Important Information

**LustrePerfMon**

User Manual­­­­

**Version 0.0.10 | ref.no tbd | Revision A0**



Information in this document is subject to change without notice and does not represent a commitment on the part of DataDirect Networks, Inc. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser’s personal use without the written permission of DataDirect Networks, Inc.

**© 2018 DataDirect Networks, Inc. All rights reserved.**

DataDirect Networks, the DataDirect Networks logo, DDN, DirectMon, Enterprise Fusion Architecture, EFA, ES7K, ES12K, ES14K, EXAScaler, GRIDScaler, GS7K, GS12K, GS14K, IME, IME14K, Infinite Memory Engine, Information in Motion, In-Storage Processing, MEDIAScaler, NAS Scaler, NoFS, ObjectAssure, ReACT, SFA, SFA 10000 Storage Fusion Architecture, SFA10K, SFA12K, SFA14K, SFA7700, SFX, Storage Fusion Architecture, Storage Fusion Fabric, Storage Fusion Xcelerator, SwiftCluster, WOS, the WOS logo are registered trademarks or trademarks of DataDirect Networks, Inc. All other brand and product names are trademarks of their respective holders.

DataDirect Networks makes no warranties, express or implied, including without limitation the implied warranties of merchantability and fitness for a particular purpose of any products or software. DataDirect Networks does not warrant, guarantee or make any representations regarding the use or the results of the use of any products or software in terms of correctness, accuracy, reliability, or otherwise. The entire risk as to the results and performance of the product and software are assumed by you. The exclusion of implied warranties is not permitted by some jurisdictions; this exclusion may not apply to you.

In no event will DataDirect Networks, their directors, officers, employees, or agents (collectively DataDirect Networks) be liable to you for any consequential, incidental, or indirect damages, including damages for loss of business profits, business interruption, loss of business information, and the like, arising out of the use or inability to use any DataDirect product or software even if DataDirect Networks has been advised of the possibility of such damages by you. Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, these limitations may not apply to you. DataDirect Networks liability to you for actual damages from any cause whatsoever, and regardless of the form of the action (whether in contract, tort including negligence, product liability or otherwise), is limited to the sum you paid for the DataDirect product or software.

All Products cited in this document are subject to DDN’s Limited Warranty Statement and, where applicable, the terms of DDN’s End User Software License Agreement (EULA).

The cited documents are available at: <http://www.ddn.com/support/policies/>

For archived versions of either document, please contact DDN.

October 2017

Table of Contents

[1.1 Terminology 10](#_Toc514766635)

[1.2 collectd plugins of DDN 11](#_Toc514766636)

[2.1 Installation Server 12](#_Toc514766637)

[2.2 Monitoring Server 12](#_Toc514766638)

[2.3 Monitoring Agent 12](#_Toc514766639)

[2.4 SFA 12](#_Toc514766640)

[3.1 Preparing the Installation Server 13](#_Toc514766641)

[3.2 Updating the configuration 13](#_Toc514766642)

[3.3 Running installation on the cluster 16](#_Toc514766643)

[3.4 Accessing the Monitoring Web Page 17](#_Toc514766644)

[4.1 Cluster Status Dashboard 18](#_Toc514766645)

[4.2 Lustre Status Dashboard 19](#_Toc514766646)

[4.3 Lustre MDS Statistics 41](#_Toc514766647)

[4.4 Lustre OSS Statistics 56](#_Toc514766648)

[4.5 Server Statistics 71](#_Toc514766649)

[4.6 SFA Physical Disk Dashboard 76](#_Toc514766650)

[4.7 SFA Virtual Disk Dashboard 79](#_Toc514766651)

[5.1 Installing stress2 RPM on collectd Client 82](#_Toc514766652)

[5.2 Updating Configuration File of Collectd Client 82](#_Toc514766653)

[5.3 Start Testing 83](#_Toc514766654)

List of Figures

[Figure 1: Grafana Login Web Page 17](#_Toc514766655)

[Figure 2: Home Dashboard 18](#_Toc514766656)

[Figure 3: Cluster Status Dashboard 19](#_Toc514766657)

[Figure 4: Lustre Statistics Dashboard 19](#_Toc514766658)

[Figure 5: Free Capacity in Total Panel 20](#_Toc514766659)

[Figure 6: Used Capacity in Total Panel 20](#_Toc514766660)

[Figure 7: Free Capacity per OST Panel 21](#_Toc514766661)

[Figure 8: Used Capacity per OST Panel 21](#_Toc514766662)

[Figure 9: Used Capacity per User Panel 22](#_Toc514766663)

[Figure 10: Used Capacity per Group Panel 22](#_Toc514766664)

[Figure 11: Free Inode Number in Total Panel 23](#_Toc514766665)

[Figure 12: Used Inode Number in Total Panel 23](#_Toc514766666)

[Figure 13: Free Inode Number per MDT Panel 24](#_Toc514766667)

[Figure 14: Used Inode Number per User Panel 24](#_Toc514766668)

[Figure 15: Used Inode Number Per Group Panel 25](#_Toc514766669)

[Figure 16: Used Inode Number per MDT Panel 25](#_Toc514766670)

[Figure 17: I/O Throughput in Total Panel 26](#_Toc514766671)

[Figure 18: I/O Throughput per OST Panel 26](#_Toc514766672)

[Figure 19: Write Throughput per OST Panel 27](#_Toc514766673)

[Figure 20: Read Throughput per OST Panel 27](#_Toc514766674)

[Figure 21: Metadata Operation Rate in Total Panel 28](#_Toc514766675)

[Figure 22: Metadata Operation Rate Per MDT Panel 28](#_Toc514766676)

[Figure 23: Metadata Operation Rate per Client Panel 29](#_Toc514766677)

[Figure 24: Metadata Operation Rate per Type Panel 29](#_Toc514766678)

[Figure 25: Write Bulk RPC Rate per Size 30](#_Toc514766679)

[Figure 26: Size Distribution of Write Bulk RPC Panel 30](#_Toc514766680)

[Figure 27: Read Bulk RPC Rate per Size Panel 31](#_Toc514766681)

[Figure 28: Size Distribution of Read Bulk RPC Panel 31](#_Toc514766682)

[Figure 29: Distribution of Discontinuous Pages in Each Write I/O Panel 32](#_Toc514766683)

[Figure 30: Distribution of Discontinuous Pages in Each Read I/O Panel 32](#_Toc514766684)

[Figure 31: Distribution of Discontinuous Blocks in Each Write I/O Panel 33](#_Toc514766685)

[Figure 32: Distribution of Discontinuous Blocks in Each Read I/O Panel 33](#_Toc514766686)

[Figure 33: Distribution of Fragments in Each Write I/O Panel 34](#_Toc514766687)

[Figure 34: Distribution of Fragments in Each Read I/O Panel 34](#_Toc514766688)

[Figure 35: Distribution of in-flight Write I/O Number when Starting Each Write I/O Panel 35](#_Toc514766689)

[Figure 36: Distribution of in-flight Read I/O Number when Starting Each Read I/O Panel 35](#_Toc514766690)

[Figure 37: Distribution of Write I/O Time Panel 36](#_Toc514766691)

[Figure 38: Distribution of Read I/O Time Panel 36](#_Toc514766692)

[Figure 39: Distribution of Write I/O size on Disk Panel 37](#_Toc514766693)

[Figure 40: Distribution of Read I/O Size on Disk Panel 37](#_Toc514766694)

[Figure 41: Write Throughput per Client Panel 38](#_Toc514766695)

[Figure 42: Read Throughput per Client Panel 38](#_Toc514766696)

[Figure 43: I/O Throughput per Job Panel 39](#_Toc514766697)

[Figure 44: Write Throughput per Job Panel 39](#_Toc514766698)

[Figure 45: Read Throughput per Job Panel 40](#_Toc514766699)

[Figure 46: Metadata Performance per Job Panel 40](#_Toc514766700)

[Figure 47: Lustre MDS Statistics Dashboard 41](#_Toc514766701)

[Figure 48：Number of Active Requests Panel 41](#_Toc514766702)

[Figure 49：Number of Incoming Requests Panel 42](#_Toc514766703)

[Figure 50：Wait time of Requests Panel 42](#_Toc514766704)

[Figure 51：Adaptive Timeout Value Panel 43](#_Toc514766705)

[Figure 52：Number of Available Request buffers Panel 43](#_Toc514766706)

[Figure 53：Handing time of LDLM ibits Enqueue Requests Panel 44](#_Toc514766707)

[Figure 54：Handing Time of Getattr Requests Panel 45](#_Toc514766708)

[Figure 55：Handing Time of Connect Requests Panel 45](#_Toc514766709)

[Figure 56：Handing Time of Get-root Requests Panel 46](#_Toc514766710)

[Figure 57：Handing Time of Statfs Requests Panel 46](#_Toc514766711)

[Figure 58：Handing Time of Getxattr Requests Panel 47](#_Toc514766712)

[Figure 59：Handing Time of Ping Requests Panel 47](#_Toc514766713)

[Figure 60：Number of Active Readpage Requests Panel 48](#_Toc514766714)

[Figure 61：Number of Incoming Readpage Requests Panel 48](#_Toc514766715)

[Figure 62：Wait Time of Readpage Requests Panel 49](#_Toc514766716)

[Figure 63：Adaptive Timeout Value of Readpage Service 49](#_Toc514766717)

[Figure 64：Number of Available Readpage Request Buffers Panel 50](#_Toc514766718)

[Figure 65：Handing Time of Close Requests Panel 50](#_Toc514766719)

[Figure 66：Handing Time of Readpage Requests Panel 51](#_Toc514766720)

[Figure 67：Number of Active LDLM Canceld Requests Panel 51](#_Toc514766721)

[Figure 68：Number of Incoming LDLM Canceld Requests Panel 52](#_Toc514766722)

[Figure 69：Wait Time of LDLM Canceld Requests Panel 52](#_Toc514766723)

[Figure 70：Adaptive Timeout Value of LDLM Canceld Service 53](#_Toc514766724)

[Figure 71：Number of Available LDLM Canceld Request Buffers Panel 53](#_Toc514766725)

[Figure 72：Number of Active LDLM Callback Requests Panel 54](#_Toc514766726)

[Figure 73：Number of Incoming LDLM Callback Requests Panel 54](#_Toc514766727)

[Figure 74：Wait time of LDLM Callback Requests Panel 55](#_Toc514766728)

[Figure 75：Adaptive Timeout Value of LDLM Callback Service Panel 55](#_Toc514766729)

[Figure 76：Number of Available LDLM Callback Request Buffers Panel 56](#_Toc514766730)

[Figure 77: Lustre OSS Dashboard 56](#_Toc514766731)

[Figure 78：I/O Bandwidth Panel 57](#_Toc514766732)

[Figure 79: Number of Active Requests Panel 57](#_Toc514766733)

[Figure 80: Number of Incoming Requests Panel 58](#_Toc514766734)

[Figure 81: Wait time of Requests Panel 58](#_Toc514766735)

[Figure 82: Adaptive Timeout Value Panel 59](#_Toc514766736)

[Figure 83: Number of Available Request Buffers Panel 59](#_Toc514766737)

[Figure 84: Number of Active I/O Requests Panel 60](#_Toc514766738)

[Figure 85: Number of Incoming I/O Requests Panel 60](#_Toc514766739)

[Figure 86: Wait Time of I/O requests Panel 61](#_Toc514766740)

[Figure 87: Adaptive Timeout Value of I/O Service Panel 61](#_Toc514766741)

[Figure 88: Number of Available I/O Request Buffers 62](#_Toc514766742)

[Figure 89: Handing Time of Punch Requests Panel 62](#_Toc514766743)

[Figure 90: Handing Time of Read Requests Panel 63](#_Toc514766744)

[Figure 91: Handing Time of Write Requests Panel 63](#_Toc514766745)

[Figure 92: Number of Active Create Requests Panel 64](#_Toc514766746)

[Figure 93: Number of Incoming Create Requests Panel 64](#_Toc514766747)

[Figure 94: Wait Time of Create Requests Panel 65](#_Toc514766748)

[Figure 95: Adaptive Timeout Value of Create Service Panel 65](#_Toc514766749)

[Figure 96: Number of Available Create Request Buffers Panel 66](#_Toc514766750)

[Figure 97: Number of Active LDLM Canceld Requests Panel 66](#_Toc514766751)

[Figure 98: Number of Incoming LDLM Canceld Requests Panel 67](#_Toc514766752)

[Figure 99: Wait Time of LDLM Canceld Requests Panel 67](#_Toc514766753)

[Figure 100: Adaptive Timeout Value of LDLM Canceld Service Panel 68](#_Toc514766754)

[Figure 101: Number of Available LDLM Canceld Request Buffers Panel 68](#_Toc514766755)

[Figure 102: Number of Active LDLM Callback Requests Panel 69](#_Toc514766756)

[Figure 103: Number of Incoming LDLM Callback Requests Panel 69](#_Toc514766757)

[Figure 104: Wait Time of LDLM Callback Requests Panel 70](#_Toc514766758)

[Figure 105: Adaptive Timeout Value of LDLM Callback Service Panel 70](#_Toc514766759)

[Figure 106: Number of Available LDLM Callback Request Buffers 71](#_Toc514766760)

[Figure 107: Server Statistics Dashboard 71](#_Toc514766761)

[Figure 108: CPU Usage Panel 72](#_Toc514766762)

[Figure 109: Memory Usage Panel 72](#_Toc514766763)

[Figure 110: Disk Write Rate Panel 73](#_Toc514766764)

[Figure 111: Disk Read Rate Panel 73](#_Toc514766765)

[Figure 112: Disk Usage on Root Panel 74](#_Toc514766766)

[Figure 113: Load Panel 74](#_Toc514766767)

[Figure 114: Uptime Panel 75](#_Toc514766768)

[Figure 115: User Panel 75](#_Toc514766769)

[Figure 116: Temperature Panel 75](#_Toc514766770)

[Figure 117: SFA Physical Disk Dashboard 76](#_Toc514766771)

[Figure 118: I/O Performance on Physical Disk Panel 76](#_Toc514766772)

[Figure 119: IOPS on Physical Disk Panel 77](#_Toc514766773)

[Figure 120: Bytes per I/O on Physical Disk Panel 77](#_Toc514766774)

[Figure 121: Write Performance on Physical Disk Panel 78](#_Toc514766775)

[Figure 122: Write I/O Size Samples on Physical Disk Panel 78](#_Toc514766776)

[Figure 123: Write Latency Samples on Physical Disk Panel 78](#_Toc514766777)

[Figure 124: SFA Virtual Disk Dashboard 79](#_Toc514766778)

[Figure 125: I/O Performance on Virtual Disk Panel 79](#_Toc514766779)

[Figure 126: I/O Operations per Second on Virtual Disk Panel 80](#_Toc514766780)

[Figure 127: Bytes per I/O on Virtual Disk Panel 80](#_Toc514766781)

[Figure 128: Write Performance on Virtual Disk Panel 80](#_Toc514766782)

[Figure 129: Write I/O Size Samples on Virtual Disk Panel 81](#_Toc514766783)

[Figure 130: Write Latency Samples on Virtual Disk Panel 81](#_Toc514766784)

[Figure 131：Influxdb Query Time 85](#_Toc514766785)

[Figure 132：Read throughput per Job stress testing 85](#_Toc514766786)

# Introduction to DDN EXAScaler Performance Monitoring System

*LustrePerfMon* is a monitoring system that can collect system statistics of DDN EXAScaler for performance monitoring and analysis. It is based on multiple widely used open-source software. Some extra plugins have been developed by DDN for enhancement.

One of the main components of *LustrePerfMon* is **сollectd**. **collectd** is a daemon, which collects system performance statistics periodically and provides mechanisms to store the values in a variety of ways. *LustrePerfMon* is based on the open-source **collectd**, yet includes more plugins, such as Filedata, Ganglia, Nagios, Stress, Zabbix and so on.

## Terminology

|  |  |
| --- | --- |
| **LustrePerfMon** | Abbreviation for DDN EXAScaler Performance Monitoring System. |
| **DDN SFA** | DDN Storage Fusion Architecture provides the foundation for balanced, high-performance storage. Using highly parallelized storage processing technology, SFA delivers both unprecedented IOPS and massive throughput. |
| **DDN EXAScaler** | Software stack developed by DDN to overcome the toughest storage and data management challenges in extreme, data-intensive environments. |
| **Installation Server** | The server on which the installation process is triggered. |
| **Monitoring Server** | The server on which the database (Influxdb) and web server (Grafana) of the monitoring system will run. |
| **Monitoring Agent(s)** | The hosts, from which the monitoring system will collect metrics from. The metrics includes information about CPU, memory, Lustre, SFA storage, and so on. A collectd daemon will run on each monitoring client. |
| **DDN IME** | DDN’s Infinite Memory Engine (IME) is a flash-native, software-defined, storage cache that streamlines application IO, eliminating system bottlenecks. |
| **Lustre** | The Lustrefile system is an open-source, parallel file system that supports many requirements of leadership class HPC simulation environments. |
| **OST** | The *Object Storage Target(OST)* of *Lustre* is the storage target that store the file data objects. |
| **OSS** | The *Object Storage Server(OSS)* of *Lustre* is the server that manage the *Object Storage Target*. |
| **MDT** | The *Metadata Target(MDT)* of *Lustre* is the storage target that stores the file metadata. |
| **MDS** | The *Metadata Servers(MDS)* of *Lustre* is the server that provides metadata services for a file system and manages one or multiple Metadata Target (MDT). |

## Collectd plugins of DDN

Several additional plugins are added to **collectd** in LustrePerfMon to support various functions.

* **Filedata plugin**: The **Filedata** plugin is able to collect data by reading and parsing a set of files. An XML-formatted definition file is required for the **Filedata** plugin to understand which files to read and how to parse these files. The most common usage of the **Filedata** plugin is to collect metrics through /proc interfaces of a running Lustre system.
* **Ganglia plugin**: The **Ganglia** plugin can send metrics collected by a **collectd** client daemon to Ganglia server.
* **IME plugin**: The **IME** plugin can collect performance information from **DDN IME**. The **IME** plugin shares the similar definition file format and configuration format with the **Filedata** plugin.
* **SSH plugin**: The **SSH** plugin is able to collect metrics by running commands on remote hosts by using SSH connections. The **SSH** plugin is used to collect metrics from DDN SFA Storage. Like the **IME** plugin, the **SSH** plugin shares the similar definition file format and configuration format with the **Filedata** plugin.
* **Stress plugin**: The **Stress** plugin can push a large amount of metrics to server from **collectd** client in order to benchmark the performance of the collecting system under high pressure.
* **Stress2 plugin**: Enhanced version of **Stress** plugin. The format of pushed metrics can be flexibly configured to simulate different real metrics.
* **Zabbix plugin**: The Zabbix plugin is used to send metrics from **collectd** to **Zabbix** system.

# Installation Requirements

## Installation Server

* **OS distribution:** CentOS7/RHEL7
* **Free disk space:** > 500 MB. The Installation Server will save all installation logs to the /var/log/esmon\_install directory, which requires some free disk space.
* **Network:** The Installation Server must be able to start SSH connections to the Monitoring Server and Monitoring Clients without a password prompt.
* **LustrePerfMon ISO image:** The LustrePerfMon ISO image must be available on the Installation Server.
* **Clock and Time Zone:** The clock and time zone should be synchronized with other nodes in the cluster.

## Monitoring Server

* **OS distribution:** CentOS7/RHEL7
* **Free disk space:** > 5 GB. Influxdb will be running on this server. More disk space is required to keep more data in Influxdb
* **Network:** SSHD should be running on the Monitoring Server. The Installation Server must be able to connect to the Monitoring Server without a password prompt.
* **Clock and Time Zone:** The clock and time zone should be synchronized with other nodes in the cluster.

## Monitoring Agent

* **OS distribution:** CentOS7/RHEL7 or CentOS6/RHEL6
* **Free disk space:** > 200 MB. The installation server will save necessary RPMs in directory /var/log/esmon\_install, which requires some free disk space.
* **Network:** SSHD should be running on the Monitoring Agent. The Installation Server must be able to connect to the Monitoring Agent without a password prompt.
* **EXAScaler version:** EXAScaler 2.x, EXAScaler 3.x or EXAScaler4.x.
* **Clock and Time Zone:** The clock and time zone should be synchronized with other nodes in the cluster.

## SFA

* **Firmware release:** 3.x or 11.x

# Installation Process

## Preparing the Installation Server

1. Copy the LustrePerfMon ISO image file to the Installation Server, for example, to /ISOs/esmon.iso.
2. Mount the LustrePerfMon ISO image on the Installation Server:

mount -o loop /ISOs/esmon.iso /media

1. On the Installation Server, back up old LustrePerfMon configuration file, if there is any:

cp /etc/esmon\_install.conf /etc/esmon\_install.conf\_backup

1. On the Installation Server, uninstall old LustrePerfMon RPM, if there is any:

rpm -e esmon

1. Install the LustrePerfMon RPM on the Installation Server:

rpm -ivh /media/RPMS/rhel7/esmon\*.rpm

## Monitoring Server

If firewall is started on the monitoring server, the ports 3000, 4242, 8086, 8088 and 25826 should be opened, otherwise the installation or running of LustrePerfMon might have problem. The 3000 port is for the webb interface of Grafana. The ports 4242, 8086, 8088, 25826 are for the data communication and management of Influxdb, Grafana and Collectd.

## Updating the configuration

After the LustrePerfMon RPM has been installed on the Installation Server, update the configuration file /etc/esmon\_install.conf, which includes all the necessary information for installation. Define the following parameters:

* In the section **agents**, specify information about all of the hosts where LustrePerfMon agent packages should be installed and configured:
  + **enable\_disk** —This option determines whether to collect disk metrics from this agent. Default value: **false**.
  + **host\_id** — This option is the ID of the host. The ID of a host is a unique value to identify the host. Two hosts should not share the same **host\_id**.
  + **ime** —This option determines whether to enable IME metrics collection on this LustrePerfMon agent. Default value: **false**.
  + **infiniband** —This option determines whether to enable Infiniband metrics collection on this LustrePerfMon agent. Default value: **false**.
  + **lustre\_mds** — Define whether to enable (**true**) or disable (**false**) metrics collection of Lustre MDS. Default value: **true**.
  + **lustre\_oss** — Define whether to enable (**true**) or disable (**false**) metrics collection of Lustre OSS. Default value: **true**.
  + **sfas** — This list includes the information of DDN SFAs on this LustrePerfMon agent.
    - **controller0\_host** — This option is the hostname/IP of the controller 0 of this SFA. Default value: **controller0\_host**.
    - **controller1\_host** — This option is the hostname/IP of the controller 1 of this SFA. Default value: **controller1\_host**.
    - **Name** —This option is the unique name of this controller. This value will be used as the value of "fqdn" tag for metrics of this SFA. Thus, two SFAs shouldn't have the same **name**.
* **agents\_reinstall** — Define whether to reinstall (**true**) LustrePerfMon clients or not (**False**). Default value: **true**.
* **collect\_interval** — The interval (in seconds) to collect data points on LustrePerfMon clients. Default value: **60**.
* **continuous\_query\_interval** — The interval of continuous query. The value of **continuous\_query\_interval \* collect\_interval** is the real interval in seconds between two adjacent data points of each continuous query. Usually, in order to down sample the data and reduce performance impact, this value should be larger than "1". Default value: **4**.
* **iso\_path** — The path where the LustrePerfMon ISO image is saved. Default value: **/root/esmon.iso**.
* **lustre\_default\_version** — The default Lustre version to use, if the Lustre RPMs installed on the LustrePerfMon client is not the supported version. The current supported values of the parameter are **es2**, **es3**, **es4** and **error**. If the parameter **error** is configured, an error will be raised when an LustrePerfMon client is using an unsupported Lustre version.
* **lustre\_exp\_ost** — Define whether to enable (**true**) or disable (**false**) metrics collection of export information of Lustre OST. To avoid a flood of metrics, this parameter is usually disabled in Lustre file systems with a large number of clients. Default value: **false**.
* **lustre\_exp\_mdt** — Define whether to enable (**true**) or disable (**false**) metrics collection of export information of Lustre MDT. To avoid a flood of metrics, this parameter is usually disabled in Lustre file systems with a large number of clients. Default value: **false**.
* In the section **server**, specify information about all of the hosts where LustrePerfMon server packages should be installed and configured:
  + **drop\_database** —If the parameter is set to **true**, the LustrePerfMon database in Influxdb will be dropped. If the parameter is set to **false**, the LustrePerfMon database in Influxdb will be kept as it is. Default value: **false**.

**Important: drop\_database** should only be enabled when the data in Influxdb is not needed anymore.

* + **erase\_influxdb** — If the parameter is enabled (set to **true**), all the data and metadata of Influxdb will be completely erased. By enabling **erase\_influxdb**, some corruption problems of Influxdb could be fixed. If the parameter is disabled (set to **False**), the data and metadata of Influxdb will not be completely erased.

**Important: erase\_influxdb** should only be enabled when the data/metadata in Influxdb is not needed anymore. Please double check the influxdb\_path option is properly configured before enabling this option.

* + **host\_id** — The unique ID of the host.
  + **influxdb\_path** — This option is Influxdb directory path on LustrePerfMon server node.Default value: **/esmon/influxdb**.

**Important:** Please do not put any other files/directries under this directory of LustrePerfMon server node, because, with "erase\_influxdb" option enabled, all of the files/directries under that directory will be removed.

* + **reinstall** —This option determines whether to reinstall the LustrePerfMon server. Default value: **true**.
* In the section **ssh\_hosts**, specify details necessary to log in to the Monitoring Server and to each Monitoring Agent using SSH connection:
  + **host\_id** — The unique ID of the host. Two hosts should not share the same **host\_id**.
  + **hostname** — The hostname/IP to use when connecting to the host using SSH. "ssh" command will use this hostname/IP to login into the host. If the host is the LustrePerfMon server, this hostname/IP will be used as the server host in the **write\_tsdb** plugin of LustrePerfMon agent.
  + **ssh\_identity\_file** — The SSH key file used for connecting to the host. If the default SSH identity file works, this option can be set to **None**. Default value: **None**.
  + **local\_host** —This option determines whether this host is local host. Default value: **false**.

Note: **host\_id** and **hostname** can be different for a host, because there can be multiple ways to connect to the same host.

Below is an example of /etc/esmon\_install.conf:

**Example:**

agents:

- enable\_disk: false

host\_id: Agent1

ime: false

infiniband: false

lustre\_mds: true

lustre\_oss: true

sfas:

- controller0\_host: 10.0.0.1

controller1\_host: 10.0.0.2

name: SFA1

- controller0\_host: 10.0.0.3

controller1\_host: 10.0.0.4

name: SFA2

- host\_id: Agent2

sfas: []

agents\_reinstall: true

collect\_interval: 60

continuous\_query\_interval: 4

iso\_path: /root/esmon.iso

lustre\_default\_version: es3

lustre\_exp\_mdt: false

lustre\_exp\_ost: false

server:

drop\_database: false

erase\_influxdb: false

host\_id: Server

influxdb\_path: /esmon/influxdb

reinstall: true

ssh\_hosts:

- host\_id: Agent1

hostname: Agent1

local\_host: false

ssh\_identity\_file: None

- host\_id: Agent2

hostname: Agent2

- host\_id: Server

hostname: Server

## Running installation on the cluster

After the */etc/esmon\_install.conf* file has been updated correctly on the Installation Server, run the following command to start the installation on the cluster:

esmon\_install

All the logs that are useful for debugging are saved under /var/log/esmon\_install directory of the Installation Server.

Apart from installing LustrePerfMon on a fresh system, the command**esmon\_install**can also be used for upgrading an existing LustrePerfMon system. The configuration file */etc/esmon\_install.conf* should be backed up after installation of LustrePerfMon in case of upgrading in the future.

**Important:** When upgrading an existing LustrePerfMon system, **erase\_influxdb** and **drop\_database** should be disabled, unless the data or metadata in Influxdb is not needed anymore.

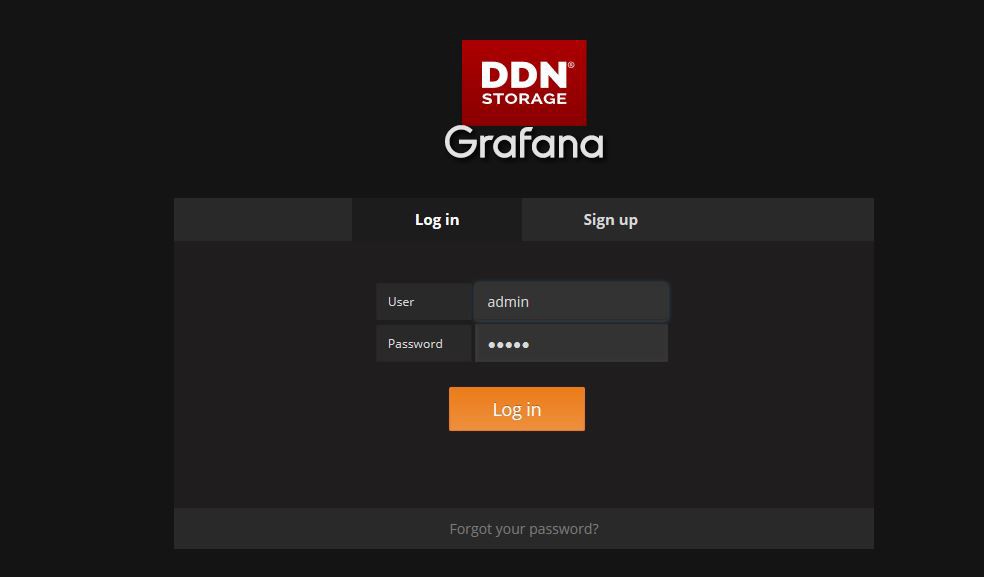
When installing or upgrading, **esmon\_install** will cleanup and install the default LustrePerfMon dashboards of Grafana. Except for the default LustrePerfMon dashboards, **esmon\_install** will not change any other existing dashboards of Grafana.

**Important:** Before upgrading an existing LustrePerfMon system, all default LustrePerfMon dashboards customized via a Grafana web page should be saved under different names, otherwise the modifications will be overwritten.

## Accessing the Monitoring Web Page

The Grafana service is started on the Monitoring Server automatically. The default HTTP port is 3000. A login web page will be shown through that port (see Figure 1 below). The default user and password are both “admin”.

Figure 1: Grafana Login Web Page

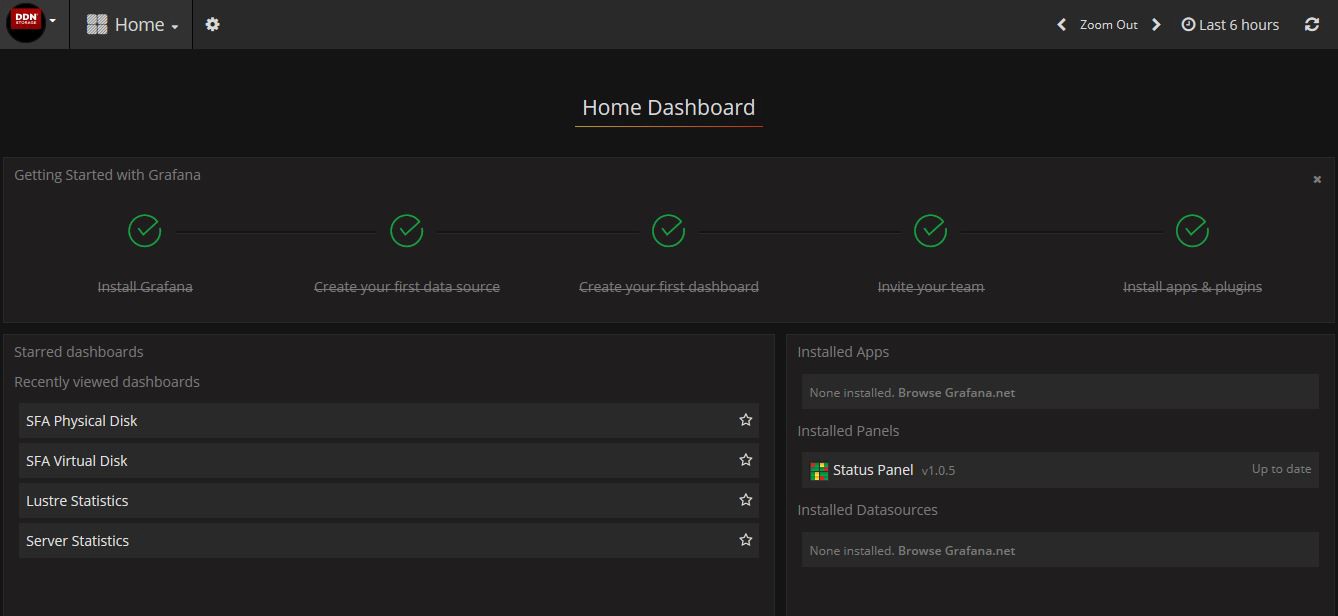


**Important:** The host that runs the web browser to access the monitoring web page should have the same time clock and time zone with the servers. Otherwise, the monitoring results might be shown incorrectly.

# Dashboards

From the Home dashboard (see Figure 2) different dashboards can be chosen to view different metrics collected by LustrePerfMon.

Figure 2: Home Dashboard

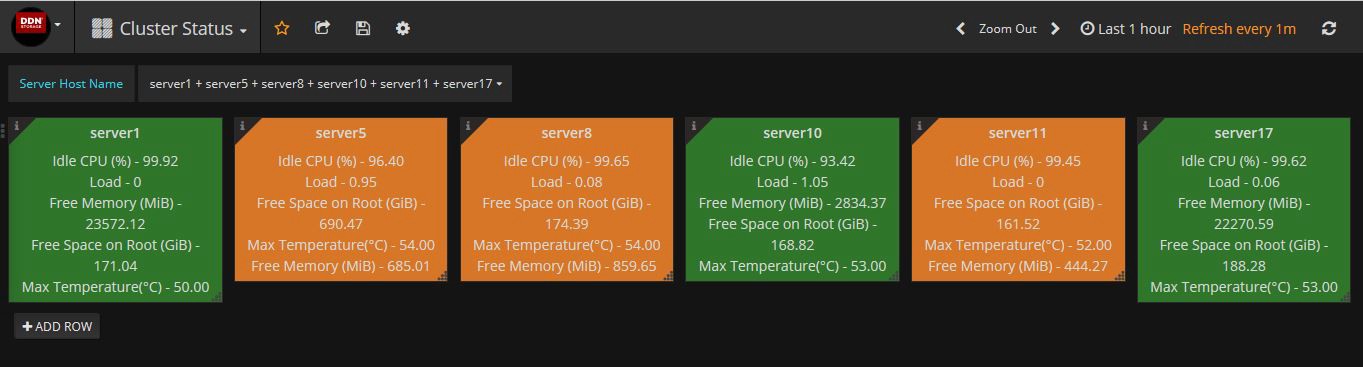


## Cluster Status Dashboard

The **Cluster Status** dashboard (see Figure 3 below) shows a summarized status of the servers in the cluster. The background color of panels show the servers’ working status:

* If the color of the panel is green, it means the server is under normal condition.
* If the color of the panel is yellow, it means the server is under warning status due to one or more of the following conditions:
  + Idle CPU is less than 20%
  + Load is higher than 5
  + Free memory is less than 1000 MiB
  + Free space of “/” is less than 10 GiB
* If the color of the panel is red, it means the server is under critical status due to one or more of the following conditions:
  + Idle CPU is less than 5%
  + Load is higher than 10
  + Free space of “/” is less than 1 GiB
  + Free memory is less than 100 MiB

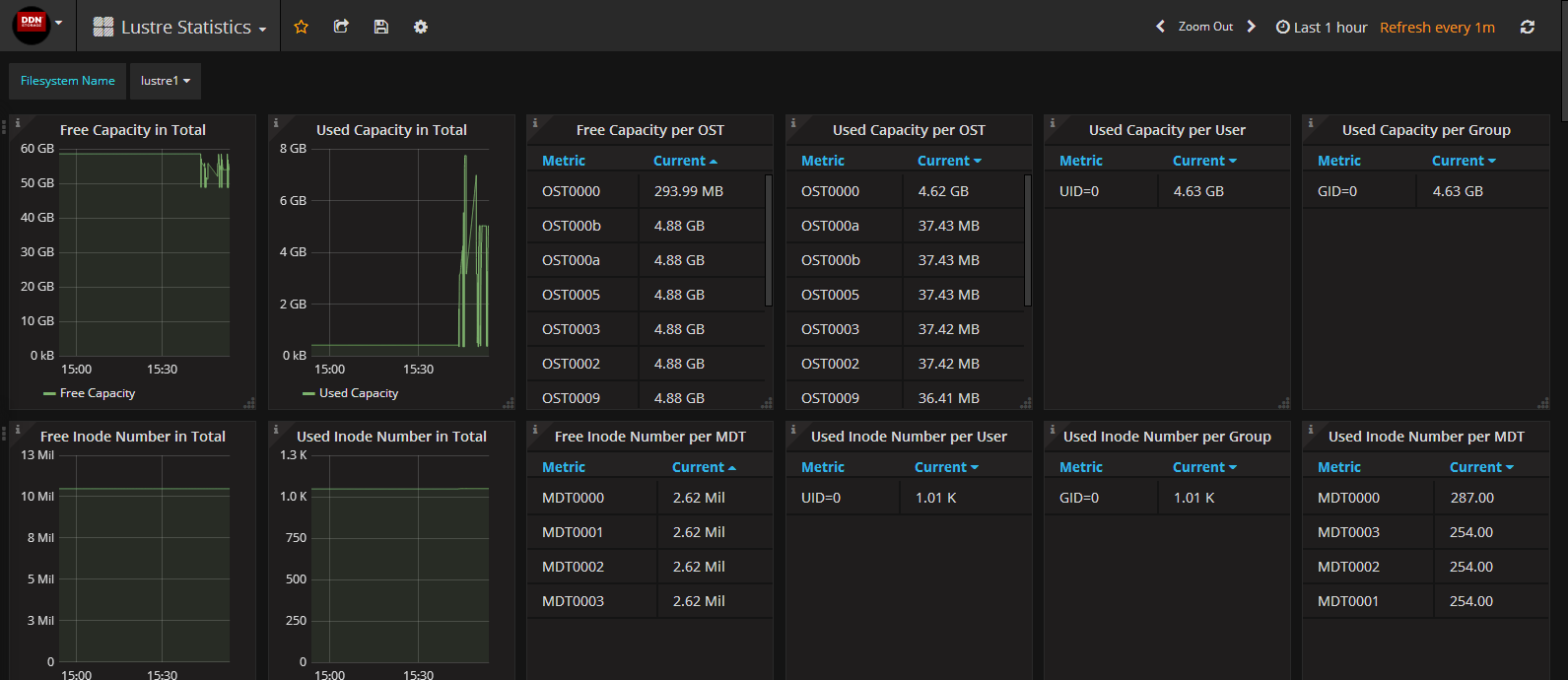
Figure 3: Cluster Status Dashboard



## Lustre Status Dashboard

The Lustre Statistics dashboard (Figure 4) shows metrics of Lustre file systems.

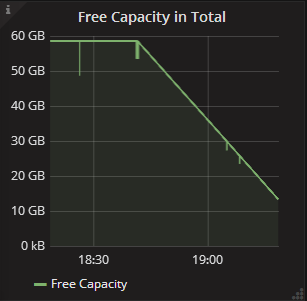
Figure 4: Lustre Statistics Dashboard



The following pictures are some of the panels in the **Lustre Statistics**dashboard.

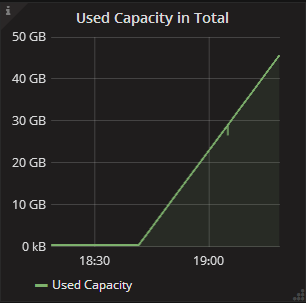
* The **Free Capacity in Total** panel (Figure 5) shows how much free capacity remains in the Lustre filesystem. The test case used in the figure is running “dd if=/dev/zero of=/mnt/lustre/file bs=1M” from about 18:40, and it shows that the free capacity is being consumed at a speed of about 20MB/s.

Figure 5: Free Capacity in Total Panel



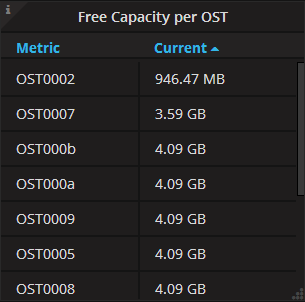
* The **Used Capacity in Total** panel (Figure 6) shows how much capacity in total is used in the Lustre filesystem. The test case used in the figure is running “dd if=/dev/zero of=/mnt/lustre/file bs=1M” from about 18:40, and it can be seen from the figure that the used capacity has increased at the rate of about 20 MB/s.

Figure 6: Used Capacity in Total Panel



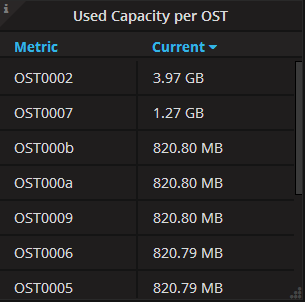
The **Free Capacity per OST** panel (Figure 7) shows how much free capacity per OST remains in the Lustre filesystem. As shown in the figure, OST0002 free capacity is 946.47MB, OST0007 free capacity is 3.59GB, the free capacity of the remaining OSTs is 4.09GB each. To display the current free capacity per OST in the ascending or descending order, click on **Current**..

Figure 7: Free Capacity per OST Panel



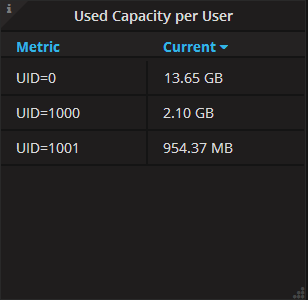
The **Used Capacity per OST** panel (Figure 8) shows how much capacity per OST is used in the Lustre filesystem. As shown in the figure, the used capacity of OST0002 is 3.97GB, the used capacity of OST0007 is 1.27GB, the used capacity of the remaining OSTs is 820.8MB. To display the current used capacity per OST in the ascending or descending order, click on **Current**.

Figure 8: Used Capacity per OST Panel



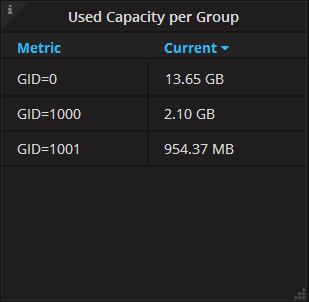
The Used Capacity per User panel (Figure 9) shows how much capacity per user is used in the Lustre filesystem. As shown in the figure, the current used capacity of the user with UID=0 is 13.65GB, the current used capacity of the user with UID=1000 is 2.10GB, the current used capacity of the user with UID=1001 is 954.37MB.

Figure 9: Used Capacity per User Panel



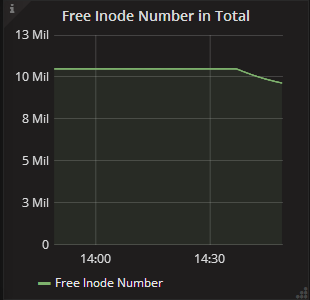
The **Used Capacity per Group** panel (Figure 10) shows how much capacity per group is used in the Lustre filesystem. As shown in the figure, the current used capacity of the group with GID=0 is 13.65GB, the current used capacity of the group with GID=1000 is 2.10GB, the current used capacity of the group with GID=1001 is 954.37MB.

Figure 10: Used Capacity per Group Panel



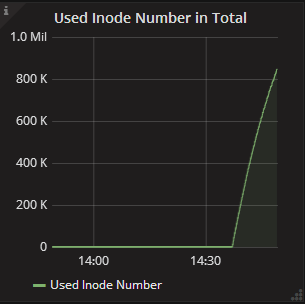
The **Free Inode Number in Total** panel (Figure 11) shows the total number of free inodes in the Lustre filesystem over time. The test case used in the figure is running“mdtest–C –n 900000 –d /mnt/lustre/mdtest/” from about 14:35. From the figure it can be seen that from that time on, the free inode number is decreased and exhausted at a speed of about 1100 Ops (Operation per Second).

Figure 11: Free Inode Number in Total Panel



The **Used Inode Number in Total** panel (Figure 12) shows the total number of used inodes in the Lustre filesystem over time. The test case used in the figure is running “mdtest–C –n 900000 –d /mnt/lustre/mdtest/” from about 14:35, from the figure it can be seen that the used inode number is increased in a speed of about 1100 Ops (Operation per Second).

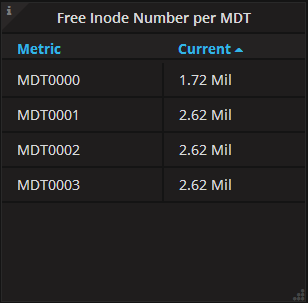
Figure 12: Used Inode Number in Total Panel



The **Free Inode Number per MDT** panel (. By clicking on the “Current”, the current free inode number per MDT in the system can be sorted in the ascending of descending order. To display the current free inode number per MDT in the ascending or descending order, click on **Current**.

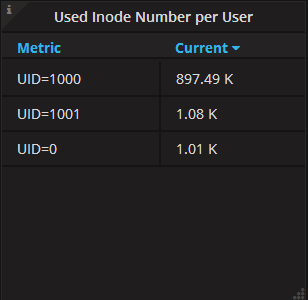
**Figure 13**) shows the current number of free inodes per MDT in the Lustre filesystem. As shown in the figure, the number of free inodes of MDT0000 is 1.72Mil, the number of free inodes of all other MDTs is 2.62 Mil. By clicking on the “Current”, the current free inode number per MDT in the system can be sorted in the ascending of descending order. To display the current free inode number per MDT in the ascending or descending order, click on **Current**.

Figure 13: Free Inode Number per MDT Panel



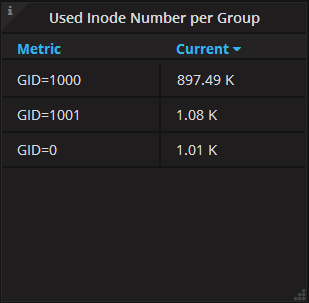
The **Used Inode Number per User** panel (Figure 14) shows the number of used inodes per user in the Lustre filesystem. As shown in the figure, the number of used nodes pertaining to the user with UID=1000 is 897.49K, the number of used inodes of the user with UID=1001 is 1.08K, the number of used inodes of the user with UID=0 is 1.01K. To display the current number of used inodes per user in the ascending or descending order, click on **Current**.

Figure 14: Used Inode Number per User Panel



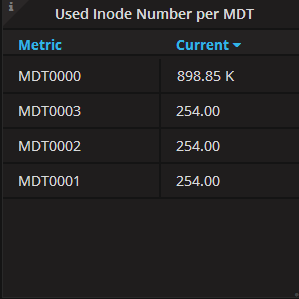
The **Used Inode Number per Group** panel (Figure 15) shows the number of used inodes per group in the Lustre Filesystem. As shown in the figure, the number of used inodes of the group with GID=1000 is 897.49K, the number of used inodes of the group with GID=1001 is 1.08K, the number of used inodes of the group with GID=0 is 1.01K. To display the current number of used inodes per group in the ascending or descending order, click on **Current**.

Figure 15: Used Inode Number Per Group Panel



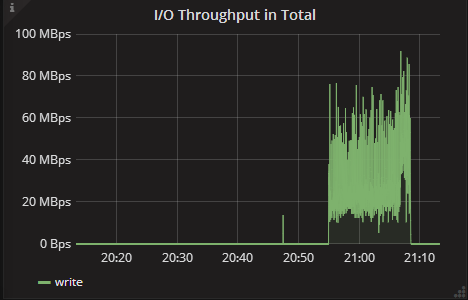
The **Used Inode Number per MDT** (Figure 16) shows the inode number per MDT used in the Lustre Filesystem. As shown in the figure, MDT0000 used inode number is 898.85K, MDT0001 is 254.

Figure 16: Used Inode Number per MDT Panel



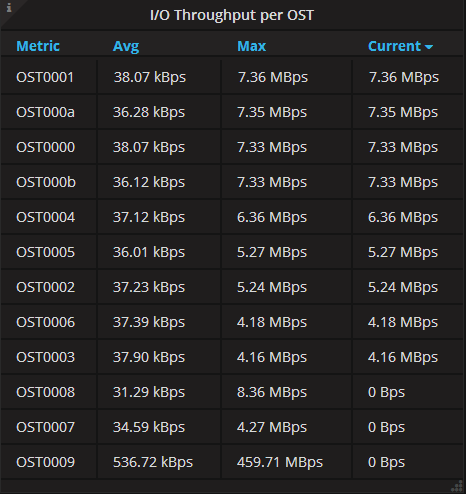
The **I/O Throughput in Total** panel (Figure 17) shows the total I/O throughput in the Lustre filesystem over time.

Figure 17: I/O Throughput in Total Panel



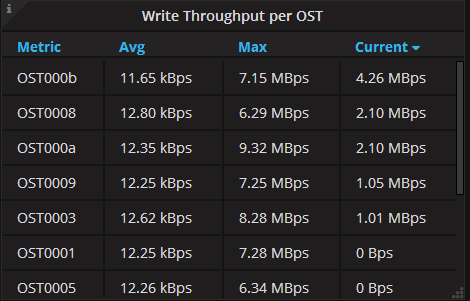
The **I/O Throughput per OST** panel (Figure 18) shows the average, maximum, and current I/O throughput per OST in the Lustre filesystem.

Figure 18: I/O Throughput per OST Panel



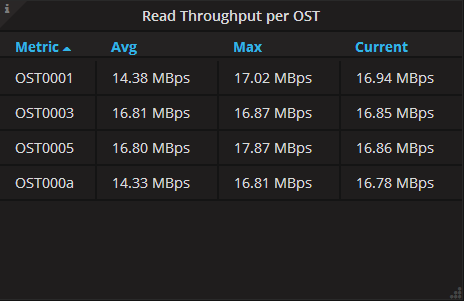
The **Write Throughput per OST** panel (Figure 19) shows the average, maximum, and current write throughput per OST in the Lustre Filesystem.

Figure 19: Write Throughput per OST Panel



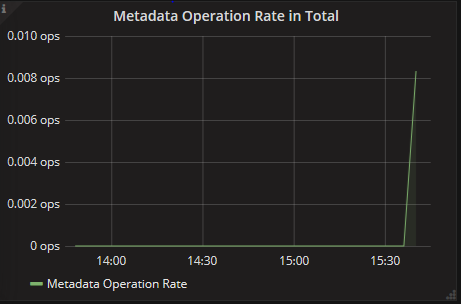
The **Read Throughput per OST** panel (Figure 20) shows the average, maximum, and current read throughput per OST in the Lustre Filesystem.

Figure 20: Read Throughput per OST Panel



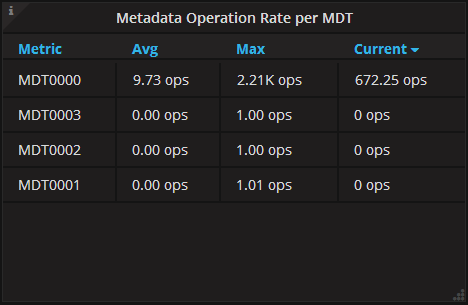
The **Metadata Operation Rate in Total** panel (Figure 21) shows the total metadata operation rate in the Lustre Filesystem over time. The unit is Ops, i.e. Operation Per Second.

Figure 21: Metadata Operation Rate in Total Panel



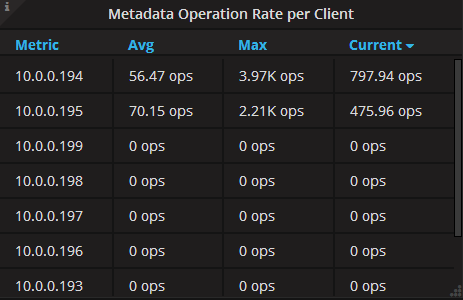
The **Metadata Operation Rate per MDT** panel (Figure 22) shows the metric information of the metadata operation rate per MDT in the Lustre filesystem. The unit is OPS (Operation Per Second). The information includes the average, maximum, and current values.

Figure 22: Metadata Operation Rate Per MDT Panel



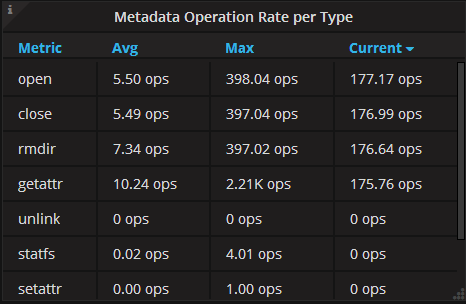
The **Metadata Operation Rate per Client** panel (Figure 23) shows the metric information of the metadata operation rate per client in the Lustre filesystem. The unit is OPS. The information includes the average, maximum, and current values.

Figure 23: Metadata Operation Rate per Client Panel



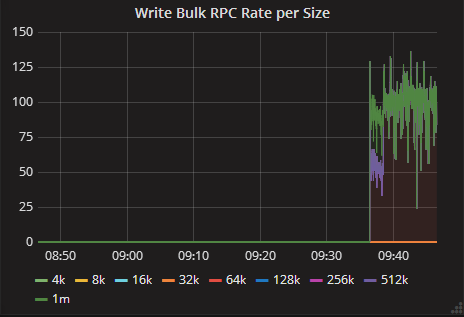
The **Metadata Operation Rate per Type** panel (Figure 24) shows the metric information of the metadata operation rate per type in the Lustre filesystem. The unit is OPS. The information includes the average, maximum, and current values. The current test case used is the operations that remove all files in a directory.

Figure 24: Metadata Operation Rate per Type Panel



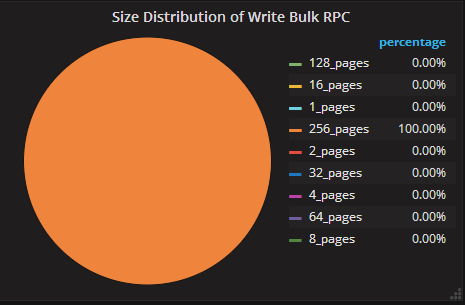
The **Write Bulk RPC Rate per Size** panel (Figure 25) shows the write bulk RPC rate with different size in the Lustre Filesystem over time. The size of Lustre Bulk RPC could be a value between 4KiB and 16MiB. The figure below shows the information of write RPC Rate with different bulk size. The test case that generated the collected information is that two clients run ”dd if=/dev/zero of=/mnt/lustre/test1 bs=1M oflag=direct”, “dd if=/dev/zero of=/mnt/lustre/test2 bs=64k oflag=direct”, respectively.

Figure 25: Write Bulk RPC Rate per Size



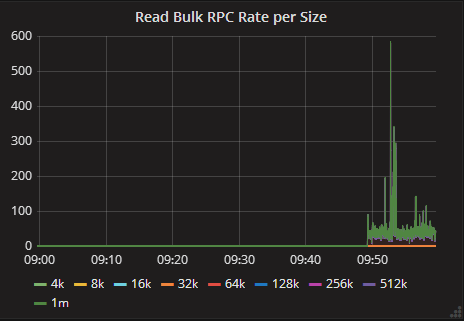
The **Size Distribution of Write Bulk RPC** panel (Figure 26) shows the ratio information of the write bulk RPC with different bulk size in the Lustre Filesystem. As shown in the figure, the percentage of total for the number of the write bulk RPC number with 256 pages is 100%.

Figure 26: Size Distribution of Write Bulk RPC Panel



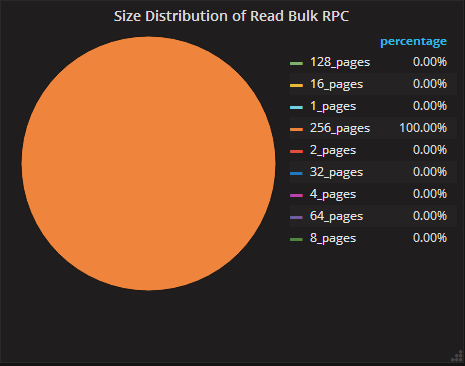
The **Read Bulk RPC Rate per Size** panel (Figure 27) shows the read bulk RPC rate per size in the Lustre filesystem over time. The size of Lustre Bulk RPC could be a value between 4KiB and 16MiB. The figure below shows the read RPC rate with different bulk I/O size. The used test case to generate the collected information is that two clients run “dd if=/mnt/lustre/test1 of=/dev/zero bs=1M iflag=direct” and “dd if=/mnt/lustre/test2 of=/dev/zero bs=64k iflag=direct”, respectively.

Figure 27: Read Bulk RPC Rate per Size Panel



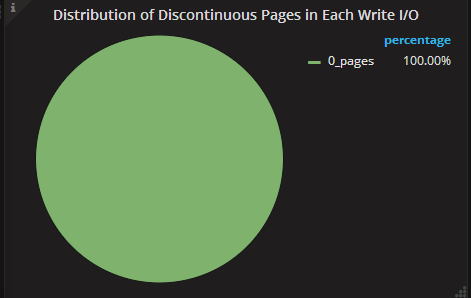
The **Size Distribution of Read Bulk RPC** panel (Figure 28) shows the ratio information of read bulk RPC with different bulk I/O size in the Lustre filesystem. As shown in the figure, the total percentage of the read bulk RPC number with 256 pages is 100% where the current used test case is running”dd if=/mnt/lustre/file of=/dev/zero bs=1M”.

Figure 28: Size Distribution of Read Bulk RPC Panel



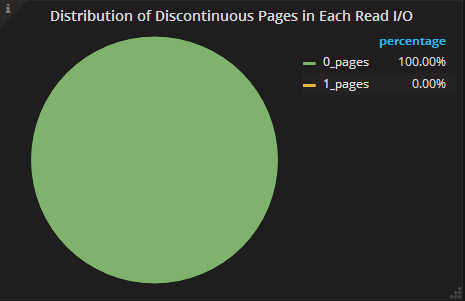
In each Lustre I/O, if the next page to be written or read in the I/O is not with the next offset, that page is a discontinuous page. There could be multiple discontinuous pages in an I/O. I/Os with less discontinuous pages are more friendly to OSTs, and underlying disk system will obtain much better performance. The **Distribution of Discontinuous Pages in Each Write I/O** panel (Figure 29) shows the ratio information of the discontinuous pages in each write I/O in the Lustre filesystem. As shown in the figure, the total percentage of discontinuous pages “0\_pages” is 100%, which means all pages are continuous.

Figure 29: Distribution of Discontinuous Pages in Each Write I/O Panel



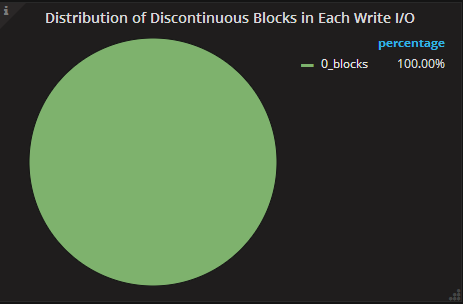
The **Distribution of Discontinuous Pages in Each Read I/O** panel (Figure 30) shows the ratio information of discontinuous pages in each read I/O in the Lustre filesystem. As shown in the figure, the percentage of discontinuous pages “0\_pages” in each read I/O is 100%, which means all pages are continuous.

Figure 30: Distribution of Discontinuous Pages in Each Read I/O Panel



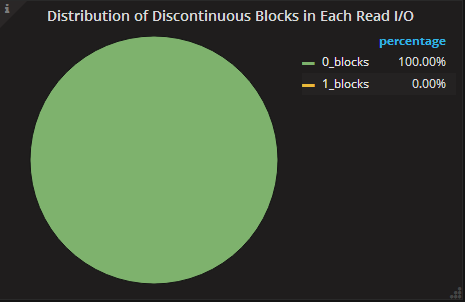
The **Distribution of Discontinuous Blocks** panel (Figure 31) shows the ratio information of the discontinuous blocks in each write I/O in the Lustre filesystem. In each Lustre read/write I/O, the meaning of discontinuous blocks is similar to discontinuous pages. How many pages a block contains is determined by the underlying filesystem (ldiskfs).If an I/O has discontinuous blocks, there must exist discontinuous pages, but the opposite is not necessarily true. As shown in the figure, the percentage of write discontinuous blocks “0\_blocks” in each write I/O is 100%, which means nearly all write I/O are continuous.

Figure 31: Distribution of Discontinuous Blocks in Each Write I/O Panel



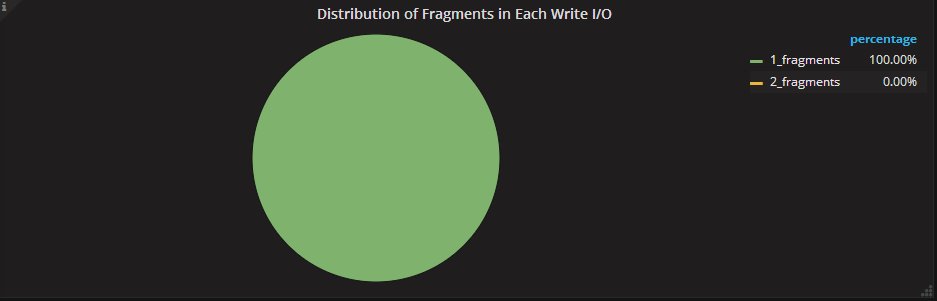
The **Distribution of Discontinuous Blocks in Each Read I/O** panel (Figure 32) shows the ratio information of discontinuous blocks in each read I/O in the Lustre filesystem. As shown in the figure, the percentage of discontinuous blocks “0\_blocks” in each read I/O is 100%, and it means that none of the read I/Os is discontinous.

Figure 32: Distribution of Discontinuous Blocks in Each Read I/O Panel



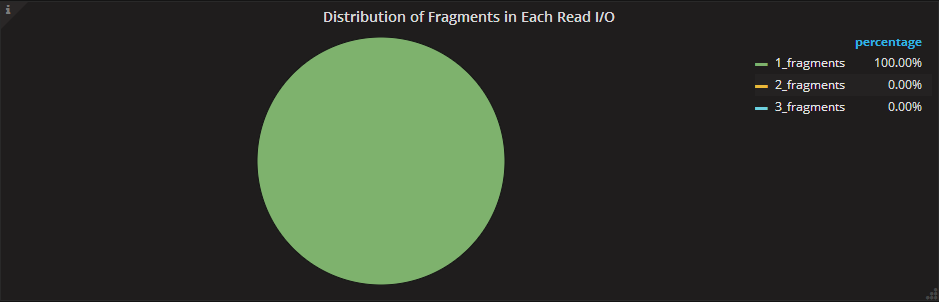
For various reasons (e.g. too many pages to read or write per single I/O), read or write I/O sent by Lustre OSD to the underlying disk system may be split into multiple disk I/Os. The **Distribution of Fragments in Each Write I/O** panel (Figure 33) shows the distribution of write I/Os by the number of disk I/Os each write I/O is split into. As shown in the figure, “1\_fragments” denotes that I/O is not split. The percentage of “1\_fragments” is 100%, which means that none of the write I/O is split and all of them are continuous. “2\_fragments” denotes that Lustre write I/O is split into two disk block I/Os, and the percentage in the figure is 0%.

Figure 33: Distribution of Fragments in Each Write I/O Panel



The **Distribution of Fragments in Each Read I/O** panel (Figure 34) shows the distribution of read I/Os by the number of disk I/Os each read I/O is split into. In the figure, the percentage of “1\_fragments” is 100%, which means that none of the read I/Os is split and all of them are continuous. “2\_fragments” denotes that Lustre read I/O is split into two disk block I/Os, and the percentage in the figure is 0%.

Figure 34: Distribution of Fragments in Each Read I/O Panel



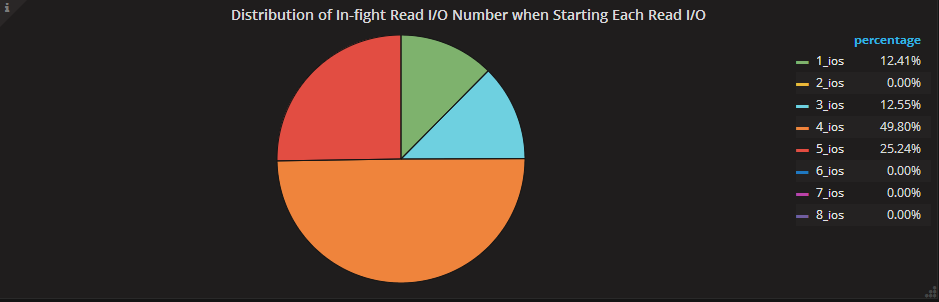
The **Distribution of in-flight Write I/O Number when Starting Each Write I/O** panel (Figure 35) shows the distribution of the number of write I/Os operations pending at the time of starting each write I/O in the Lustre filesystem. In the figure, ”1\_ios” has percentage of 100%. That means, when the write I/O operations started on the OST, this I/O was the only one write I/O that is currently being submitted to disk.

Figure 35: Distribution of in-flight Write I/O Number when Starting Each Write I/O Panel



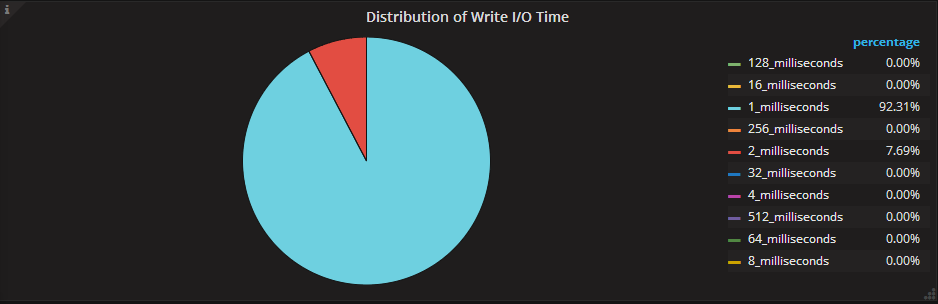
The **Distribution of in-flight Read I/O Number when Starting Each Read I/O** panel (Figure 36) shows the distribution of the number of read I/Os operations pending at the time of starting each read I/O in the Lustre filesystem. For example, “4\_ios” has percentage of 49.80% in the figure. That means 49.80% of the read I/O operations started when there were four in-flight I/O operations on that OST.

Figure 36: Distribution of in-flight Read I/O Number when Starting Each Read I/O Panel



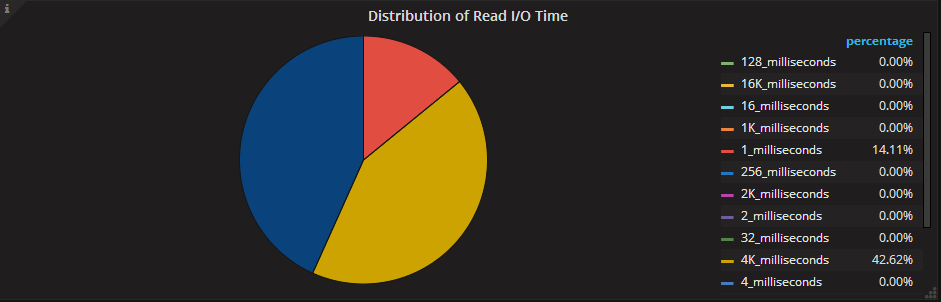
The **Distribution of Write I/O Time** panel (Figure 37) shows the current distribution of OSD write I/O time in the Lustre filesystem. “1\_milliseconds” represents the percentage of I/O operations whose duration is less than 1 millisecond, “2\_milliseconds” represents the percentage of I/O operations whose duration is between 1 millisecond and 2 milliseconds, and so on.

Figure 37: Distribution of Write I/O Time Panel



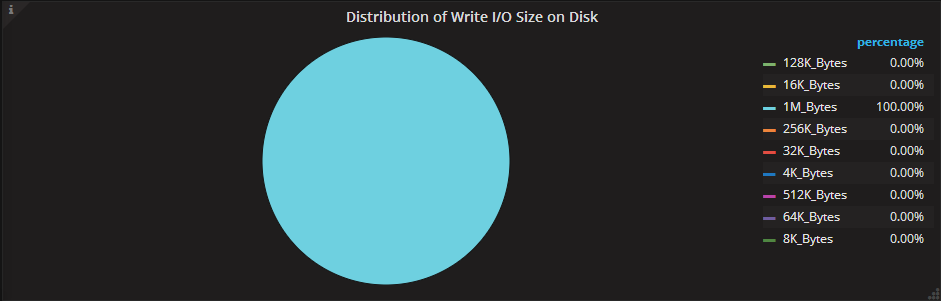
The **Distribution of Read I/O Time** panel (Figure 38) shows the current distribution of OSD write I/O size in the Lustre filesystem. In the figure, the percentage of “1\_milliseconds” I/Os (I/Os whose duration is less than 1 millisecond) is 14.11%, “4K\_milliseconds” I/Os (I/Os whose duration is between 2K milliseconds and 4K milliseconds) take up 42.62%.

Figure 38: Distribution of Read I/O Time Panel



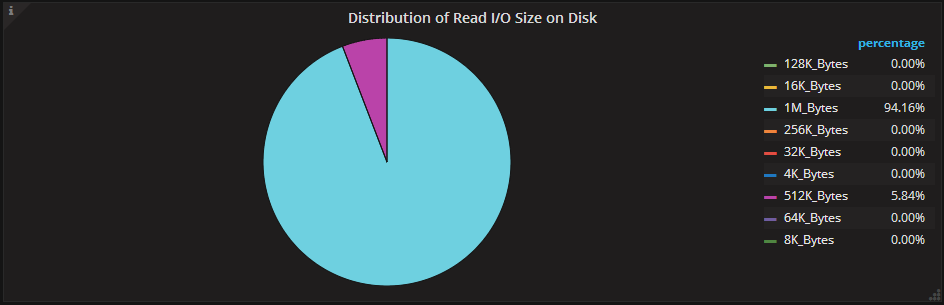
The **Distribution of Write I/O size on Disk** panel (Figure 39) shows the current distribution of OSD write I/O size in the Lustre filesystem. In the panel, “1M\_Bytes” represents disk I/Os that have sizes between 512K and 1M bytes, “512K\_Bytes” represents I/Os with disk I/O size between 256K and 512K bytes, etc.

Figure 39: Distribution of Write I/O size on Disk Panel



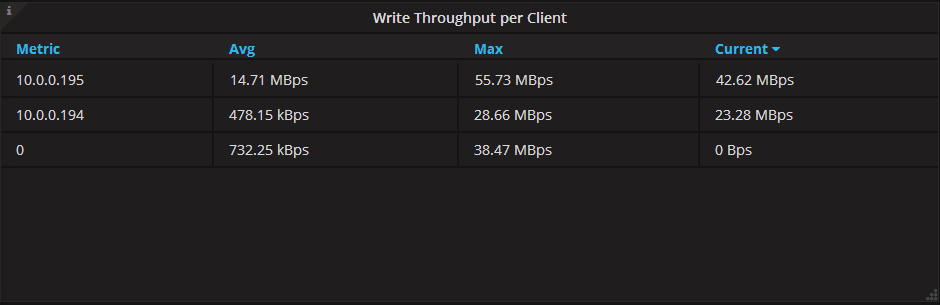
The **Distribution of Read I/O Size on Disk** panel (Figure 40) shows the distribution of OSD read I/O size in the Lustre filesystem. In the panel, “1M\_Bytes” represents I/Os with disk I/O size between 512K and 1M bytes, “512K\_Bytes” represents I/Os with disk I/O size between 256K and 512K bytes, etc. In the figure, the percentage of “1M\_Bytes” I/Os is 94.16% and the percentage of “512K\_Bytes” I/Os is 5.84%.

Figure 40: Distribution of Read I/O Size on Disk Panel



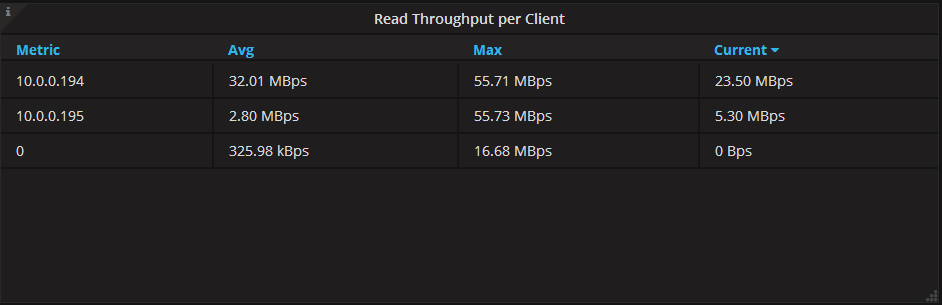
The **Write Throughput per Client** panel (Figure 41) shows the average, max, and current write throughput per client in the Lustre filesystem. As shown in the figure, the average/max/current values of the write throughput for the client with the IP address 10.0.0.195 are 14.71MBps/55.73MBps/42.62MBps, respectively.

Figure 41: Write Throughput per Client Panel



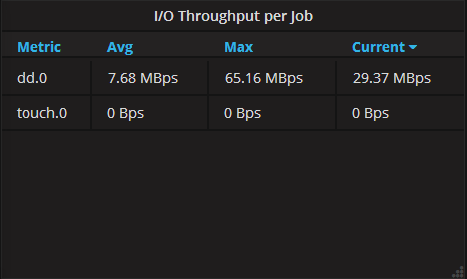
The **Read Throughput per Client** panel (Figure 42) shows the metric information of the read throughput per client in the Lustre filesystem. It includes average, max, and current values. As shown in the figure, the average, max, and current values of the read throughput for the client with the IP address 10.0.0.194 are 32.01MBps/55.71MBps/23.50MBps.

Figure 42: Read Throughput per Client Panel



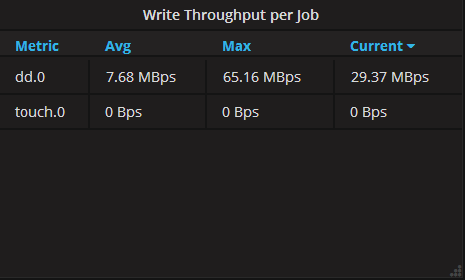
The **I/O Throughput per Job** panel (Figure 43) shows the metric information of the I/O throughput per job in the Lustre filesystem. It includes average, max, and current values. As shown in the figure, for the job with JOBID “dd.0”, the average I/O throughput is 7.68MBps, the max value is 65.16MBps, and the current I/O throughput is 29.37MBps.

Figure 43: I/O Throughput per Job Panel



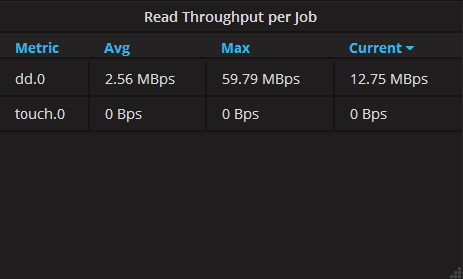
The **Write Throughput per Job** panel (Figure 44) shows the metric information of the write throughput per job in the Lustre filesystem. It includes average, max, and current values. As shown in the figure, for the job with JOBID “dd.0”, the average I/O throughput is 7.68MBps, the max value is 64.16MBps, and the current I/O throughput is 29.37MBps.

Figure 44: Write Throughput per Job Panel



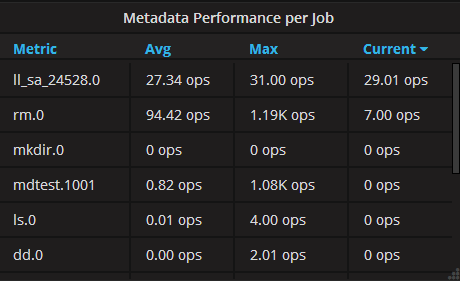
The **Read Throughput per Job** panel (Figure 45) shows the metric information of the read throughput per job in the Lustre filesystem. It includes average, max, and current values. As shown in the figure, for the job with JOBID “dd.0”, the average I/O throughput is 2.56MBps, the max value is 59.79MBps, and the current I/O throughput is 12.75MBps.

Figure 45: Read Throughput per Job Panel



The **Metadata Performance per Job** panel (Figure 46) shows the metric information of the metadata performance per job in the Lustre filesystem. It includes average, max, and current values, and the unit is OPS (Operations per Second). As shown in the figure, for the job with JOBID “rm.0”, the average metadata performance is 94.42 ops, max value is 1.19K ops, and the current performance is 7.00 ops.

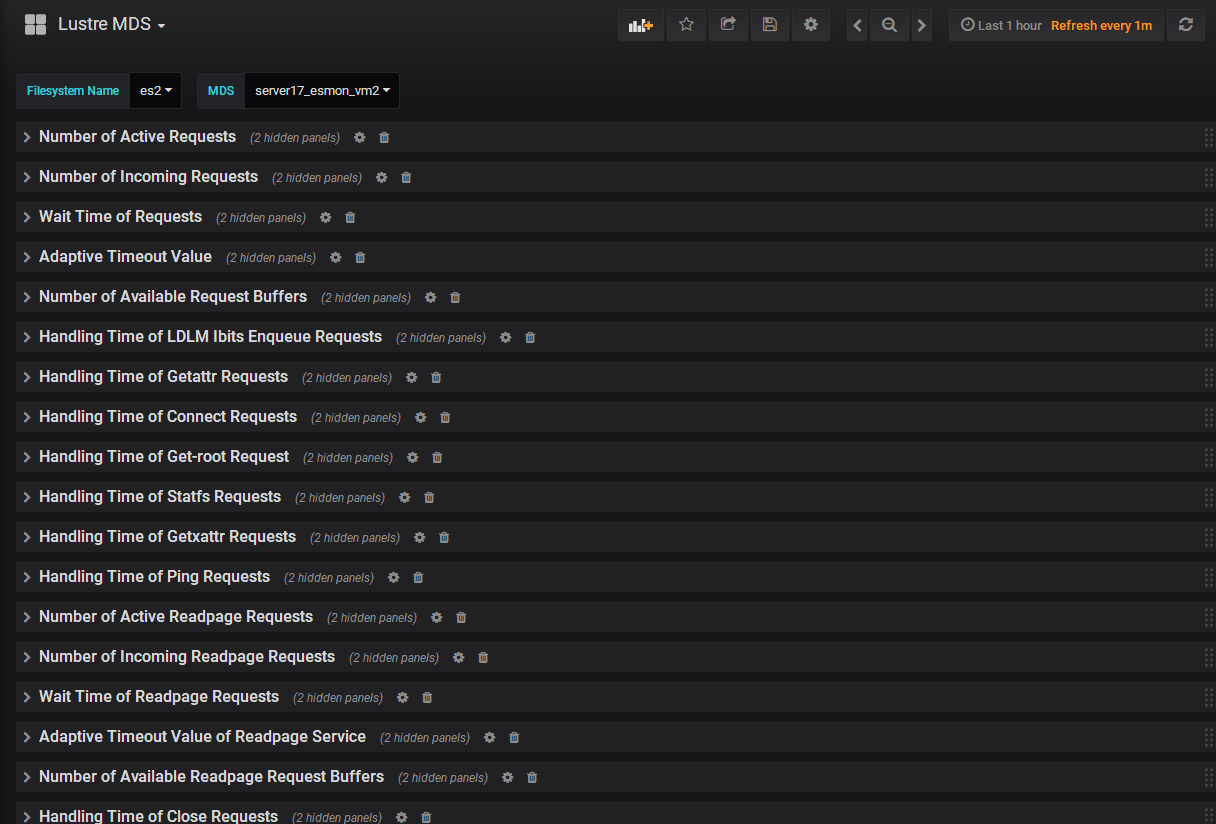
Figure 46: Metadata Performance per Job Panel



## Lustre MDS Statistics

The Lustre MDS Statisticsdashboard (Figure 47) shows detailed information about a Lustre MDS server.

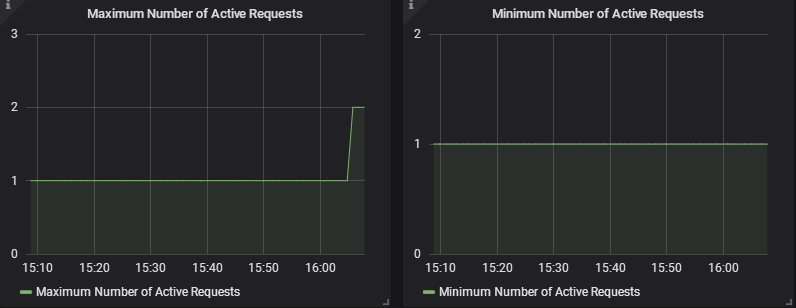
Figure 47: Lustre MDS Statistics Dashboard



Below you will find description of some of the panels in the **Lustre MDS Statistics** dashboard:

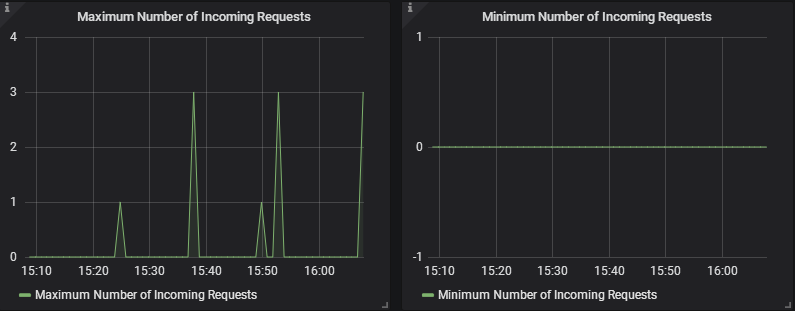
The **Number of Active Requests** Panel (Figure 48) shows the maximum and minimum number of active requests varying on time on MDS. Active requests are the requests that is being actively handled by this MDS, not including the requests that are waiting in the queue. If the number of active requests is smaller than PTLRPC thread number minus two (one for incoming request handling and the other for incoming high priority request handling), it generally means the thread number should be enough. The value shown in the left graph blew is the maximum number during the last collect interval. The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 48：Number of Active Requests Panel



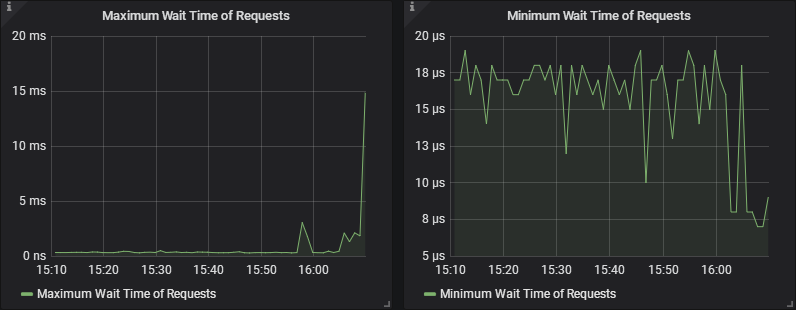
The Number of Incoming Requests Panel (Figure 49) shows the maximum and minimum number of incoming requests varying on time on MDS. Incoming requests are the requests that waiting on preprocessing. A request is not incoming request any more when its proprocessing begins. And after preprocessing, the requests will be put into processing queue. The value shown in the left graph blew is the maximum number of incoming requests during the last collect interval; The value shown in the right graph blew is the minimum number of incoming requests during the last collect interval.

Figure 49：Number of Incoming Requests Panel



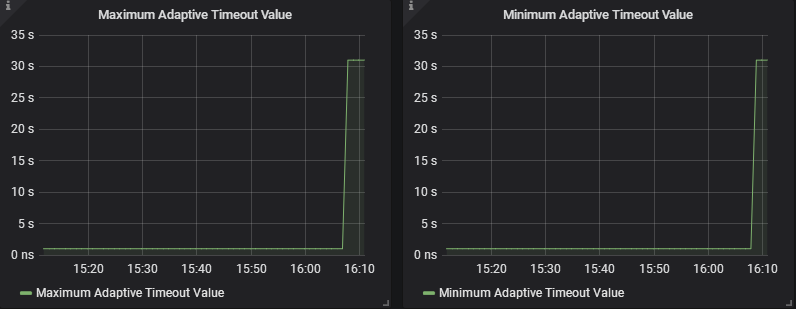
The Wait time of Requests Panel (Figure 50) shows the maximum and minimum wait time of requests varying on time on MDS. The wait time of a request is the time interval between its arrival time and the time when it starts to be handled. The value shown in the left graph blew is the maximum wait time of the requests during the last collect interval; The value shown in the right graph blew is the minimum wait time of the requests during the last collect interval.

Figure 50：Wait time of Requests Panel



The Adaptive Timeout Value Panel (Figure 51) shows the maximum and minimum adaptive timeout value varying on time on MDS. When a client sends a request, it has a timeout deadline for the reply. The timeout value of a service is an adaptive value negotiated between server and client during run-time. The value shown in the left graph is the maximum timeout of the MDS service during the last collect interval; The value shown in the right graph is the minimum timeout of the MDS service during the last collect interval.

Figure 51：Adaptive Timeout Value Panel



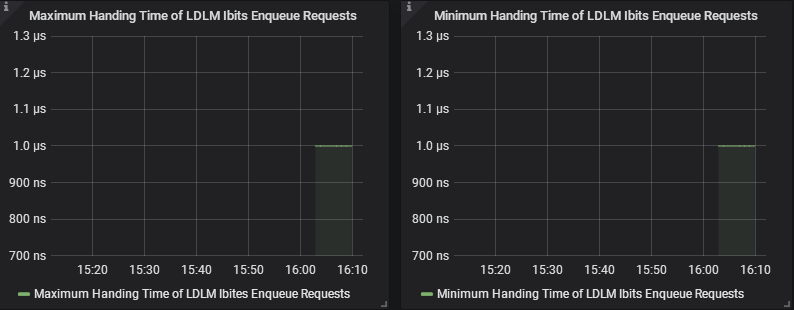
The Number of Available Request buffers Panel (Figure 52) shows the maximum and minimum number of available request buffers varying on time on MDS. When a request arrives, one request buffer will be used. When number of available request buffers is under low water, more buffers are needed to avoid performance bottleneck. The value shown in the left graph blew is the maximum number during the last collect interval; The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 52：Number of Available Request buffers Panel



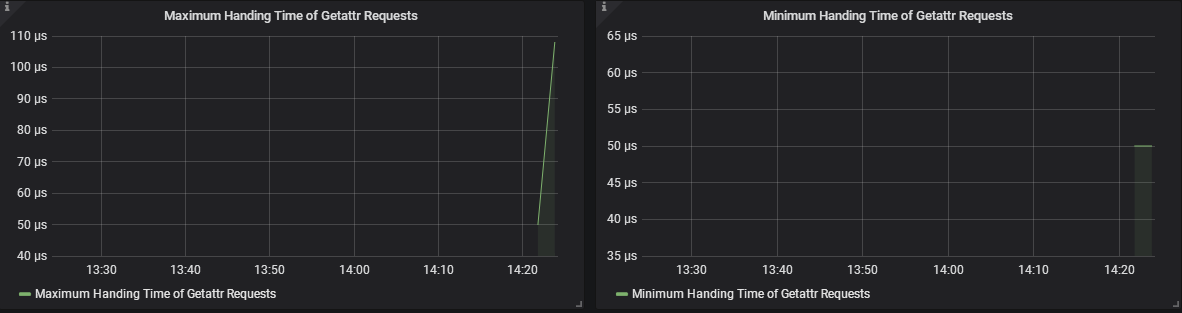
The Handing time of LDLM Ibits Enqueue Requests Panel (Figure 53) shows the maximum and minimum Handling time of LDLM ibits enqueue request varying on time on MDS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the LDLM ibits enqueue requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the LDLM ibits enqueue requests during the last collect interval.

Figure 53：Handing time of LDLM ibits Enqueue Requests Panel



The Handing time of Getattr Requests Panel (Figure 54) shows the maximum and minimum Handling time of Getattr requests varying on time on MDS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Getattr requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Getattr requests during the last collect interval.

Figure 54：Handing Time of Getattr Requests Panel



The Handing time of Connect Requests Panel (Figure 55) shows the maximum and minimum Handling time of Connect requests varying on time on MDS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Connect requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Connect requests during the last collect interval.

Figure 55：Handing Time of Connect Requests Panel



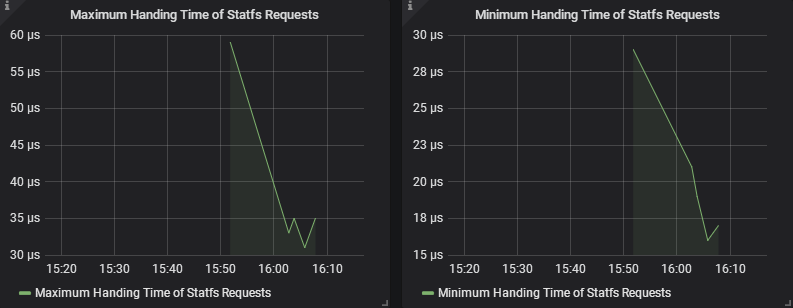
The Handing time of Get-root Requests Panel (Figure 56) shows the maximum and minimum Handling time of Get-root requests varying on time on MDS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Get-root requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Get-root requests during the last collect interval.

Figure 56：Handing Time of Get-root Requests Panel



The Handing time of Statfs Requests Panel (Figure 57) shows the maximum and minimum Handling time of Statfs requests varying on time on MDS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Statfs requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Statfs requests during the last collect interval.

Figure 57：Handing Time of Statfs Requests Panel



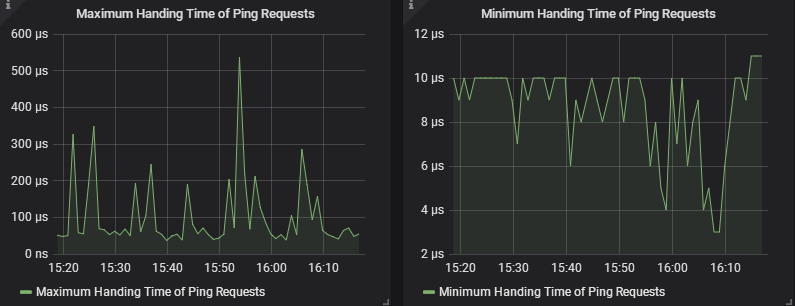
The Handing time of Getxattr Requests Panel (Figure 58) shows the maximum and minimum Handling time of Getxattr requests varying on time on MDS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Getxattr requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Getxattr requests during the last collect interval.

Figure 58：Handing Time of Getxattr Requests Panel



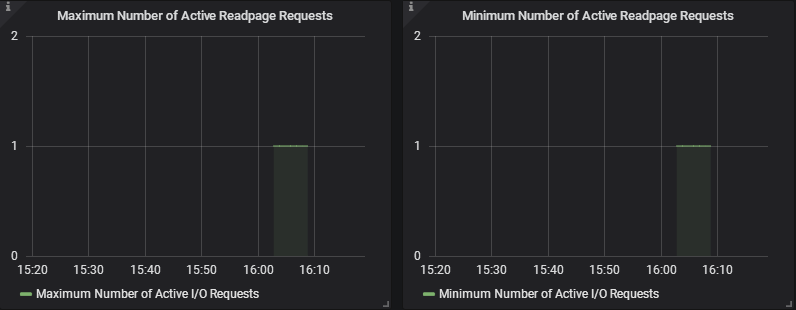
The Handing time of Ping Requests Panel (Figure 59)shows the maximum and minimum Handling time of Ping requests varying on time on MDS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Ping requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Ping requests during the last collect interval.

Figure 59：Handing Time of Ping Requests Panel



The **Number of Active Readpage Requests** Panel (Figure 48) shows the maximum and minimum number of active Readpage requests varying on time on MDS. Active requests are the requests that is being actively handled by this MDS, not including the requests that are waiting in the queue. If the number of active requests is smaller than PTLRPC thread number minus two (one for incoming request handling and the other for incoming high priority request handling), it generally means the thread number should be enough. The value shown in the left graph blew is the maximum number during the last collect interval. The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 60：Number of Active Readpage Requests Panel



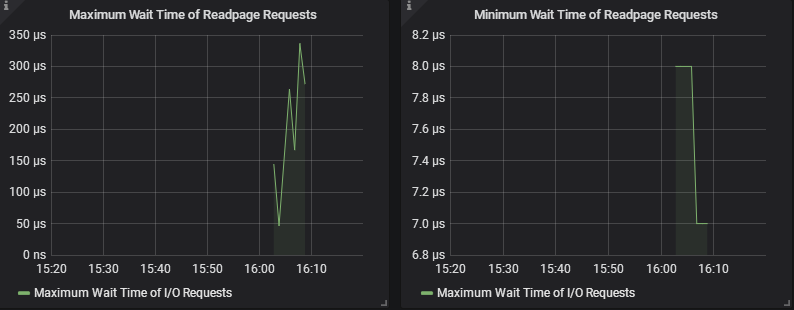
The Number of Incoming Readpage Requests Panel (Figure 61) shows the maximum and minimum number of incoming Readpage requests varying on time on MDS. Incoming requests are the requests that waiting on preprocessing. A request is not incoming request any more when its proprocessing begins. And after preprocessing, the requests will be put into processing queue. The value shown in the left graph blew is the maximum number of incoming Readpage requests during the last collect interval; The value shown in the right graph blew is the minimum number of incoming Readpage requests during the last collect interval.

Figure 61：Number of Incoming Readpage Requests Panel



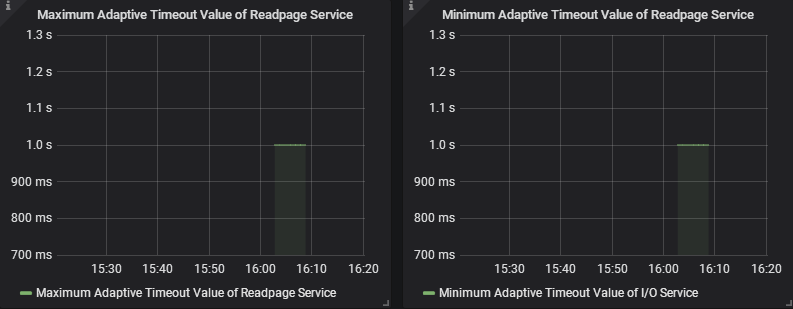
The Wait time of Readpage Requests Panel (Figure 62) shows the maximum and minimum wait time of Readpage requests varying on time on MDS. The wait time of a request is the time interval between its arrival time and the time when it starts to be handled. The value shown in the left graph blew is the maximum wait time of the Readpage requests during the last collect interval; The value shown in the right graph blew is the minimum wait time of the Readpage requests during the last collect interval.

Figure 62：Wait Time of Readpage Requests Panel



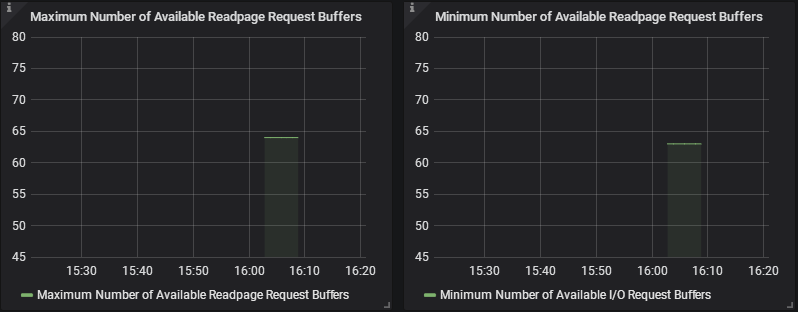
The Adaptive Timeout Value of Readpage Service Panel (Figure 63) shows the maximum and minimum adaptive timeout value of Readpage Service varying on time on MDS. When a client sends a request, it has a timeout deadline for the reply. The timeout value of a service is an adaptive value negotiated between server and client during run-time. The value shown in the left graph is the maximum timeout of the Readpage service during the last collect interval; The value shown in the right graph is the minimum timeout of the Readpage service during the last collect interval.

Figure 63：Adaptive Timeout Value of Readpage Service



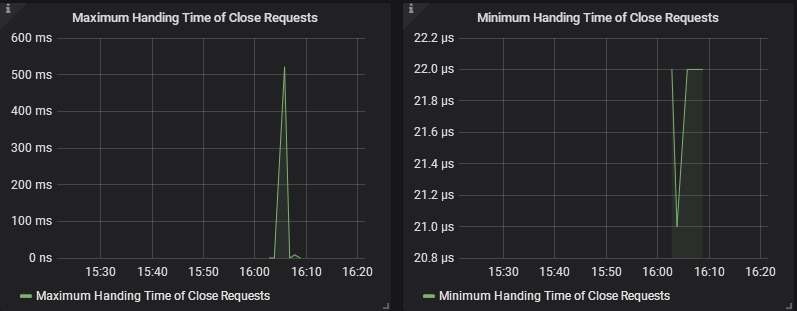
The Number of Available Readpage Request buffers Panel (Figure 52) shows the maximum and minimum number of available Readpage request buffers varying on time on MDS. When a request arrives, one request buffer will be used. When number of available request buffers is under low water, more buffers are needed to avoid performance bottleneck. The value shown in the left graph blew is the maximum number during the last collect interval; The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 64：Number of Available Readpage Request Buffers Panel



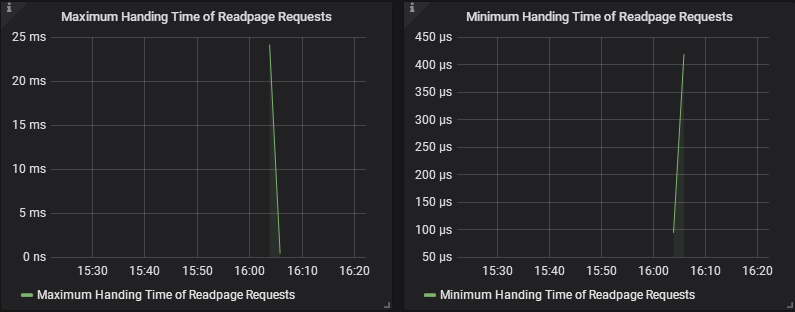
The Handing time of Close Requests Panel (Figure 65) shows the maximum and minimum Handling time of Close requests varying on time on MDS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Close requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Close requests during the last collect interval.

Figure 65：Handing Time of Close Requests Panel



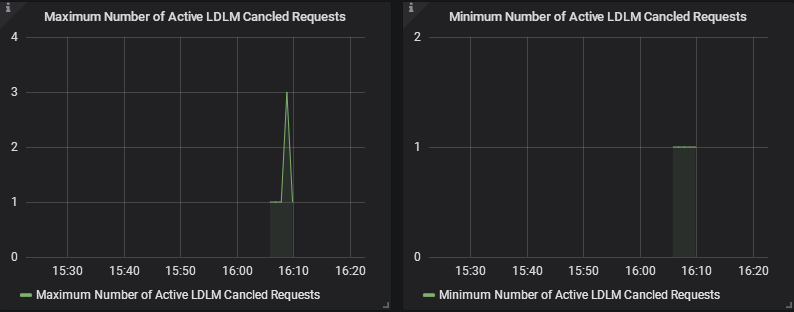
The Handing time of Readpage Requests Panel (Figure 66) shows the maximum and minimum Handling time of Readpage requests varying on time on MDS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Readpage requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Readpage requests during the last collect interval.

Figure 66：Handing Time of Readpage Requests Panel



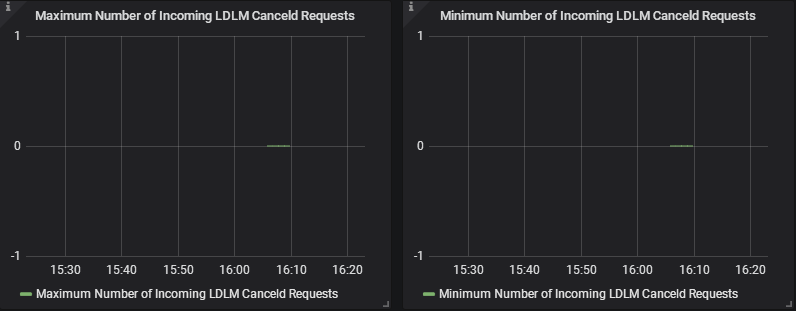
The **Number of Active LDLM Canceld** **Requests** Panel (Figure 67) shows the maximum and minimum number of active LDLM Canceld requests varying on time on MDS. Active requests are the requests that is being actively handled by this MDS, not including the requests that are waiting in the queue. If the number of active requests is smaller than PTLRPC thread number minus two (one for incoming request handling and the other for incoming high priority request handling), it generally means the thread number should be enough. The value shown in the left graph blew is the maximum number during the last collect interval. The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 67：Number of Active LDLM Canceld Requests Panel



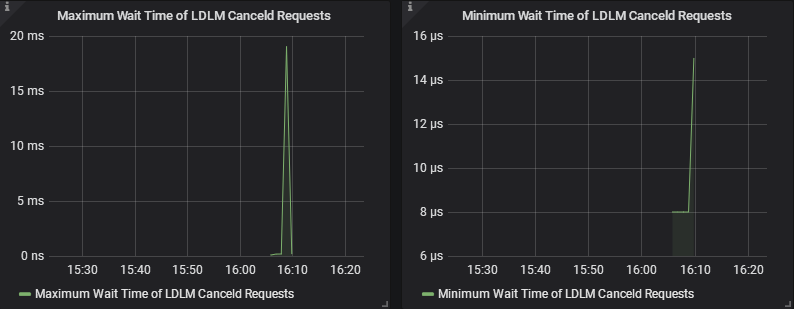
The Number of Incoming LDLM Canceld Requests Panel (Figure 68) shows the maximum and minimum number of incoming LDLM Canceld requests varying on time on MDS. Incoming requests are the requests that waiting on preprocessing. A request is not incoming request any more when its proprocessing begins. And after preprocessing, the requests will be put into processing queue. The value shown in the left graph blew is the maximum number of incoming LDLM Canceld requests during the last collect interval; The value shown in the right graph blew is the minimum number of incoming LDLM Canceld requests during the last collect interval.

Figure 68：Number of Incoming LDLM Canceld Requests Panel



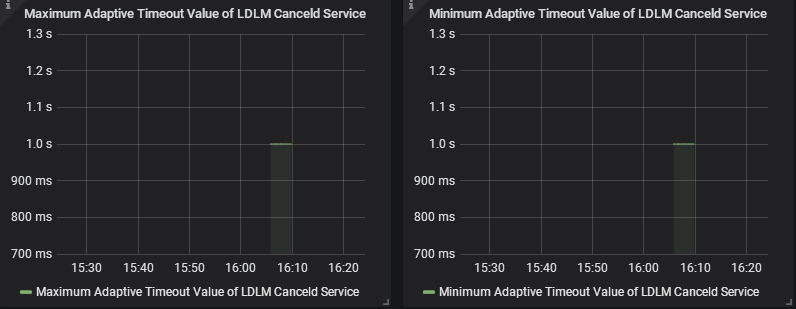
The Wait time of LDLM Canceld Requests Panel (Figure 69) shows the maximum and minimum wait time of LDLM Canceld requests varying on time on MDS. The wait time of a request is the time interval between its arrival time and the time when it starts to be handled. The value shown in the left graph blew is the maximum wait time of the LDLM Canceld requests during the last collect interval; The value shown in the right graph blew is the minimum wait time of the LDLM Canceld requests during the last collect interval.

Figure 69：Wait Time of LDLM Canceld Requests Panel



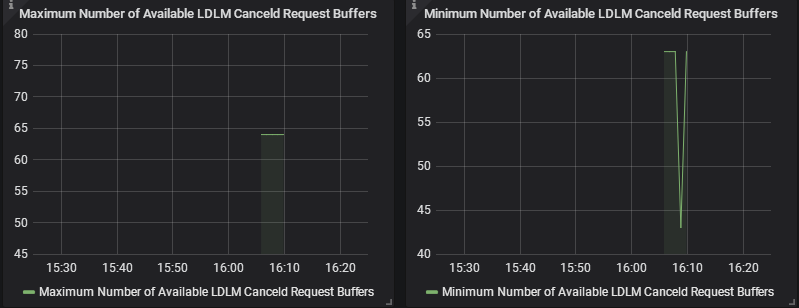
The Adaptive Timeout Value of LDLM Canceld Service Panel (Figure 70) shows the maximum and minimum adaptive timeout value of LDLM Canceld Service varying on time on MDS. When a client sends a request, it has a timeout deadline for the reply. The timeout value of a service is an adaptive value negotiated between server and client during run-time. The value shown in the left graph is the maximum timeout of the LDLM Canceld service during the last collect interval; The value shown in the right graph is the minimum timeout of the LDLM Canceld service during the last collect interval.

Figure 70：Adaptive Timeout Value of LDLM Canceld Service



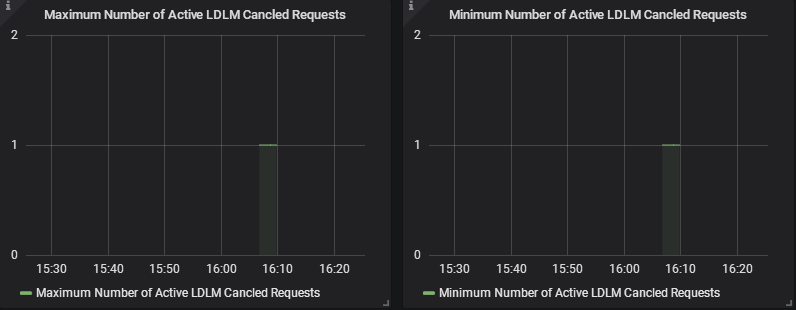
The Number of Available LDLM Canceld Request buffers Panel (Figure 71) shows the maximum and minimum number of available LDLM Canceld request buffers varying on time on MDS. When a request arrives, one request buffer will be used. When number of available request buffers is under low water, more buffers are needed to avoid performance bottleneck. The value shown in the left graph blew is the maximum number during the last collect interval; The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 71：Number of Available LDLM Canceld Request Buffers Panel



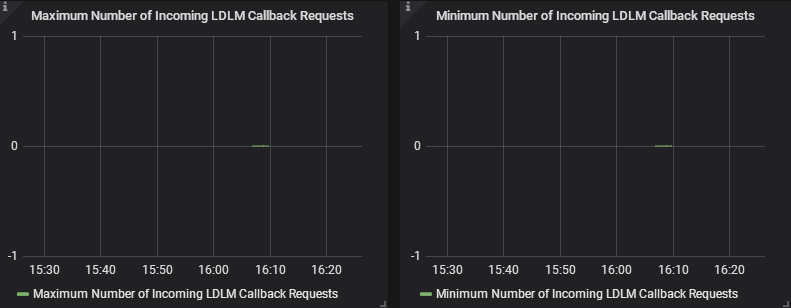
The **Number of Active LDLM Callback** **Requests** Panel (Figure 72) shows the maximum and minimum number of active LDLM Callback requests varying on time on MDS. Active requests are the requests that is being actively handled by this MDS, not including the requests that are waiting in the queue. If the number of active requests is smaller than PTLRPC thread number minus two (one for incoming request handling and the other for incoming high priority request handling), it generally means the thread number should be enough. The value shown in the left graph blew is the maximum number during the last collect interval. The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 72：Number of Active LDLM Callback Requests Panel



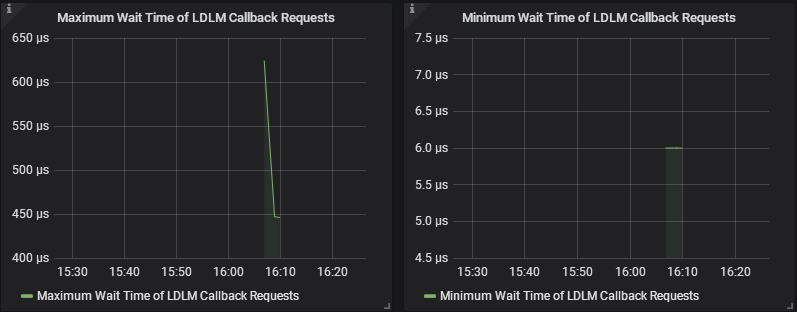
The Number of Incoming LDLM Callback Requests Panel (Figure 73) shows the maximum and minimum number of incoming LDLM Callback requests varying on time on MDS. Incoming requests are the requests that waiting on preprocessing. A request is not incoming request any more when its proprocessing begins. And after preprocessing, the requests will be put into processing queue. The value shown in the left graph blew is the maximum number of incoming LDLM Callback requests during the last collect interval; The value shown in the right graph blew is the minimum number of incoming LDLM Callback requests during the last collect interval.

Figure 73：Number of Incoming LDLM Callback Requests Panel



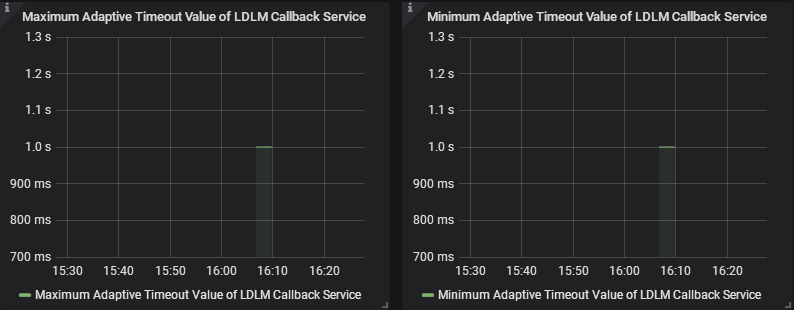
The Wait time of LDLM Callback Requests Panel (Figure 74) shows the maximum and minimum wait time of LDLM Callback requests varying on time on MDS. The wait time of a request is the time interval between its arrival time and the time when it starts to be handled. The value shown in the left graph blew is the maximum wait time of the LDLM Callback requests during the last collect interval; The value shown in the right graph blew is the minimum wait time of the LDLM Callback requests during the last collect interval.

Figure 74：Wait time of LDLM Callback Requests Panel



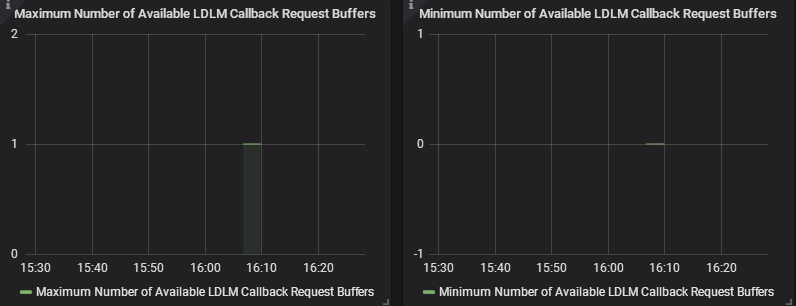
The Adaptive Timeout Value of LDLM Callback Service Panel (Figure 75) shows the maximum and minimum adaptive timeout value of LDLM Callback Service varying on time on MDS. When a client sends a request, it has a timeout deadline for the reply. The timeout value of a service is an adaptive value negotiated between server and client during run-time. The value shown in the left graph is the maximum timeout of the LDLM Callback service during the last collect interval; The value shown in the right graph is the minimum timeout of the LDLM Callback service during the last collect interval.

Figure 75：Adaptive Timeout Value of LDLM Callback Service Panel



The Number of Available LDLM Callback Request buffers Panel (Figure 76) shows the maximum and minimum number of available LDLM Callback request buffers varying on time on MDS. When a request arrives, one request buffer will be used. When number of available request buffers is under low water, more buffers are needed to avoid performance bottleneck. The value shown in the left graph blew is the maximum number during the last collect interval; The value shown in the right graph blew is the minimum number during the last collect interval.

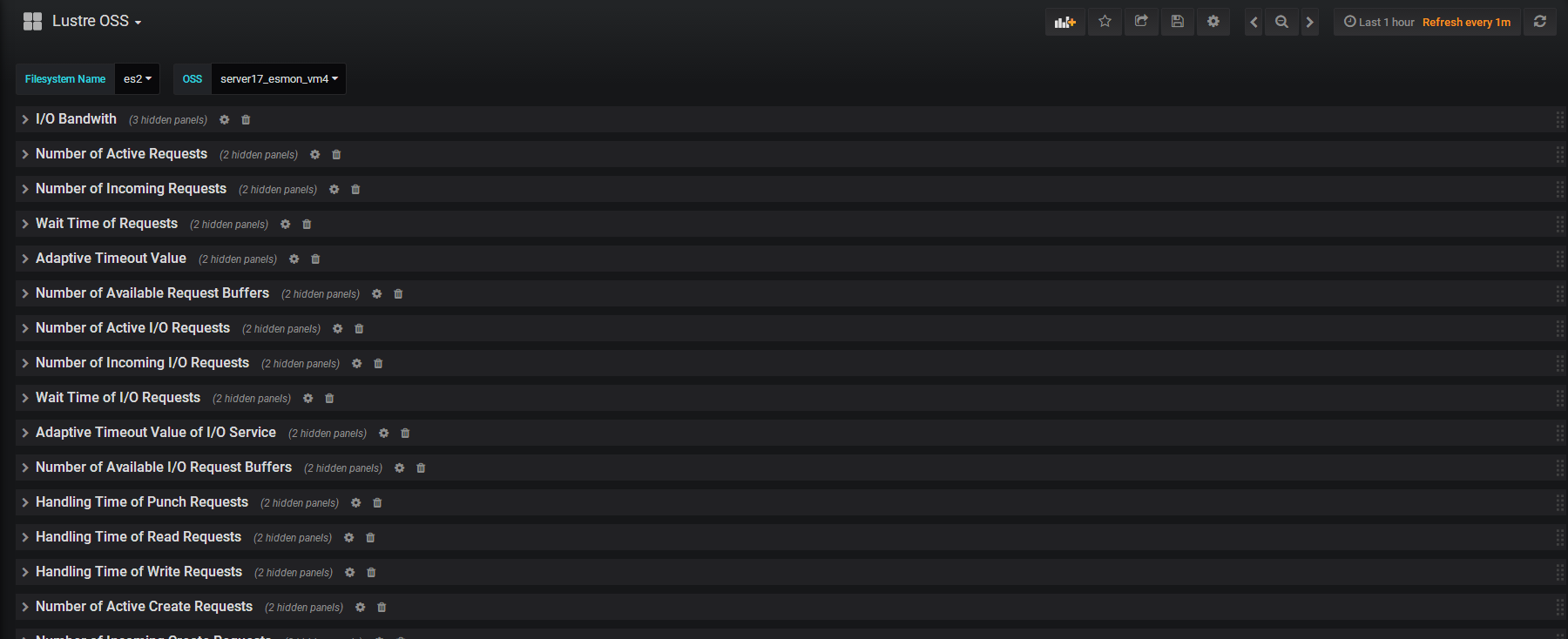
Figure 76：Number of Available LDLM Callback Request Buffers Panel



## Lustre OSS Statistics

The Lustre OSS dashboard (Figure 77) shows detailed information about a Lustre OSS server.

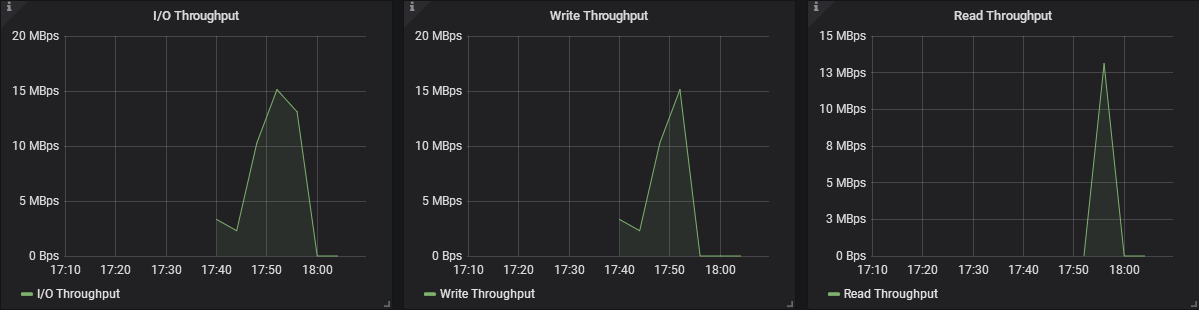
Figure 77: Lustre OSS Dashboard



Below you will find description of some of the panels in the **Lustre OSS Statistics** dashboard:

I/O Bandwidth Panel (Figure 78) shows the I/O throughput, write throughput and read throughput of an OSS server, respectively.

Figure 78：I/O Bandwidth Panel



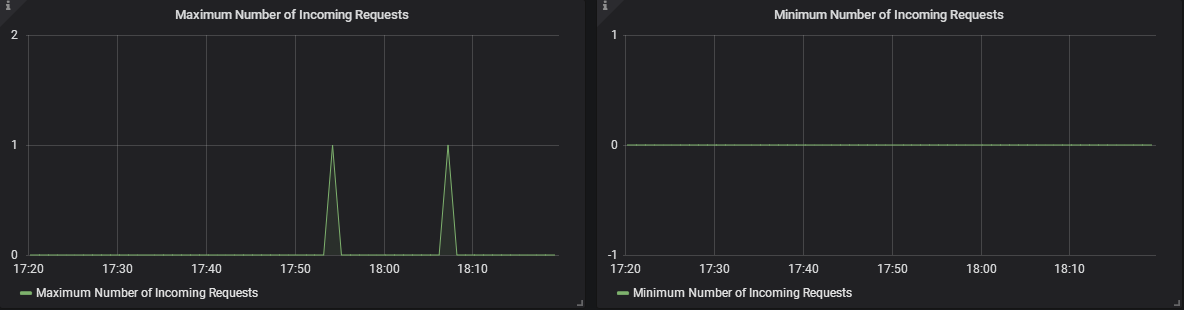
The **Number of Active Requests** Panel (Figure 79) shows the maximum and minimum number of active requests varying on time on OSS. Active requests are the requests that is being actively handled by this OSS, not including the requests that are waiting in the queue. If the number of active requests is smaller than PTLRPC thread number minus two (one for incoming request handling and the other for incoming high priority request handling), it generally means the thread number should be enough. The value shown in the left graph blew is the maximum number during the last collect interval. The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 79: Number of Active Requests Panel



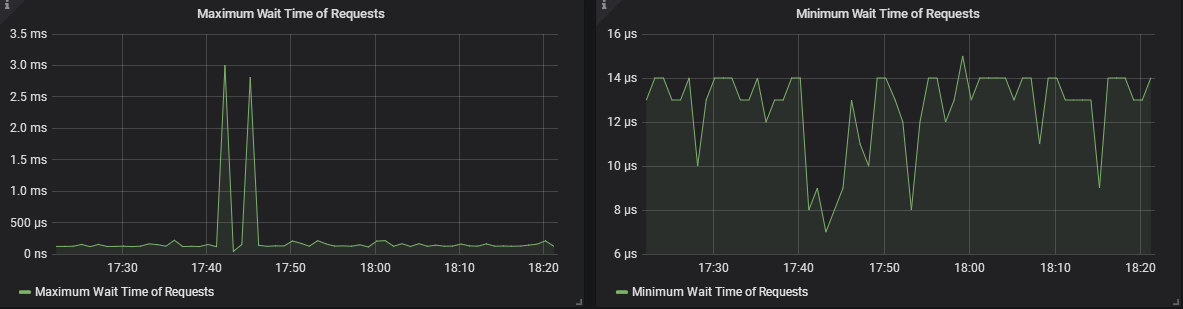
The Number of Incoming Requests Panel (Figure 80) shows the maximum and minimum number of incoming requests varying on time on OSS. Incoming requests are the requests that waiting on preprocessing. A request is not incoming request any more when its proprocessing begins. And after preprocessing, the requests will be put into processing queue. The value shown in the left graph blew is the maximum number of incoming requests during the last collect interval; The value shown in the right graph blew is the minimum number of incoming requests during the last collect interval.

Figure 80: Number of Incoming Requests Panel



The Wait time of Requests Panel (Figure 81) shows the maximum and minimum wait time of requests varying on time on OSS. The wait time of a request is the time interval between its arrival time and the time when it starts to be handled. The value shown in the left graph blew is the maximum wait time of the requests during the last collect interval; The value shown in the right graph blew is the minimum wait time of the requests during the last collect interval.

Figure 81: Wait time of Requests Panel



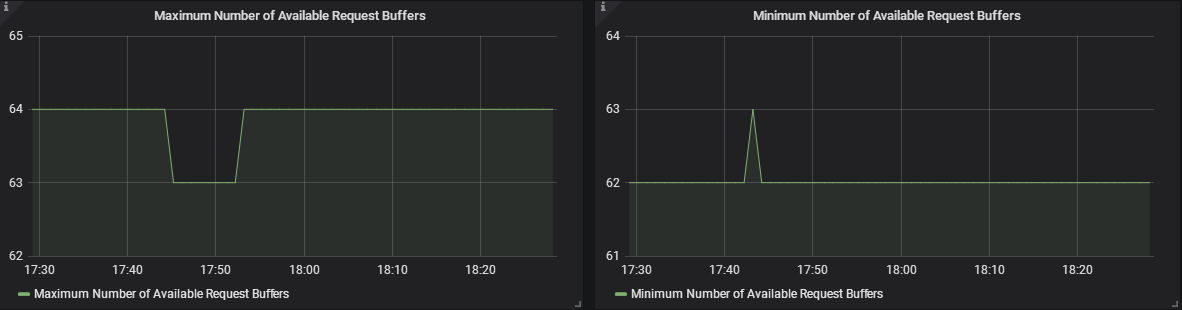
The Adaptive Timeout Value Panel (Figure 82) shows the maximum and minimum adaptive timeout value varying on time on OSS. When a client sends a request, it has a timeout deadline for the reply. The timeout value of a service is an adaptive value negotiated between server and client during run-time. The value shown in the left graph is the maximum timeout of the MDS service during the last collect interval; The value shown in the right graph is the minimum timeout of the MDS service during the last collect interval.

Figure 82: Adaptive Timeout Value Panel



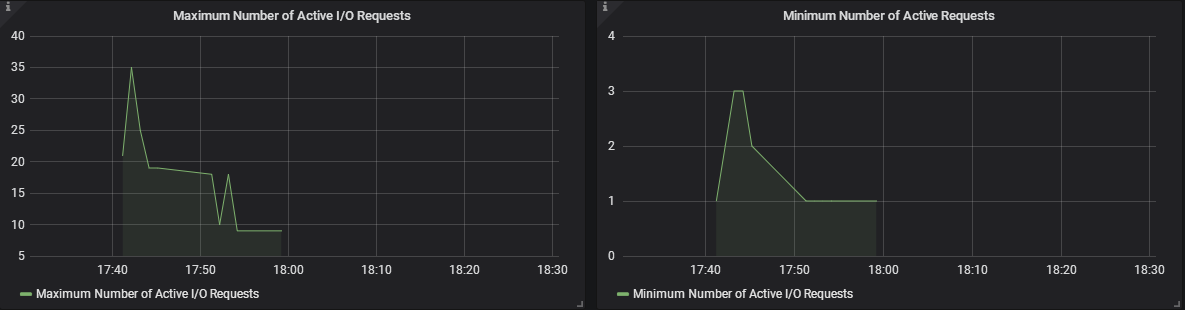
The Number of Available Request buffers Panel (Figure 83) shows the maximum and minimum number of available request buffers varying on time on OSS. When a request arrives, one request buffer will be used. When number of available request buffers is under low water, more buffers are needed to avoid performance bottleneck. The value shown in the left graph blew is the maximum number during the last collect interval; The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 83: Number of Available Request Buffers Panel



The **Number of Active I/O Requests** Panel (Figure 84) shows the maximum and minimum number of active I/O requests varying on time on OSS. Active requests are the requests that is being actively handled by this OSS, not including the requests that are waiting in the queue. If the number of active requests is smaller than PTLRPC thread number minus two (one for incoming request handling and the other for incoming high priority request handling), it generally means the thread number should be enough. The value shown in the left graph blew is the maximum number during the last collect interval. The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 84: Number of Active I/O Requests Panel



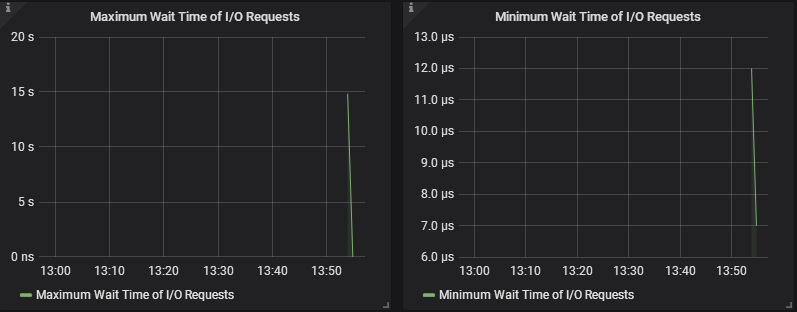
The Number of Incoming I/O Requests Panel (Figure 85) shows the maximum and minimum number of incoming I/O requests varying on time on OSS. Incoming requests are the requests that waiting on preprocessing. A request is not incoming request any more when its proprocessing begins. And after preprocessing, the requests will be put into processing queue. The value shown in the left graph blew is the maximum number of incoming I/O requests during the last collect interval; The value shown in the right graph blew is the minimum number of incoming I/O requests during the last collect interval.

Figure 85: Number of Incoming I/O Requests Panel



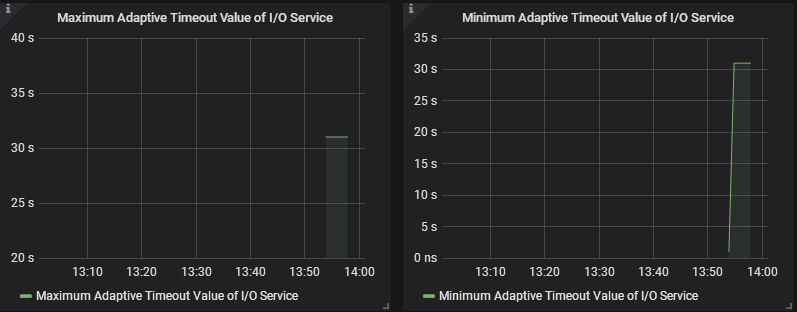
The Wait time of I/O Requests Panel (Figure 86) shows the maximum and minimum wait time of I/O requests varying on time on OSS. The wait time of a request is the time interval between its arrival time and the time when it starts to be handled. The value shown in the left graph blew is the maximum wait time of the I/O requests during the last collect interval; The value shown in the right graph blew is the minimum wait time of the I/O requests during the last collect interval.

Figure 86: Wait Time of I/O requests Panel



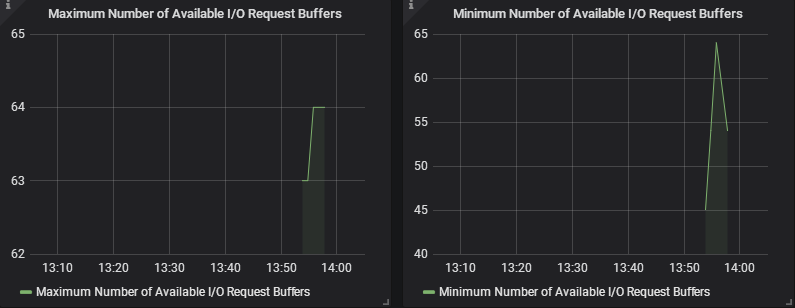
The Adaptive Timeout Value of I/O Service Panel (Figure 87) shows the maximum and minimum adaptive timeout value of I/O Service varying on time on OSS. When a client sends a request, it has a timeout deadline for the reply. The timeout value of a service is an adaptive value negotiated between server and client during run-time. The value shown in the left graph is the maximum timeout of the I/O service during the last collect interval; The value shown in the right graph is the minimum timeout of the I/O service during the last collect interval.

Figure 87: Adaptive Timeout Value of I/O Service Panel



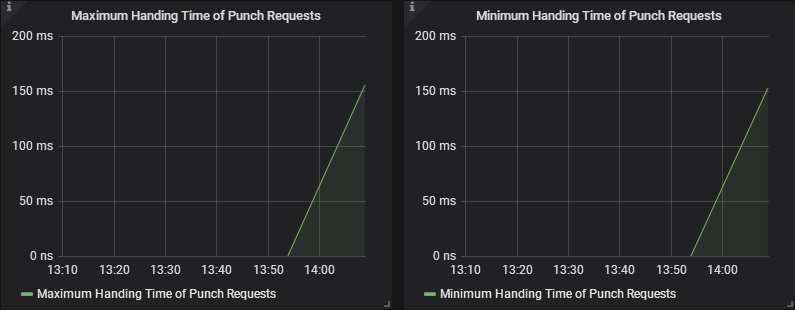
The Number of Available I/O Request buffers Panel (Figure 88) shows the maximum and minimum number of available I/O request buffers varying on time on OSS. When a request arrives, one request buffer will be used. When number of available request buffers is under low water, more buffers are needed to avoid performance bottleneck. The value shown in the left graph blew is the maximum number during the last collect interval; The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 88: Number of Available I/O Request Buffers



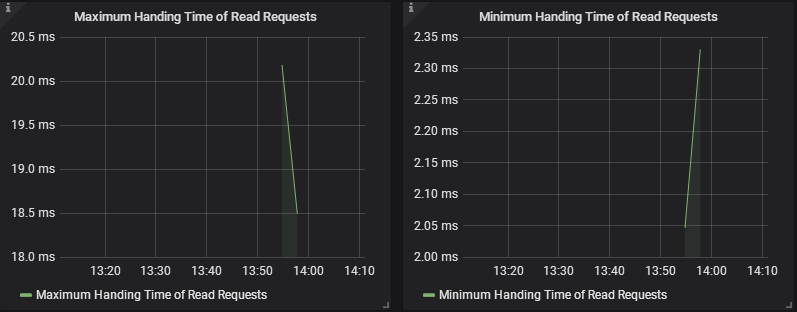
The Handing time of Punch Requests Panel (Figure 89) shows the maximum and minimum Handling time of Punch requests varying on time on OSS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Punch requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Punch requests during the last collect interval.

Figure 89: Handing Time of Punch Requests Panel



The Handing time of Read Requests Panel (Figure 90) shows the maximum and minimum Handling time of Read requests varying on time on OSS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Read requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Read requests during the last collect interval.

Figure 90: Handing Time of Read Requests Panel



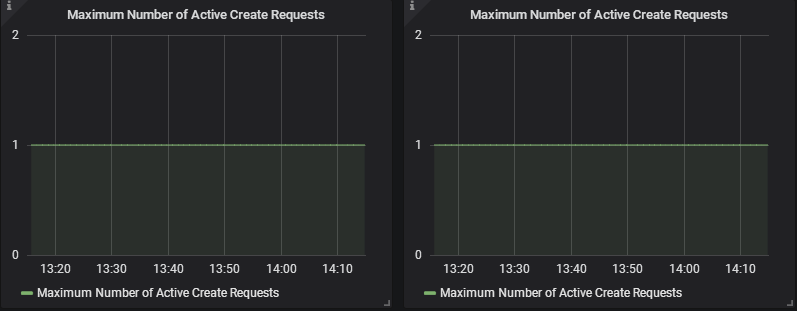
The Handing time of Write Requests Panel (Figure 91) shows the maximum and minimum Handling time of Write requests varying on time on OSS. The handling time of a request is the time interval between the time that it is started to be handled time and the time the handling finishes. The value shown in the left graph blew is the minimum handling time of the Write requests during the last collect interval; The value shown in the left graph blew is the minimum handling time of the Write requests during the last collect interval.

Figure 91: Handing Time of Write Requests Panel



The **Number of Active Create Requests** Panel (Figure 92) shows the maximum and minimum number of active create requests varying on time on OSS. Active requests are the requests that is being actively handled by this OSS, not including the requests that are waiting in the queue. If the number of active requests is smaller than PTLRPC thread number minus two (one for incoming request handling and the other for incoming high priority request handling), it generally means the thread number should be enough. The value shown in the left graph blew is the maximum number during the last collect interval. The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 92: Number of Active Create Requests Panel



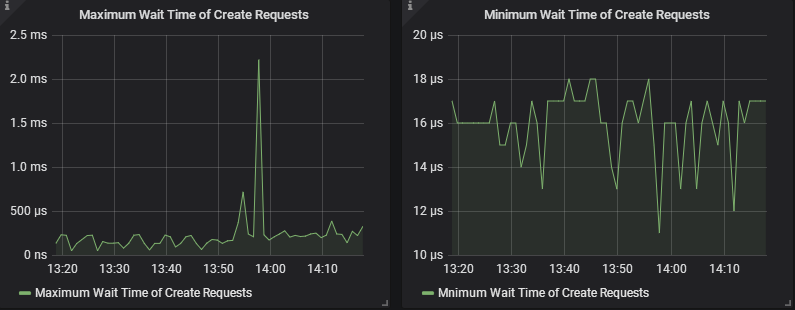
The Number of Incoming Create Requests Panel (Figure 93) shows the maximum and minimum number of incoming create requests varying on time on OSS. Incoming requests are the requests that waiting on preprocessing. A request is not incoming request any more when its proprocessing begins. And after preprocessing, the requests will be put into processing queue. The value shown in the left graph blew is the maximum number of incoming create requests during the last collect interval; The value shown in the right graph blew is the minimum number of incoming create requests during the last collect interval.

Figure 93: Number of Incoming Create Requests Panel



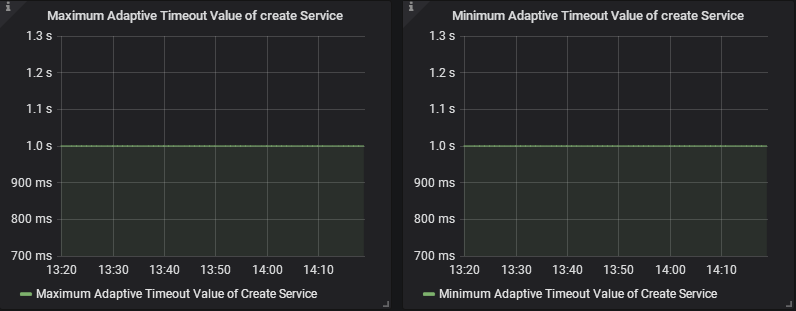
The Wait time of Create Requests Panel (Figure 94) shows the maximum and minimum wait time of create requests varying on time on OSS. The wait time of a request is the time interval between its arrival time and the time when it starts to be handled. The value shown in the left graph blew is the maximum wait time of the create requests during the last collect interval; The value shown in the right graph blew is the minimum wait time of the create requests during the last collect interval.

Figure 94: Wait Time of Create Requests Panel



The Adaptive Timeout Value of Create Service Panel (Figure 95) shows the maximum and minimum adaptive timeout value of the create Service varying on time on OSS. When a client sends a request, it has a timeout deadline for the reply. The timeout value of a service is an adaptive value negotiated between server and client during run-time. The value shown in the left graph is the maximum timeout of the create service during the last collect interval; The value shown in the right graph is the minimum timeout of the create service during the last collect interval.

Figure 95: Adaptive Timeout Value of Create Service Panel



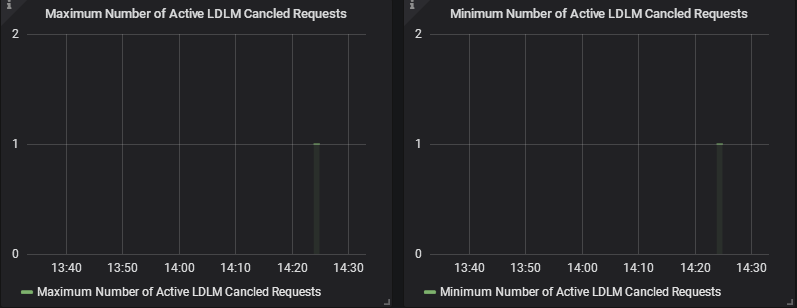
The Number of Available Create Request buffers Panel (Figure 96) shows the maximum and minimum number of available create request buffers varying on time on OSS. When a request arrives, one request buffer will be used. When number of available request buffers is under low water, more buffers are needed to avoid performance bottleneck. The value shown in the left graph blew is the maximum number during the last collect interval; The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 96: Number of Available Create Request Buffers Panel



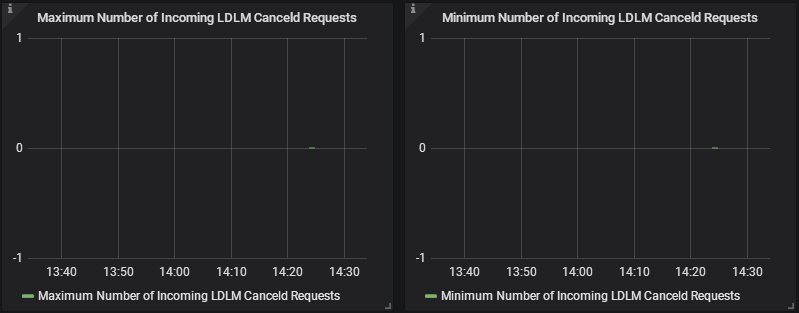
The **Number of Active LDLM Canceld** **Requests** Panel (Figure 97) shows the maximum and minimum number of active LDLM Canceld requests varying on time on OSS. Active requests are the requests that is being actively handled by this OSS, not including the requests that are waiting in the queue. If the number of active requests is smaller than PTLRPC thread number minus two (one for incoming request handling and the other for incoming high priority request handling), it generally means the thread number should be enough. The value shown in the left graph blew is the maximum number during the last collect interval. The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 97: Number of Active LDLM Canceld Requests Panel



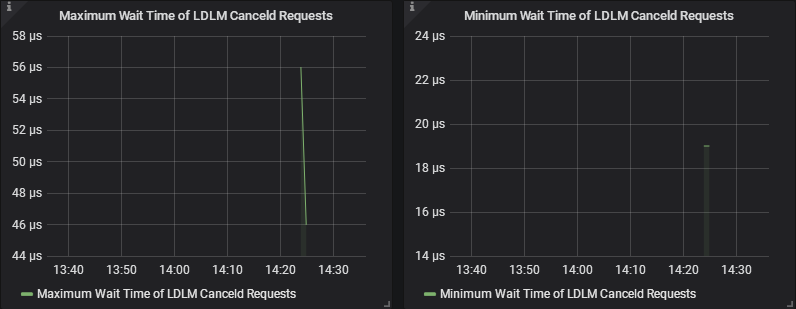
The Number of Incoming LDLM Canceld Requests Panel (Figure 98) shows the maximum and minimum number of incoming LDLM Canceld requests varying on time on OSS. Incoming requests are the requests that waiting on preprocessing. A request is not incoming request any more when its proprocessing begins. And after preprocessing, the requests will be put into processing queue. The value shown in the left graph blew is the maximum number of incoming LDLM Canceld requests during the last collect interval; The value shown in the right graph blew is the minimum number of incoming LDLM Canceld requests during the last collect interval.

Figure 98: Number of Incoming LDLM Canceld Requests Panel



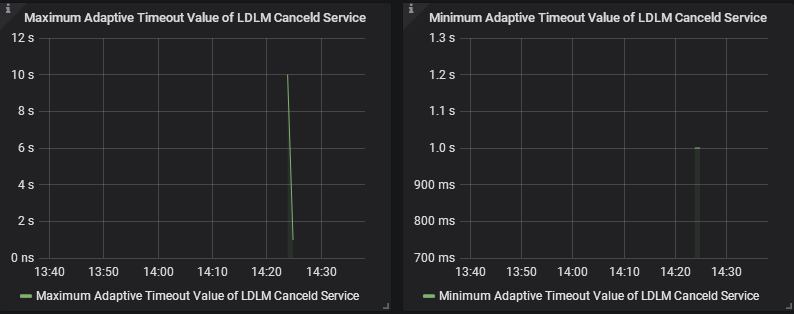
The Wait time of LDLM Canceld Requests Panel (Figure 99) shows the maximum and minimum wait time of LDLM Canceld requests varying on time on MDS. The wait time of a request is the time interval between its arrival time and the time when it starts to be handled. The value shown in the left graph blew is the maximum wait time of the LDLM Canceld requests during the last collect interval; The value shown in the right graph blew is the minimum wait time of the LDLM Canceld requests during the last collect interval.

Figure 99: Wait Time of LDLM Canceld Requests Panel



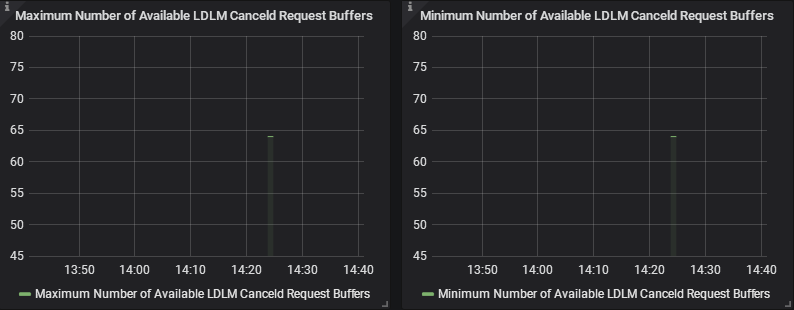
The Adaptive Timeout Value of LDLM Canceld Service Panel (Figure 100) shows the maximum and minimum adaptive timeout value of LDLM Canceld Service varying on time on OSS. When a client sends a request, it has a timeout deadline for the reply. The timeout value of a service is an adaptive value negotiated between server and client during run-time. The value shown in the left graph is the maximum timeout of the LDLM Canceld service during the last collect interval; The value shown in the right graph is the minimum timeout of the LDLM Canceld service during the last collect interval.

Figure 100: Adaptive Timeout Value of LDLM Canceld Service Panel



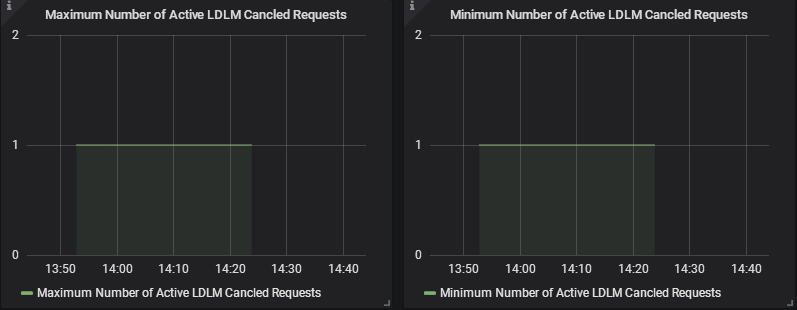
The Number of Available LDLM Canceld Request buffers Panel (Figure 101) shows the maximum and minimum number of available LDLM Canceld request buffers varying on time on OSS. When a request arrives, one request buffer will be used. When number of available request buffers is under low water, more buffers are needed to avoid performance bottleneck. The value shown in the left graph blew is the maximum number during the last collect interval; The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 101: Number of Available LDLM Canceld Request Buffers Panel



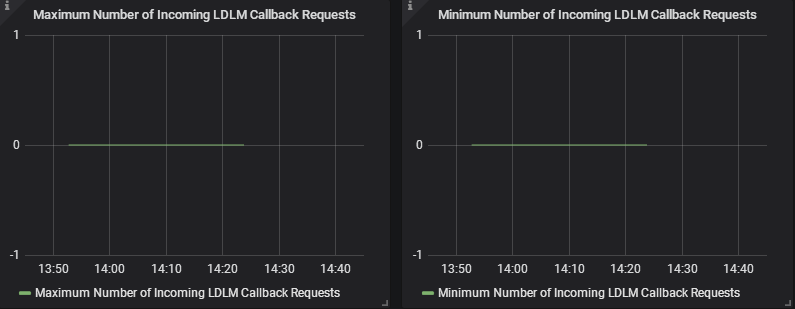
The **Number of Active LDLM Callback** **Requests** Panel (Figure 102) shows the maximum and minimum number of active LDLM Callback requests varying on time on OSS. Active requests are the requests that is being actively handled by this OSS, not including the requests that are waiting in the queue. If the number of active requests is smaller than PTLRPC thread number minus two (one for incoming request handling and the other for incoming high priority request handling), it generally means the thread number should be enough. The value shown in the left graph blew is the maximum number during the last collect interval. The value shown in the right graph blew is the minimum number during the last collect interval.

Figure 102: Number of Active LDLM Callback Requests Panel



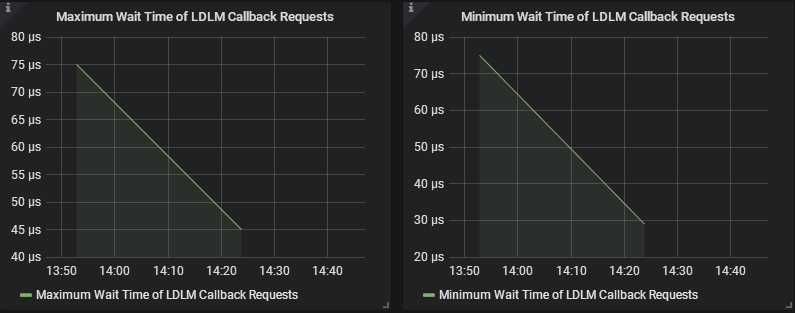
The Number of Incoming LDLM Callback Requests Panel (Figure 103) shows the maximum and minimum number of incoming LDLM Callback requests varying on time on OSS. Incoming requests are the requests that waiting on preprocessing. A request is not incoming request any more when its proprocessing begins. And after preprocessing, the requests will be put into processing queue. The value shown in the left graph blew is the maximum number of incoming LDLM Callback requests during the last collect interval; The value shown in the right graph blew is the minimum number of incoming LDLM Callback requests during the last collect interval.

Figure 103: Number of Incoming LDLM Callback Requests Panel



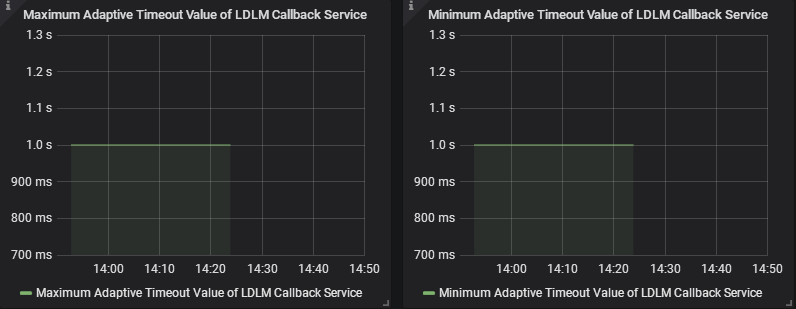
The Wait time of LDLM Callback Requests Panel (Figure 104) shows the maximum and minimum wait time of LDLM Callback requests varying on time on OSS. The wait time of a request is the time interval between its arrival time and the time when it starts to be handled. The value shown in the left graph blew is the maximum wait time of the LDLM Callback requests during the last collect interval; The value shown in the right graph blew is the minimum wait time of the LDLM Callback requests during the last collect interval.

Figure 104: Wait Time of LDLM Callback Requests Panel



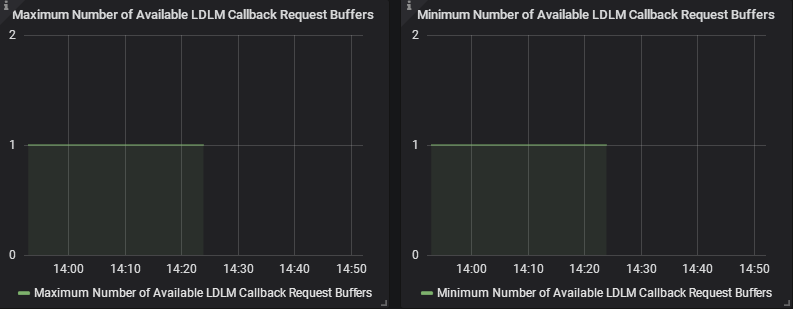
The Adaptive Timeout Value of LDLM Callback Service Panel (Figure 105) shows the maximum and minimum adaptive timeout value of LDLM Callback Service varying on time on OSS. When a client sends a request, it has a timeout deadline for the reply. The timeout value of a service is an adaptive value negotiated between server and client during run-time. The value shown in the left graph is the maximum timeout of the Readpage service during the last collect interval; The value shown in the right graph is the minimum timeout of the Readpage service during the last collect interval.

Figure 105: Adaptive Timeout Value of LDLM Callback Service Panel



The Number of Available LDLM Callback Request buffers Panel (Figure 106) shows the maximum and minimum number of available LDLM Callback request buffers varying on time on MDS. When a request arrives, one request buffer will be used. When number of available request buffers is under low water, more buffers are needed to avoid performance bottleneck. The value shown in the left graph blew is the maximum number during the last collect interval; The value shown in the right graph blew is the minimum number during the last collect interval.

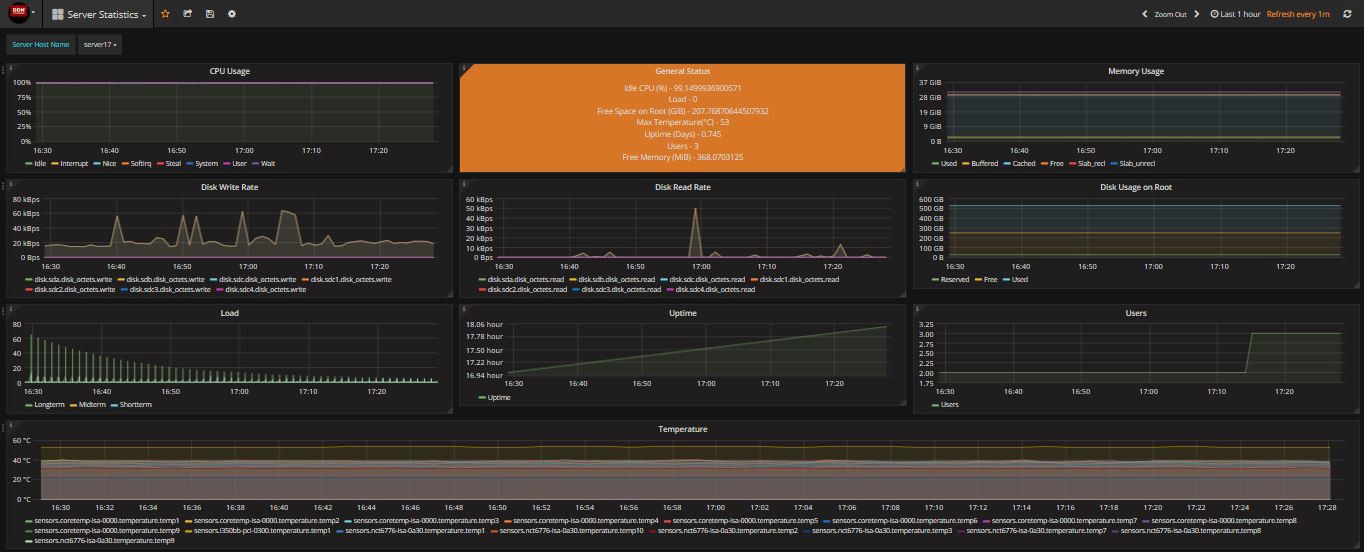
Figure 106: Number of Available LDLM Callback Request Buffers



## Server Statistics

The Server Statisticsdashboard (Figure 107) shows detailed information about a server.

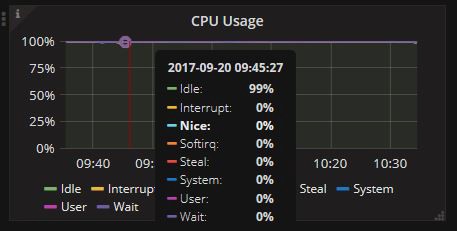
Figure 107: Server Statistics Dashboard



Below you will find description of some of the panels in the **Server Statistics** dashboard:

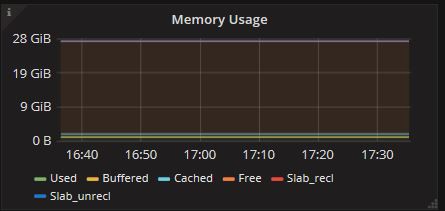
* The **CPU Usage** panel (Figure 108) shows the amount of time spent by the CPU in various states, most notably executing user code, executing system code, waiting for IO-operations and being idle.

Figure 108: CPU Usage Panel



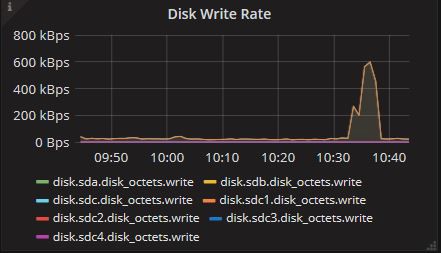
* The **Memory Usage** panel (Figure 109) shows how much memory has been used. The values are reported by the operating system. The categories are: **Used**, **Buffered**, **Cached**, **Free**, **Slab\_recl**, **Slab\_unrecl**.

Figure 109: Memory Usage Panel



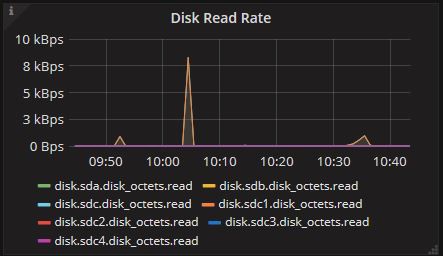
* The **Disk Write Rate** panel (Figure 110) shows the disk write rate of the server.

Figure 110: Disk Write Rate Panel



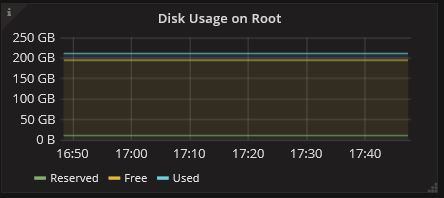
* The **Disk Read Rate** panel (Figure 111) shows the disk read rate of the server.

Figure 111: Disk Read Rate Panel



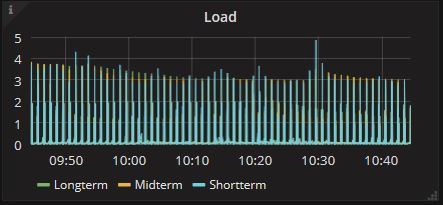
* The **Disk Usage on Root** panel (Figure 112) shows free space, used space and reserved space on the disk that is mounted as Root.  
  A warning message will be generated when there’s little free space left.

Figure 112: Disk Usage on Root Panel



* The **Load** panel (Figure 113) shows the load on the server. The system load is defined as the number of runnable tasks in the run-queue and is provided by many operating systems as follows:
  + **Shortterm**— one minute average
  + **Midterm**— five minutes average
  + **Longterm**— fifteen minutes average

Figure 113: Load Panel



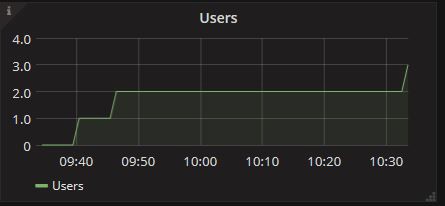
* The **Uptime** panel (Figure 114) shows how long the server has been working. It keeps track of the system uptime, providing such information as the average running time or the maximum reached uptime over a certain period of time.

Figure 114: Uptime Panel



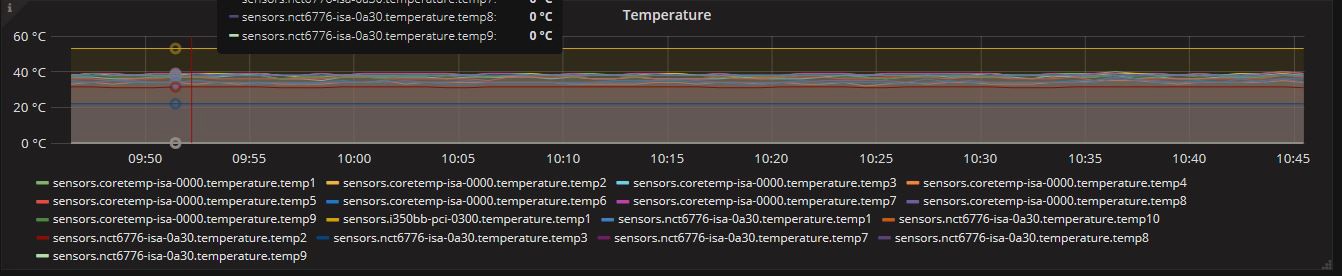
* The **User** panel (Figure 115) shows the number of users currently logged into the system.

Figure 115: User Panel



* The **Temperature** panel (Figure 116) shows the temperature collected from sensors.

Figure 116: Temperature Panel



## SFA Physical Disk Dashboard

The **SFA Physical Disk** dashboard shown in Figure 117 displays information about DDN SFA physical disks.

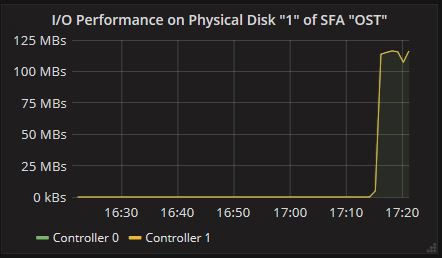
Figure 117: SFA Physical Disk Dashboard



Below you will find description of some of the panels in the **SFA Physical Disk** dashboard:

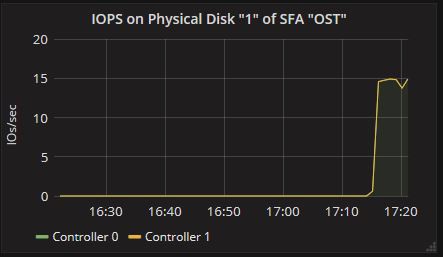
* The **I/O Performance on Physical Disk** panel (Figure 118) shows I/O speed over time.

Figure 118: I/O Performance on Physical Disk Panel



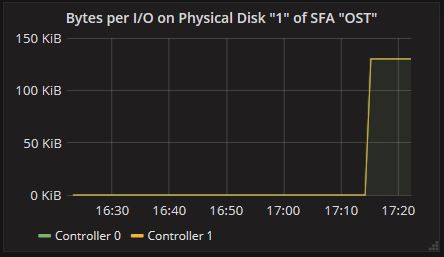
* The **IOPS on Physical Disk** panel (Figure 119) shows I/O operations per second on Physical Disk.

Figure 119: IOPS on Physical Disk Panel



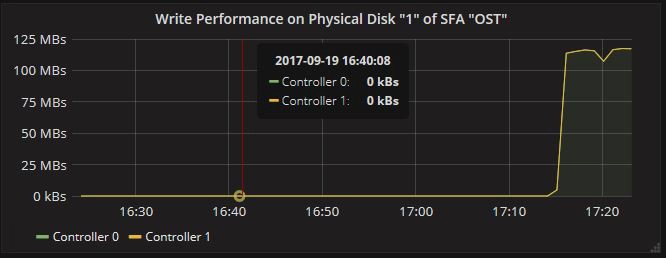
* The **Bytes per I/O** panel (Figure 120) shows the I/O bytes per second on each controller.

Figure 120: Bytes per I/O on Physical Disk Panel



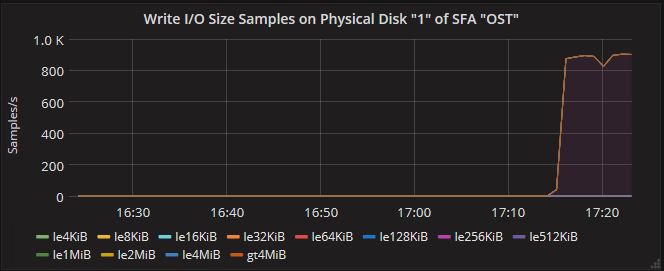
* The **Write Performance** panel (Figure 121) shows the write performance on each controller.

Figure 121: Write Performance on Physical Disk Panel



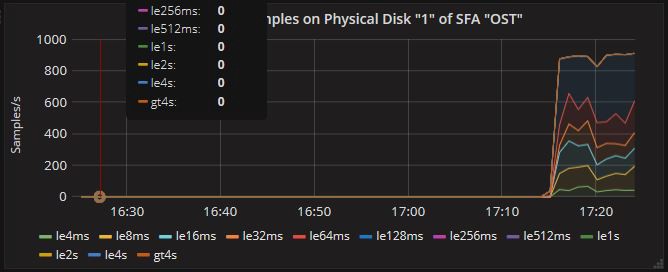
* The **Write I/O Size Samples** panel (Figure 122) shows the account of writting operation on each size.

Figure 122: Write I/O Size Samples on Physical Disk Panel



* The **Write Latency Samples** panel (Figure 123) shows the account of writing operation on each latency.

Figure 123: Write Latency Samples on Physical Disk Panel



## SFA Virtual Disk Dashboard

The **SFA Virtual Disk** dashboard (Figure 124 ) shows information about DDN SFA virtual disks:

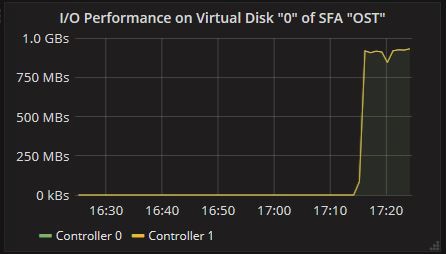
Figure 124: SFA Virtual Disk Dashboard



Below you will find description of some of the panels in the **SFA Virtual Disk** dashboard:

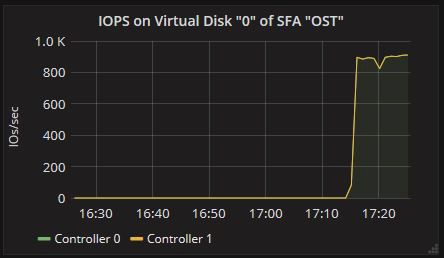
* The I/O Performance panel (Figure 125) in shows the I/O speed at a specific time.

Figure 125: I/O Performance on Virtual Disk Panel



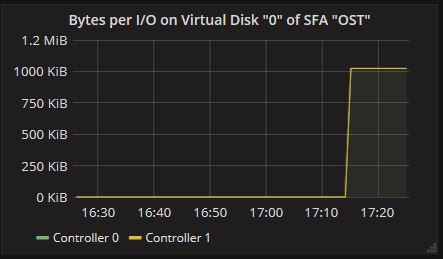
* The IOPS panel (Figure 126) shows I/O operations per second on Virtual Disk.

Figure 126: I/O Operations per Second on Virtual Disk Panel



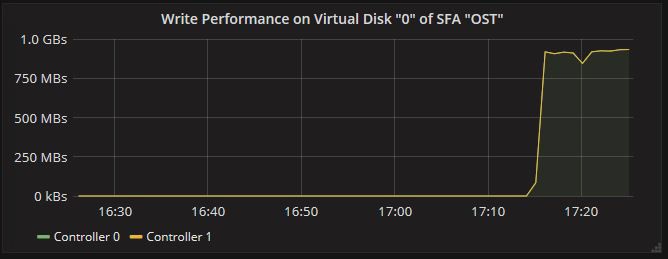
* The Bytes per I/O panel (Figure 127) shows I/O bytes per second on each controller.

Figure 127: Bytes per I/O on Virtual Disk Panel



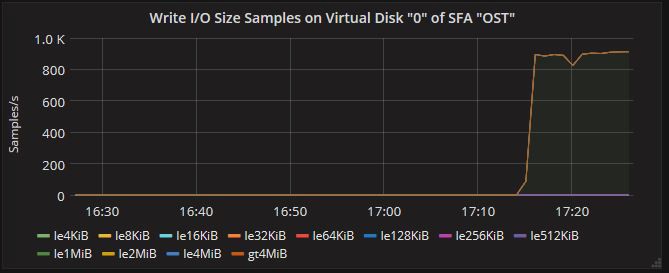
* The Write Performance panel (Figure 128) shows write performance on each controller.

Figure 128: Write Performance on Virtual Disk Panel



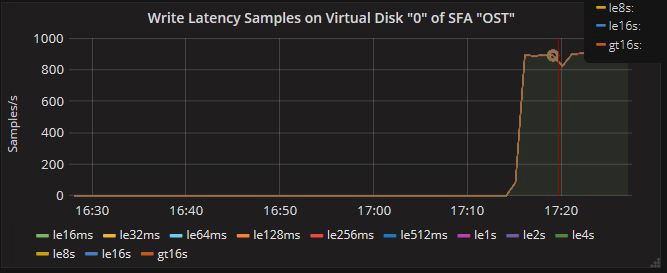
* The Write I/O Size Samples panel (Figure 129) shows the size distributions of write I/Os.

Figure 129: Write I/O Size Samples on Virtual Disk Panel



* The Write Latency Samples panel (Figure 130) shows the latency distributions of write I/Os.

Figure 130: Write Latency Samples on Virtual Disk Panel



# Stress Testing

In order to check whether the monitoring system works well under high pressure, DDN designed the **collectd-stress2** plugin for stress testing. It is an upgraded version of the **Stress** plugin, which can use a couple of **collectd** clients to simulate tens of thousands of metrics collected from hundreds of servers.

## Installing stress2 RPM on collectd Client

Because the **stress2** plugin generates a large amount of simulated monitoring data and contaminates the database, the plugin should not be installed on all clients by default. After the monitoring system has been installed using esmon\_install, select a couple of **collectd** clients as testing hosts and install the **stress2** plugins on each of the testing hosts. The RPM collectd-stress2 \* .rpm should be located in the ISO directory. To install the RPM, run the following command:

rpm --ivh collectd-stress2\*.rpm

## Updating Configuration File of Collectd Client

After **stress2** RPMs have been installed, update the configuration file /etc/collectd.conf and add the following configuration:

* **Thread** —Defines the number of test threads.
* **Metric** — Defines all the attributes of the monitoring target. It can be specified multiple times to simulate different monitoring targets at the same time. It contains the following attributes:
  + **Variable** — Defines the scope of the monitoring target changes and the speed of change, it can be specified multiple times.
    - **Name** — Defines the variable name.
    - **Number** — Defines the maximum range of variable changes.
    - **UpdateInterval** — Defines the time interval between variable changes.
  + **Host**—Defines the host name of the client, usually defined as "$ {key: hostname}", the program automatically sets the current host name. It describes the discriminator of the collection data object together with the following **Plugin**, **PluginInstance**, **Type**, **TypeInstance**. See [Naming Schema](https://collectd.org/wiki/index.php/Naming_schema) for details.
  + **Plugin**—Defines the plugin member in the collectd identifier.
  + **PluginInstance**—Defines the plugininstance member in the collectd identifier.
  + **Type**—The type member of the collectd identifier. For details, see <https://collectd.org/wiki/index.php/Derive>.
  + **TypeInstance**—Defines the type instance member in the collectd identifier.
  + **TsdbName**—Defines the name submitted to the database format.
  + **TsdbTags**—Defined the tags submitted to the database format to facilitate the late classification search.

Below is an example of /etc/collectd.conf.

**Example:**

LoadPlugin stress2

<Plugin "stress2">

  Thread 32

  <Metric>

<Variable>

    Name "ost\_index"

    Number 10

    UpdateIterval 0

</Variable>

<Variable>

    Name "job\_id"

    Number 7000

    UpdateIterval 10

</Variable>

  Host "${key:hostname}"

  Plugin "stress-${variable:ost\_index:OST%04x}"

  PluginInstance "jobstat\_${variable:job\_id:job%d}"

  Type "derive"

  TypeInstance "sum\_read\_bytes"

  TsdbName "ost\_jobstats\_samples"

  TsdbTags "optype=sum\_read\_bytes fs\_name=stress ost\_index=${variable:ost\_index:OST%04x} job\_id=${variable:job\_id:job%d}"

   </Metric>

  <Metric>

<Variable>

    Name "mdt\_index"

    Number 10

    UpdateIterval 0

</Variable>

<Variable>

    Name "md\_stats"

    Number 10

    UpdateIterval 10

</Variable>

  Host "${key:hostname}"

  Plugin "stress-${variable:mdt\_index:MDT%04x}"

  PluginInstance "md\_stats"

  Type "derive"

  TypeInstance "open"

  TsdbName "md\_stats"

  TsdbTags "optype=open fs\_name=stress mdt\_index=${variable:mdt\_index:MDT%04x} mdt\_stats\_open=${variable:mdt\_stats\_open:%d}"

   </Metric>

</Plugin>

## Start Testing

After modifying the configuration file, restart collectd:

service collectd restart

A message like the following should appear in */var/log/messages*:

server11 collectd[20830]: stress2: time: 1.79244 for 70100 commits with 32 threads, 39108.70099 commits/second

The above information shows that stress2 plugin successfully loaded , and generated a lot of monitoring data. With the above configuration file and following specified hardware environment, the corresponding monitoring bottlenecks were checked.

* **OS：**CentOS7.
* **Memory：**128GB.
* **CPU:** Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz.
* **Disk:** Samsung SSD 850 2B6Q.

The monitoring client and database server are running on the same host, Influxdb data is stored on SSD with ext4 file system.

**Preconditions:**

* **Collectd Interval:** 60 seconds.
* **Grafana History:** 1 hour.
* **Grafana Refresh Interval:** 60 seconds.
* **Collectd Running Time:** more than 1 hour.

**Conclusion:**

* **Grafana keeps on refreshing:** monitor overload.
* **Grafana has idle time:** monitor running well.

In theory, Grafana's refresh time equals the database query time plus the web page load time.

We can query the database to measure the performance of the database query. For example the following is the default query command for LustrePerfMon Grafana **Read Throughput per Job**:

influx -database esmon\_database –execute \

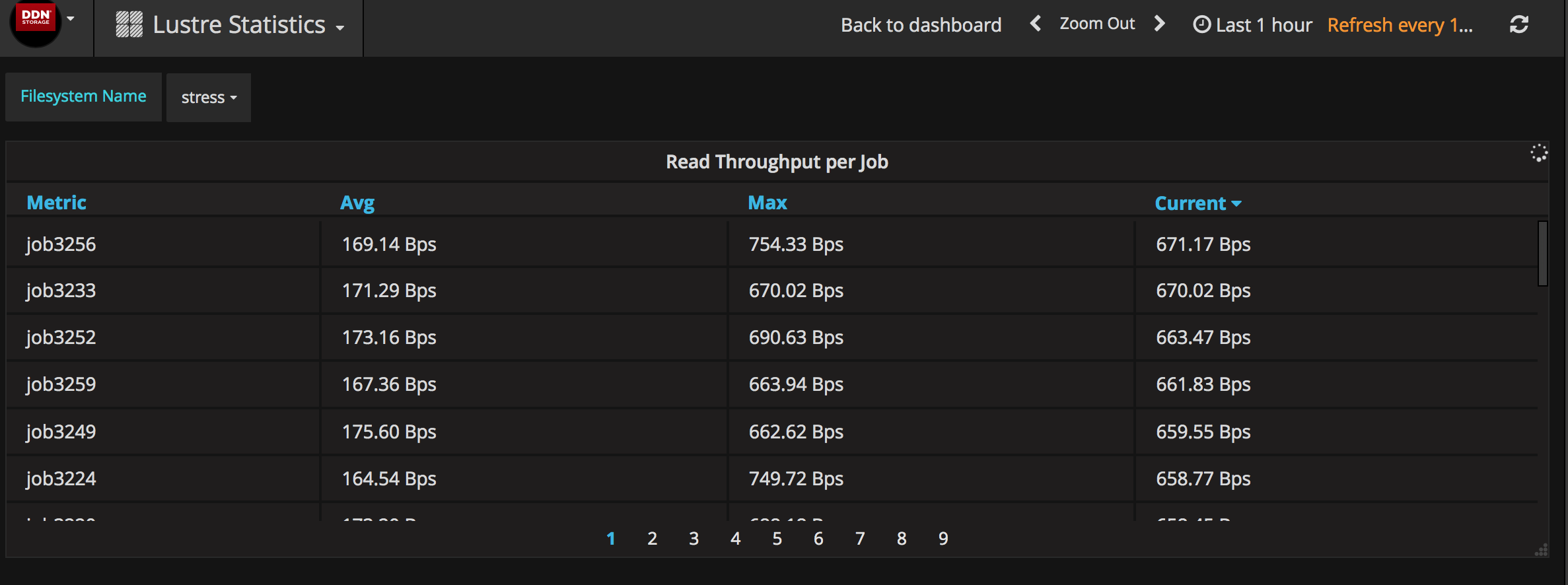
"SELECT "value" FROM "ost\_jobstats\_samples" WHERE ("optype" = 'sum\_read\_bytes' AND "fs\_name" = 'stress') AND time >= now() - 1h GROUP BY "job\_id""

With the monitoring software running, the above command on the database host can be executed to verify the query time. As shown in Figure 71, the query time of the Influxdb grew linearly during the first hour, because the data points kept on accumulating . But after an hour, the query time became steady, which is also expected behavior.

Figure 131：Influxdb Query Time

After verifying the load on the database side, we also need to verify the loading status of Grafana. Log in to Grafana to see **Read Throughput per Job** (see Figure 72)

Figure 132：Read throughput per Job stress testing



If the page is always refreshing and the page can be loaded within 60 seconds, that means, under the current configuration, the monitoring system can handle the current pressure. Otherwise, the monitoring system can be considered overloaded. In that case, either hardware need to be upgraded or the data collecting/refreshing intervals need to be increased. By continuously adjusting the number of **job\_id** in */etc/collectd.conf* and checking the page refreshing latency, the maximum supported metrics can be known under the current hardware configuration. Tests show that if Lustre has 10 OSTs, with above hardware, the monitoring system can support up to 7000 running jobs at the same time without any problem.

# Troubleshooting

The directory */var/log/esmon\_install/[installing\_date]* on the Installation Servergathers all the logs that is useful for debugging. If a failure happens, some error messages will be written to the file */var/log/esmon\_install/[installing\_date]/error.log*. The first error message usually contains the information about the cause of failure.