load (# of clients) [log scale]

4.2 Further Analysis of MDS2 with Caching Results

This section depicts several graphs showing the raw performance (every point being a 60 second average for 1 client) before it was massaged into aggregate response times (i.e. Figure 7 where every point was a 480 second average for all the concurrent active clients). The kind of details that become evident in looking at the raw performance numbers is the distribution of response times observed for any given load. Figure 7 showed quite high standard deviation for the aggregated response times, but it did not offer us information about the distribution.

For example, Figure 10 shows the same information as Figure 7, but with experiment progression time on the x-axis (rather than load). Also, every point is a raw point (as defined above) rather than being aggregated over all the concurrent clients; note that each index size experiment is denoted by a particular color. The period that seemed abnormal from Figure 7 corresponds to Figure 10's x-axis time of 600 – 5400 sec. In this time period, we see that the response time performance for any given index size varied greatly, sometimes even 3 orders of magnitude at any given time.

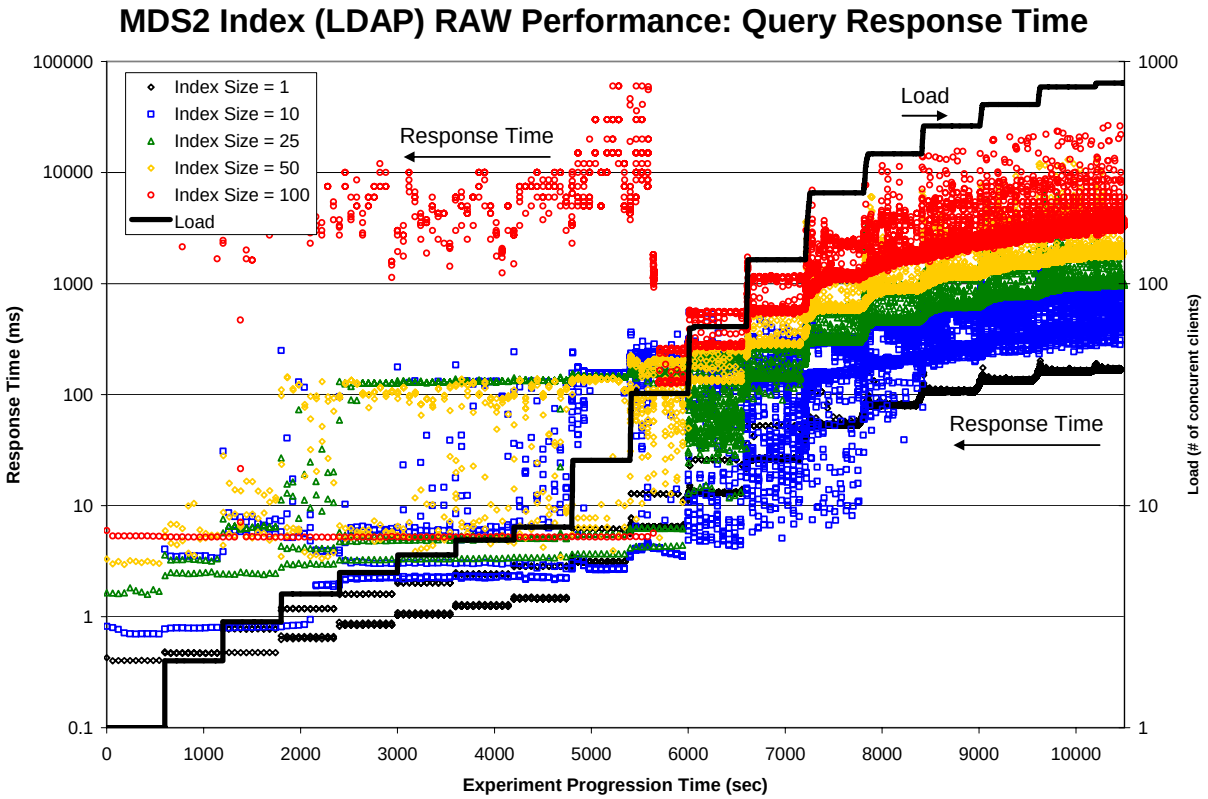

Figure 10: MDS2 Index (size = 1, 10, 25, 50, 100) Performance: RAW (per client with 60 second averages, from 1 – 800 concurrent clients) Query Response Time as a function of time; y-axis (left) – response time (ms) [log scale]; y-axis (right) – load (# of concurrent clients) [log scale]; x-axis – experiment progression time (seconds)

Figure 10 is quite complex, and hard to interpret because of so much information (nearly 250K individual performance samples); we therefore broke up the graph into 5 separate graphs, one for each index size in Figure 11 – Figure 15 for better interpretation.

Figure 11 shows the raw performance of the MDS2 index with a size of 1. Although this experiment looked relatively normal in Figure 7, we can see below that the MDS2 Index (LDAP) really had some resource management issues once more than 1 concurrent client started making queries. Notice the bands of different performance numbers (between from about 1200 sec to about 7500 sec). The log scale might be deceiving in terms of how big the gap is between these bands, but when examining the values closely, we notice that the slow bands have twice the response times as the fast bands. Apparently, this was not enough of a difference to make this visible in Figure 7 outside of some increase in the standard deviation. We will see in the next few figures how the increased index sizes (10, 25, 50, and 100) each widens the gap between these bands, to the point where the average response times between these bands increases drastically, producing the anomaly depicted in Figure 7 at the aggregate level.

If an index size of 1 did not shows signs of the anomaly in Figure 7, an index of size 10 already began to show some signs. Figure 12 shows the raw performance of the index with size of 10. Notice that there are now not just 2 bands, but multiple and not quite as nicely defined as before, with the extremes being even 2 orders of magnitude performance difference between the bands.

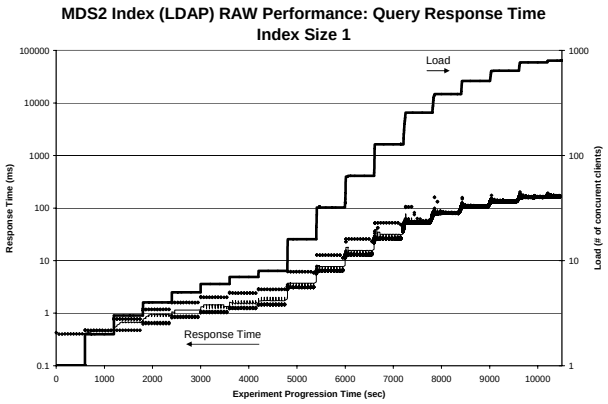

Figure 11: MDS2 Index (size = 1) Performance: RAW (per client with 60 second averages, from 1 – 800 concurrent clients) Query Response Time as a function of time; y-axis (left) – response time (ms) [log scale]; y-axis (right) – load (# of concurrent clients) [log scale]; x-axis – experiment progression time (seconds)

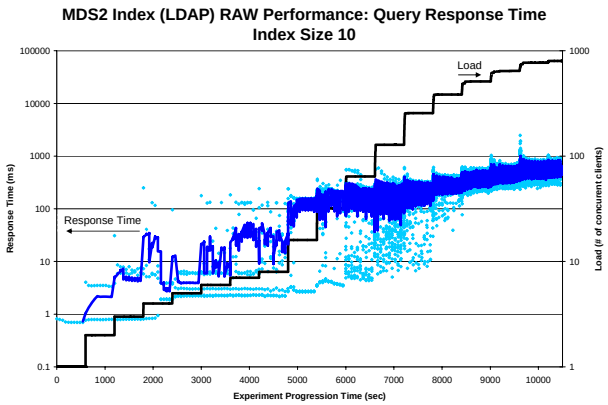

Figure 12: MDS2 Index (size = 10) Performance: RAW (per client with 60 second averages, from 1 – 800 concurrent clients) Query Response Time as a function of time; y-axis (left) – response time (ms) [log scale]; y-axis (right) – load (# of concurrent clients) [log scale]; x-axis – experiment progression time (seconds)

Figure 13 and Figure 14 shows the raw performance of the index with size of 25 and 50 respectively. Notice that the banding persisted, with the extremes being almost 2 orders of magnitude performance difference between the bands.

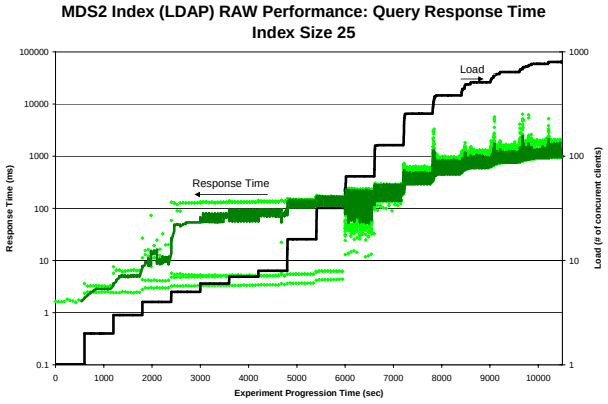

Figure 13: MDS2 Index (size = 25) Performance: RAW (per client with 60 second averages, from 1 – 800 concurrent clients) Query Response Time as a function of time; y-axis (left) – response time (ms) [log scale]; y-axis (right) – load (# of concurrent clients) [log scale]; x-axis – experiment progression time (seconds)

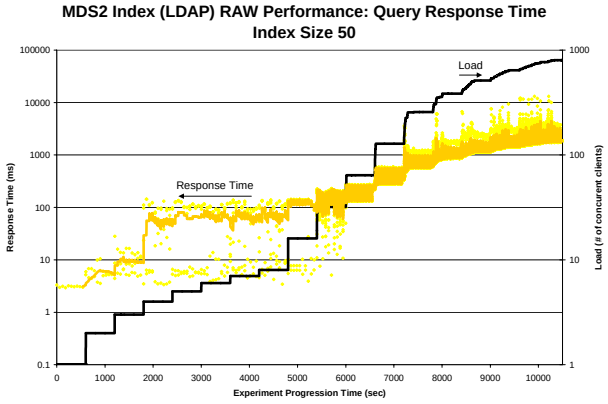

Figure 14: MDS2 Index (size = 50) Performance: RAW (per client with 60 second averages, from 1 – 800 concurrent clients) Query Response Time as a function of time; y-axis (left) – response time (ms) [log scale]; y-axis (right) – load (# of concurrent clients) [log scale]; x-axis – experiment progression time (seconds)

Finally, the MDS2 performance with an index size of 100 seemed to be the most negatively affected by this banding; we believe that as we would increase the size of the index, this anomaly would become worse and worse. Notice that in Figure 15 there are basically 2 well defined bands between 600 seconds and 5500 seconds. The faster band seemed to remain very consistent at about 5 ms regardless of the number of concurrent clients (in the period in question) while the second band slowly increased from 2000 ms to 60000 ms while the number of concurrent clients was raised from 2 to 16. This equates to 3 to 4 orders of magnitude difference in performance for the same number of concurrent clients.

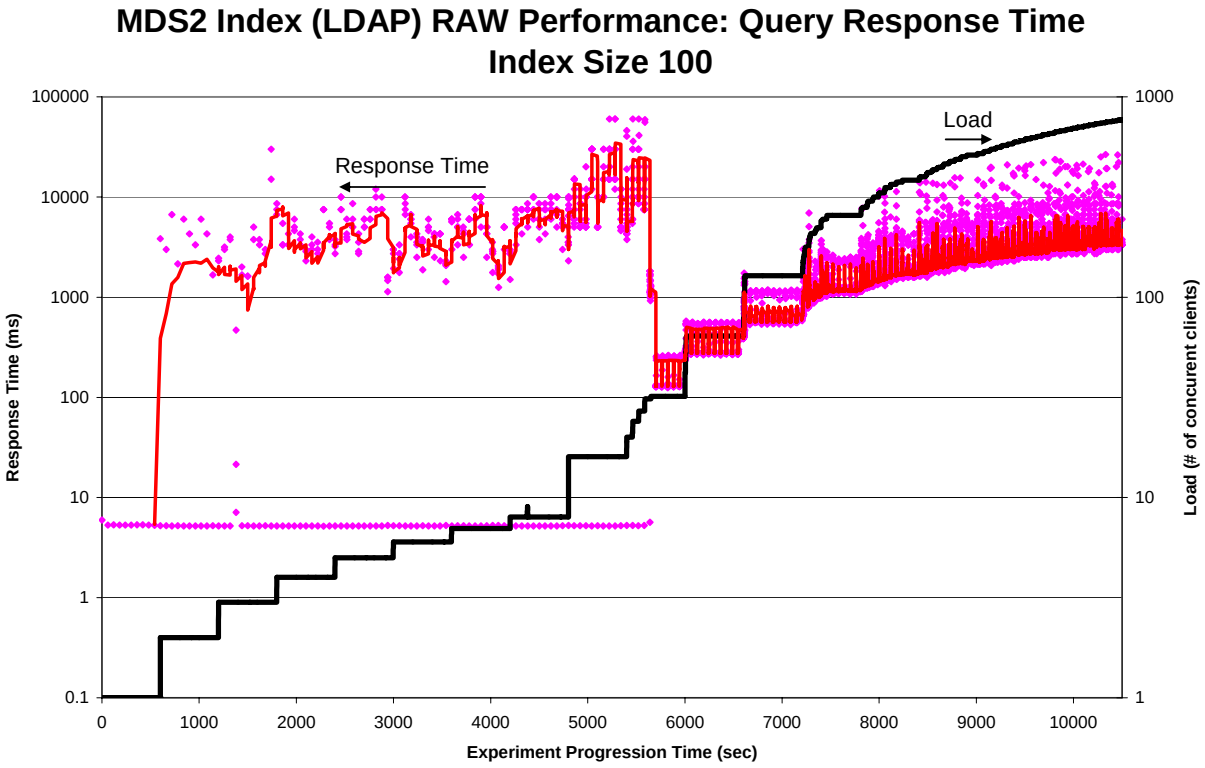

Figure 15: MDS2 Index (size = 100) Performance: RAW (per client with 60 second averages, from 1 – 800 concurrent clients) Query Response Time as a function of time; y-axis (left) – response time (ms) [log scale]; y-axis (right) – load (# of concurrent clients) [log scale]; x-axis – experiment progression time (seconds)

The one thing none of the previous graphs (Figure 10 – Figure 15) showed was the performance per client, and how it changed as new clients started up and began querying the MDS2 index. Figure 16 attempts to capture this exact information, by only looking at the 0 – 5400 seconds from Figure 15, and color coding each client to easily track the performance per client; in this time period, we had 16 concurrent clients eventually, which translates into 16 different color used. The really interesting thing is the fact that when the 1st client starts up by itself, it gets about 5 ms per query; when the 2nd client starts up, the 2nd client gets response times in the 2000 to 7000 ms range, while the 1st client continues to enjoy consistent 5 ms queries. Once the 3rd client starts up, the 3rd client gets poor performance in the range of 2500 ms to 15000 ms; the really interesting part is that the 2nd client goes from very slow response times to blazing 5 ms queries, but the 1st client goes from very fast queries to performance similar to that of the 3rd client. For the rest of this graph, the 2nd client enjoys 5 ms queries, while each new client that joins the experiment gets very poor performance in the range of 1000 ms up to 60000 ms. For the duration of 5400 seconds, it seem that 1 client dominated the MDS2 index resources yielding about 5 ms per query (generating 1,020,920 queries), while the rest of the 15 clients generated a mere 3977 queries. This is a clear case of resource starvation due to the underlying mechanisms of distributing the available resources. One would expect that this resource starvation to get even worse with more and more concurrent clients, but as it is shown above in Figure 15, once there were 32 concurrent clients, the fastest band (with 5 ms query response times) disappeared, and the gap between the existing gaps became smaller (only twice the values, rather than entire magnitudes), and hence it almost appears that the response time behaves as we would have expected it to perform in the first place.

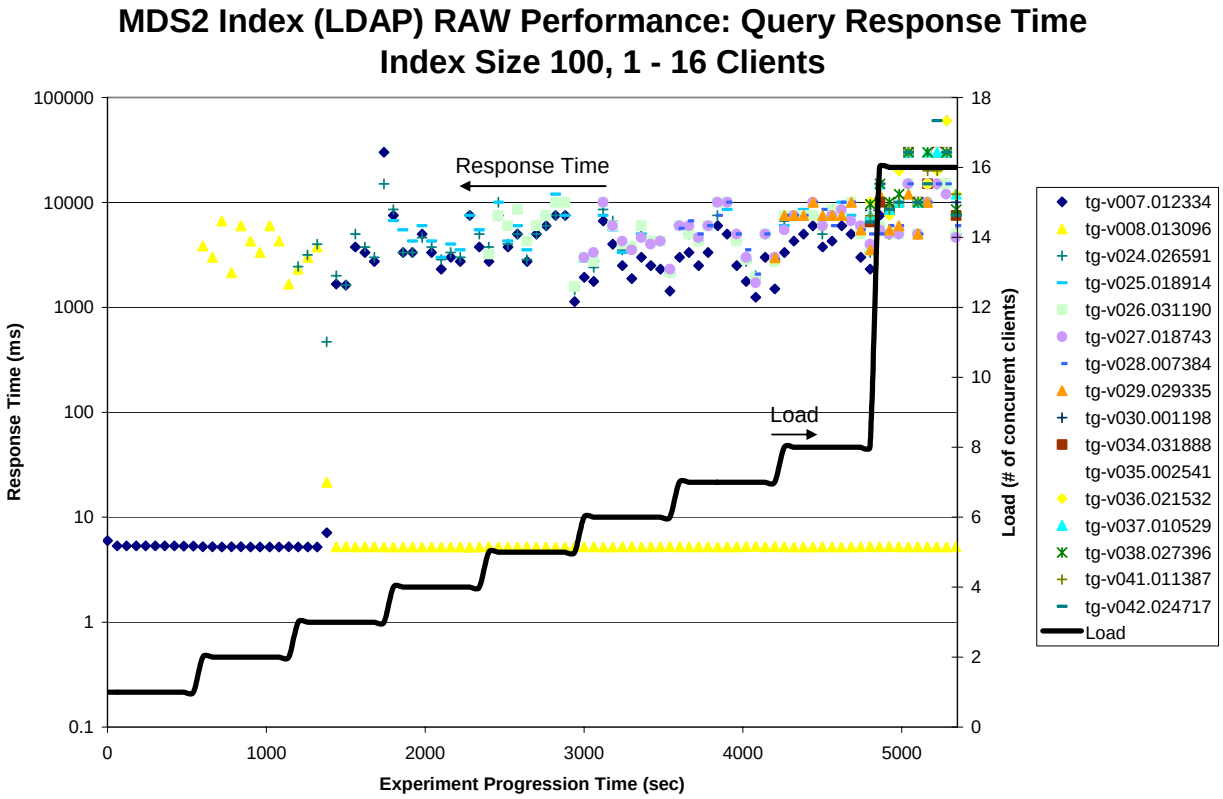

Figure 16: MDS2 Index (size = 100) Performance: RAW (per client with 60 second averages, from 1 – 16 concurrent clients) Query Response Time as a function of time; y-axis (left) – response time (ms) [log scale]; y-axis (right) – load (# of concurrent clients) [log scale]; x-axis – experiment progression time (seconds)

5 Summary and Conclusions

In order to clearly visualize the performance difference between MDS2 and MDS4, we computed the speed-up that MDS2 has over MDS4.

A speed-up is defined as follows:

- Speed-up = 1: MDS2 and MDS4 perform the same
- Speed-up > 1: MDS2 performs better than MDS4
- Speed-up < 1: MDS2 performs worse than MDS4

Although the speed-up is well over 10 and up to 30 for small number of concurrent clients, once both MDS2 and MDS4 Indexes were saturated with concurrent clients, the speed-up was more between 4 and 17. Speed-up improvements on the order of 10 times for MDS2, or better yet speed-slowdown of 10 times for MDS4 is definitely significant. Unfortunately, the two implementations of MDS are based on two very different technologies, with MDS2 being based on LDAP, while MDS4 is based on the WS Core. Furthermore, the MDS2 implementation is in C, while the WS Core is implemented in JAVA. Perhaps, in future releases of the Globus Toolkit, MDS4 will have the option of utilizing the WS Core C implementation, which might recuperate some of the lost performance to MDS2.

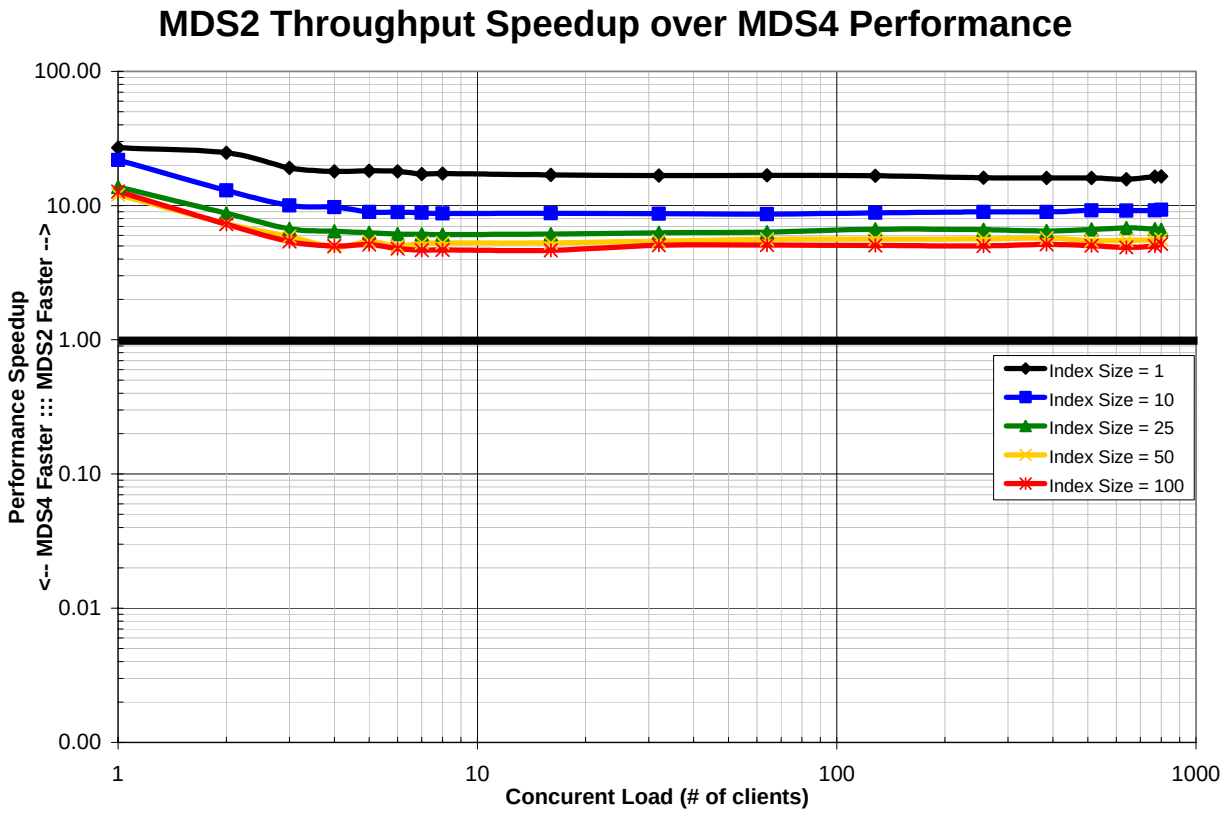

Figure 17: MDS2 Throughput Speedup over MDS4 Performance; y-axis – Performance Speed-up (MDS2 is Faster for values > 1 and MDS4 is faster for values <1) [log scale]; x-axis – concurrent load (# of clients) [log scale]

In a previous section, we witnessed the performance of the MDS2 response time fluctuate significantly. Figure 18 shows how the fluctuation translated into giving MDS4 positive speed-up over MDS2 for 2 to 16 concurrent clients and for large index sizes. Outside the range that seriously affect the MDS2 performance (i.e. more than 128 concurrent clients), the MDS2 response time speed-up over MDS4 seems to be comparable to that of the throughput speed-up observed in Figure 17.

I re-ran the MDS2 experiments testing to see if the results were repeatable, and unfortunately I got essentially identical results. Perhaps the NetLoger analysis with different number of concurrent clients (i.e. 1 & 800 concurrent clients [normal behavior], and 2 & 16 concurrent clients [abnormal behavior]) can give us some insight in terms of where the abnormal client is spending so much time during the periods in question. Looking at the raw data gives us some idea about what is happening, but not why it is happening. I believe that there are a few clients that are literally starving (very long response times, while the majority of the clients had good response times). The response time performance is averaged over a given time period (in our case, over 60 second intervals per client); a few outliers in terms of response times can adversely affect the average response time reported, and hence we get the abnormally high response times. If we would have used the median (instead of the average), it would have been very likely (as long as there were more normal cases than abnormal) that we might not even have discovered this abnormal behavior.

The overall conclusion about the performance difference between MDS2 and MDS4 is that they are significantly different, with the MDS2 performance generally being 1 order of magnitude faster. However, MDS2 did seem to have some concurrency issues, and hence MDS4 seemed to offer the most reliable and robust indexing service, with the potential of getting a big boost in performance once it is offered with the WS Core C version.

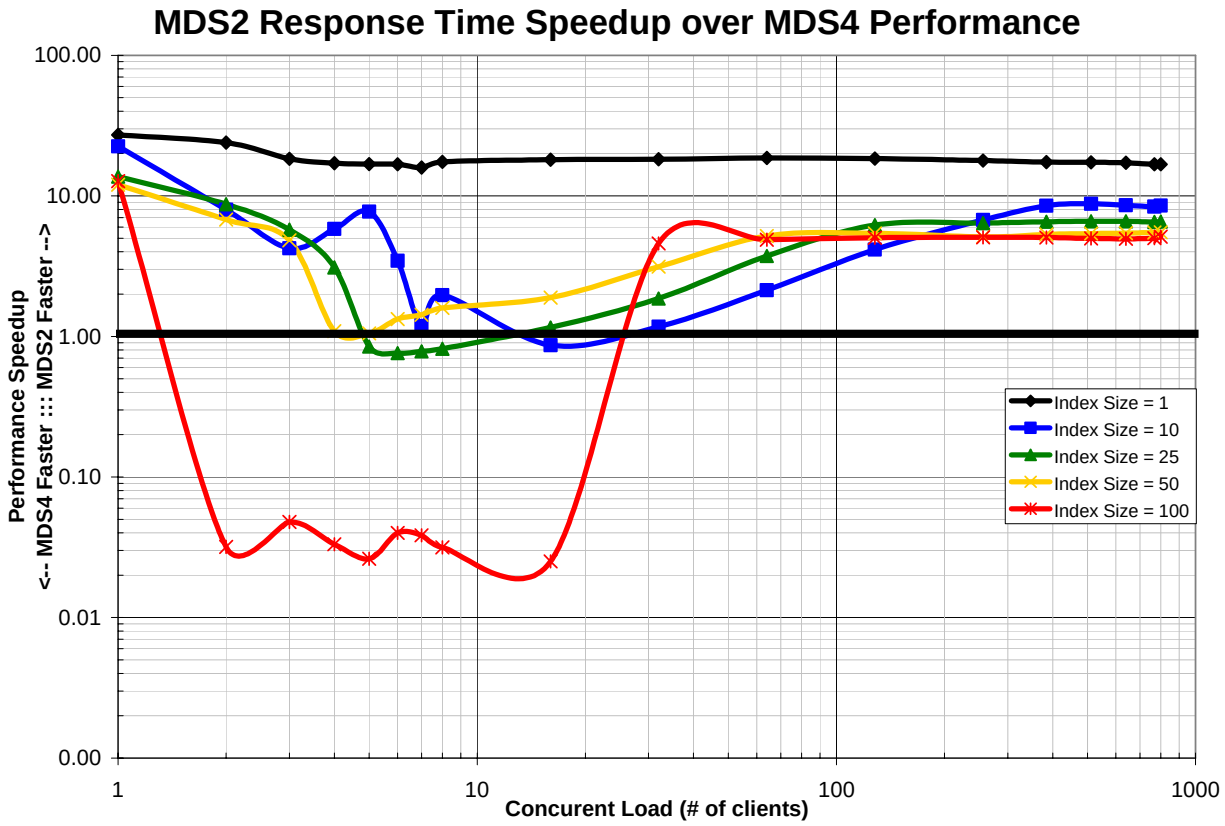

Figure 18: MDS2 Response Time Speedup over MDS4 Performance; y-axis – Performance Speed-up (MDS2 is Faster for values > 1 and MDS4 is faster for values <1) [log scale]; x-axis – concurrent load (# of clients) [log scale]

6 References

[1] Ioan Raicu, Catalin Dumitrescu, Ian Foster. "A Performance Evaluation of WS-MDS in the Globus Toolkit", http://people.cs.uchicago.edu/~iraicu/research/docs/MDS_GRID2005_v02.web.pdf

[2] Jennifer M. Schopf, Mike D'Arcy, Neill Miller, Laura Pearlman, Ian Foster, Carl Kesselman. "Monitoring and Discovery in a Web Services Framework: Functionality and Performance of the Globus Toolkit's MDS4", http://www-unix.mcs.anl.gov/~schopf/Pubs/mds4.sc.pdf

[3] Ioan Raicu. "A Performance Study of the Globus Toolkit® and Grid Services via DiPerF, an automated DIstributed PERformance testing Framework", http://people.cs.uchicago.edu/~iraicu/research/publications/UChicago_MS_thesis_2005_v19.pdf

[4] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay Testbed for Broad-Coverage Services," ACM Computer Communications Review, July 2003.

[5] Xuehai Zhang, Jeffrey Freschl, and Jennifer M. Schopf. "A Performance Study of Monitoring and Information Services for Distributed Systems", Proceedings of HPDC-12, June 2003.

[6] Xuehai Zhang and Jennifer M. Schopf. "Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2", Proceedings of the International Workshop on Middleware Performance (MP 2004), April 2004.