# Performance Optimization of the Open XDMoD Datawarehouse


Gregary M. Dean
Center for Computational Research, University at Buffalo
Buffalo, NY, USA
gmdean@buffalo.edu

Joshua Moraes
Center for Computational Research, University at Buffalo
Buffalo, NY, USA
joshuamoraes@berkeley.edu

Joseph P. White
Center for Computational Research, University at Buffalo
Buffalo, NY, USA
jpwhite4@buffalo.edu

Robert L. DeLeon
Center for Computational Research, University at Buffalo
Buffalo, NY, USA
rldeleon@buffalo.edu

Matthew D. Jones
Center for Computational Research, University at Buffalo
Buffalo, NY, USA
jonesm@buffalo.edu

Thomas R. Furlani
Roswell Park Comprehensive Cancer Center
Buffalo, NY, USA
Thomas.Furlani@roswellpark.org



## ABSTRACT

Open XDMoD is an open source tool to facilitate the management of high performance computing resources. It is widely deployed at academic, industrial, and governmental HPC centers and is used to monitor large and small HPC and cloud systems. The core of Open XDMoD is a MySQL based data warehouse that is designed to support the storage of historical information for hundreds of millions of jobs with a fast query time for the interactive web portal. In this paper, we describe the transition that we made from the MyISAM to the InnoDB storage engine. In addition, other improvements were also made to the database queries such as reordering and adding indices. We were able to attain substantial performance improvements in both the query execution and in the data ingestion/aggregation. It is a common trend that databases tend to grow in size and complexity throughout their lifetime; this work presents a practical guide for the types of practices and procedures that can be done to maintain data retrieval and ingestion performance.



## CCS CONCEPTS

• **General and reference** → **Metrics**; **Performance**; • **Hardware** → **Platform power issues**.

## KEYWORDS

databases, MySQL, XDMoD

**ACM Reference Format:**
Gregary M. Dean, Joshua Moraes, Joseph P. White, Robert L. DeLeon, Matthew D. Jones, and Thomas R. Furlani. 2022. Performance Optimization of the Open XDMoD Datawarehouse. In *Practice and Experience in Advanced Research Computing (PEARC '22), July 10–14, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3491418.3530290




*PEARC '22, July 10–14, 2022, Boston, MA, USA*



ACM ISBN 978-1-4503-9161-0/22/07…$15.00
https://doi.org/10.1145/3491418.3530290


## 1 INTRODUCTION

Open XDMoD [3] is an open source tool for the comprehensive management of cyber-infrastructure resources including high performance computing (HPC), storage and cloud computing resources. Its main components are a MySQL-based datawarehouse supporting the storage and querying of CI data and an interactive web portal that facilitates rapid access to the data. The interactive web portal provides a rich set of features via an intuitive graphical interface that is tailored to the role of the logged in user (such as center director, center staff, principal investigator, end user). Metrics provided by XDMoD include information about HPC resource usage, Cloud resource usage, Storage usage and HPC job performance data. Metrics can be broken down by many dimensions including by user, principal investigator, resource, queue, job size and field of science.

Open XDMoD is designed to support rapid retrieval and analysis of large amounts of historical data. The datawarehouse architecture is designed so that the queries run by the interactive XDMoD web portal run as fast as possible. For example, the XDMoD instance that is used for monitoring of XSEDE-allocated CI resources has information about more than 200 million HPC jobs over the last 19 years and has a web portal that supports chart load times in less than 200 ms for most charts.

Open XDMoD development has been ongoing since 2011. During this time the development team have added new features, updated to support new operating systems and engaged in a program of continuous improvements. During the early stages of the program, extensive performance testing was done to ensure the responsiveness of the XDMoD portal. One of the design decisions involved the choice of MySQL database storage engine. At the time, the `MyISAM` storage engine was found to have better performance than the `InnoDB` storage engine for the XDMoD use case.

Since the original XDMoD datawarehouse design, the MySQL software has undergone significant development including new storage engines and improvements to the existing engines. A community-developed, commercially supported fork of MySQL, called MariaDB was also created. In recent versions of MySQL and MariaDB, the default database storage engine changed from MyISAM to InnoDB. The MySQL documentation [1] states that the MyISAM storage engine performs best with read-mostly data or with low-concurrency operations, which aligns well with XDMoD's requirements. However InnoDB has undergone

significant performance improvements since the original XDMoD performance analysis and is now the default storage engine.

The purpose of this work was to determine if the InnoDB storage table engine, with its performance improvements, would now be a better choice for XDMoD. To measure the impact of this change, testing was split into three parts. First, determine if any database changes were needed to our tables before converting XDMoD tables to the InnoDB table engine. Second, measure the performance impact of converting XDMoD's tables to InnoDB on the XDMoD web portal. Last, test XDMoD's data processing pipelines for any performance impact when they operate on InnoDB tables for ingesting and aggregating data into the XDMoD data warehouse.

## 2 BACKGROUND

The main database performance testing was performed using the Open XDMoD instance at the Center for Computational Research at the University at Buffalo (CCR). The XDMoD instance contains data from CCR's academic HPC cluster and CCR's cloud service. During this study, it was running Open XDMoD software version 9.0 and contained approximately 20 years of job accounting data for 47 million HPC jobs from the cluster as well as data from CCR's OpenStack cloud instance, storage data and job performance data. The database server uses Centos 7 operating system with MariaDB 5.5 and has disk usage of approximately 300 GB.

### 2.1 Open XDMoD data architecture

A high-level overview of the data flow in Open XDMoD is shown in Figure 1. Source records (such as job accounting data or cloud instance events) are loaded into a MySQL database via a batch process. This process is designed to run in batch mode with no end-user interaction and will typically write large amounts of data and has a high I/O load on the database. The batch process has two stages: ingestion, which loads new data into intermediate database tables; and aggregation, where queries are run on the ingested data and their results stored. The aggregation step ensures that the data are stored in the database for efficient retrieval by the portal. Each stage can consist of multiple steps, which are called actions in XDMoD, and those actions can be combined to run in a specific order. XDMoD calls the groupings of these actions into a specific order a data pipeline. For example, the data pipeline for aggregation consists of actions to aggregate data for each day, month, quarter and year.

End users interact with the datawarehouse via the web portal. When users request charts or data in the portal, php code running on the webserver generates an SQL query, which is executed. The SQL queries for charts and data are all read-only and do not modify the database contents. Since the queries are generated and executed in response to user input, they are designed to generate results as quickly as possible to keep the portal responsive.

The ingestion and aggregation steps are designed to run periodically and the web portal typically is unresponsive when the aggregation step is running. The Open XDMoD instance at CCR runs the ingestion and aggregation process daily at 4 am and it runs for approximately 1.5 hours. This ensures that the ingestion and aggregation does not happen during normal office hours when CCR users are typically using the Open XDMoD portal.

### 2.2 Differences between MyISAM to InnoDB

The first task was to determine the differences between the MyISAM and InnoDB database engines and how they would affect XDMoD. The main differences were InnoDB's support of row-level locking, foreign key relationships and transactions, and that all InnoDB tables must also have a PRIMARY KEY. If a PRIMARY KEY is not specified, MySQL will use the first non-NULL UNIQUE key. If a non-NULL unique does not exist, MySQL creates a 6-byte hidden integer.

Those differences did not present a problem with moving to InnoDB but some MyISAM-specific functionality was found in XDMoD's tables that prevented conversion to InnoDB. The issue was the column order of some of our PRIMARY KEY indexes. For MyISAM tables, an AUTO_INCREMENT directive can be specified on a secondary column in a multiple-column index but, this is not supported when using InnoDB. When using this functionality, the generated value for the AUTO_INCREMENT column is calculated as MAX(auto_increment_column) + 1 WHERE prefix=given-prefix. This can be useful when you want to put data into ordered groups. Tables that use this multi-column index were changed to have a single-column auto-incrementing ID column.

Differences also exist in how InnoDB constructs table indexes. InnoDB uses clustered indexes which are data structures that store data together with it's indexes. In InnoDB, the clustered index is usually the PRIMARY KEY for a table. Accessing a row through the clustered index provides performance improvements since the index search leads directly to the page that contains the row data. MyISAM does not use clustered indexes.

After identifying the differences between MyISAM and InnoDB and how they may affect XDMoD, the tables were converted to InnoDB. The time to convert tables to InnoDB was considerable for larger tables. For example, it took around 3 hours to convert a table with 42 million rows. An increase in the amount of disk space needed for InnoDB tables was observed too. For example, converting the `modw` schema (which stores job accounting data) on the development instance of Open XDMoD at CCR resulted in disk space increasing 58 GB to 94 GB. How much more space InnoDB needs is dependent on the table schema and data in the table being converted.

### 2.3 XDMoD Web Portal

XDMoD Portal testing was split into two main parts. First, we tested the XDMoD web portal to measure the performance impact of changing to the InnoDB table engine with our specific data models and access patterns. Second, we tested XDMoD's data processing pipelines to measure the performance impact of the table engine conversion, and identify any changes that should be made to the tables when using the InnoDB engine in order to improve performance.

XDMoD's suite of regression tests were used to measure the performance of XDMoD's web portal in response to the table engine conversion. XDMoD's regression tests are used to ensure XDMoD functions as expected after code changes. These regression tests work by sending a request to the XDMoD API for each combination of Statistic and Group By in a Realm, comparing the data returned from the XDMoD API to the data expected. In XDMoD, Statistics


**Figure 1: Overview of the main process interactions with the XDMoD datawarehouse. The ingestion and aggregation process is responsible for loading data into the database. XDMoD users have read-only access to the data via queries run from a webserver.**

determine the SQL columns that are selected for an autogenerated query that is used to show data in the XDMoD Web Portal. A Group By specifies the columns to add to the SQL Group By clause of the autogenerated query. A Realm represents a collection of Statistics and Group By's for a set of data. For example, the Jobs realm contains data about each job run on an HPC system and contains XDMoD statistics such as the number of jobs ended and the total number of CPU hours used and XDMoD group by's for the HPC resource and User among others. These regression tests provided us with a repeatable way to measure how the conversion to InnoDB affected XDMoD and make sure that XDMoD reported the expected value for each Statistic and Group By. The tests also recorded the time each test took before and after converting the table engine to measure the performance impact of this change.

Our testing plan consisted of setting up a virtual machine with the current version of Open XDMoD (which at the time was version 9.0), installed on it and running our regression tests with the tables using the MyISAM engine. Once the regression tests completed, the MyISAM tables were converted to InnoDB and the regression tests were run again. Each time the regression tests ran, the time it took for each test to complete was collected, along with the statistic and group by for the test and the data was analyzed to determine the performance impacts of converting our tables to InnoDB.

### 2.4 XDMoD data pipeline processing

By default, XDMoD logs the time each data processing pipeline takes when ingesting and aggregating data into XDMoD. These logs were used to compare the performance difference between MyISAM and InnoDB tables for XDMoD's data pipeline processing.

The process for comparing our data processing pipelines before and after the table engine changes was the following: Run the ingestion and aggregation process to ingest and aggregate data, convert necessary tables to InnoDB, and then run the ingestion and aggregation process again to ingest and aggregate data with tables as InnoDB.

After completing this process, we compared data collected from our logs for both runs to measure the performance difference. After analyzing this data, we identified data pipelines that were performing either slower than when the tables were MyISAM or data pipelines that took a long time regardless of the table engine.

## 3 OPTIMIZATIONS

After converting XDMoD tables to InnoDB, analysis of our log data allowed us to identify poorly performing data pipelines, which we investigated and optimized to provide better performance. Four areas were identified where better performance was potentially achievable. Those area were: the aggregation of data, queries that use a 'last_timestamp' column in a where clause, queries that load data into dimension tables, and the use of the `OPTIMIZE TABLE` command on aggregate tables.

### 3.1 Aggregation of data

XDMoD aggregates data for a realm on four different levels of time granularity: daily, monthly, quarterly, and yearly. This aggregation is a mechanism to improve XDMoD performance. XDMoD uses the term `Aggregation Unit` to identify these time periods. After converting XDMoD's tables to InnoDB, analysis of the logs for data processing pipelines for the Jobs realm data identified two main issues that slowed the run times of XDMoD's data pipelines. First, a sub-optimal table index on the Jobs realm aggregate tables. With the tables as InnoDB, aggregation at the daily granularity level took between 25 and 40 minutes. Using the MySQL EXPLAIN EXTENDED command we looked at the query plan MySQL was making for this query. The aggregate query's WHERE clause "WHERE start_day_id <= end_day_ AND end_day_id >= start_day AND is_deleted = 0" was identified as the culprit as to why these queries were taking longer than expected. The source table for this query had an index for the is_deleted column and a multi-column index consisting of the start_day_id and end_day_id but, there was not an index that contained all three columns. An index that contains all columns in a where clause will speed up a query compared with one that does not, especially as the number of rows in a table grows.

After adding a new multi-column index that contained all three columns, is_deleted, start_day_id, and end_day_id, the data aggregation pipelines were run again. While this produced some improvements they were still not running optimally. To optimize the query further, the columns in the new multi-column index were re-ordered. The order of columns in an index is helpful when the most selective columns in the index come first. The suggested order for columns in an index is: columns in the where clause that use the equality operator, columns in the where clause that are used in a 'range' (LIKE, BETWEEN, <), all columns in the Group By clause, matched to the order in the Group By clause, and all columns in Order By clause, matched to the order in the Order BY clause.

When looking at XDMoD's data access patterns, changing the column order of the index to (is_deleted, end_day_id, start_day_id) is the most selective and thus gives better performance. After changing the index column order, the EXPLAIN EXTENDED command showed that this new column order would create a more efficient query plan. When the aggregation pipelines ran again with this change, we observed a significant reduction in the time for the daily aggregation time period to run. While we focused on the day

aggregation time period, the addition of the new multi-column index provided performance improvements for the month, quarter, and year aggregation time periods too.

### 3.2 Indexing the `last_timestamp` column

Queries whose WHERE clause includes the `last_timestamp` column is another area where we sought performance improvements. In the XDMoD data warehouse, the `last_timestamp` field is defined as a TIMESTAMP data type and has the ON UPDATE CURRENT_TIMESTAMP directive. This directive sets the column's value to the current time whenever a row is inserted into the table or any value in a row is changed. XDMoD uses this column to determine what rows to select for aggregation in the data aggregation pipelines.

After converting to InnoDB, the query responsible for choosing the time period to aggregate took longer than expected. The EXPLAIN EXTENDED command showed that the query's WHERE clause used the `last_modified` column, which was not part of any index. Adding an index that contained the `last_timestamp` column improved the query significantly, making it quicker than when the tables were MyISAM. Adding this index to other tables with a `last_timestamp` column showed improvements to the aggregation of other realms, such as the Storage realm, but they were not as significant. This is mainly because other realms do not aggregate as many records as the Jobs realm.

One other location where the `last_timestamp` column is used in a WHERE clause are two queries that run after ingesting data into the Jobs realm. These queries update the processor count of recently ingested jobs and the CPU Time, GPU Time, Submit Time, and Wait Duration. The performance improvement for these queries was significant.

### 3.3 Dimension table optimizations

The XDMoD data warehouse uses a star schema architecture [2] that consists of fact and dimension tables. When ingesting data, each HPC job is put into a fact table whose columns may reference a dimension table. In XDMoD, these dimension tables are populated by selecting the unique values for a column from a fact table and inserting them into the appropriate dimension table. As the number of rows in the fact table increase, so to does the queries completion time when you try to select all the unique values from all of the jobs ever ingested into an XDMoD instance. The data pipeline action for populating the Queue dimension table worked in this manner. Instead of selecting the Queues from all HPC jobs that XDMoD has ingested, it is more efficient for the query to only select Queues from HPC jobs that have not been ingested. Re-writing the SQL query to only select jobs that have not been ingested provided a significant performance improvement.

### 3.4 Use of the `Optimize Table` command on aggregate tables

In MySQL, the `OPTIMIZE TABLE` command reorganizes the physical storage of table data and the associated index data to reduce storage space and improve I/O efficiency when accessing tables. The exact changes made to a table depend on the storage engine the table uses. XDMoD runs an `OPTIMIZE TABLE` command on aggregate tables at the end of the data aggregation pipelines. After converting our tables to InnoDB, the `OPTIMIZE TABLE` command started taking much longer to run than when using the MyISAM table engine. Along with this, the .ibdata file, which stored all the InnoDB data, began growing larger at a very rapid pace.

MySQL also has a setting called 'innodb_file_per_table'. In MySQL 5.5 and lower, the default for this setting is false. This setting determines if data for any InnoDB tables are in one file, .ibdata., or in separate files for each table. The .ibdata file does not shrink even when deleting data from an InnoDB table, dropping an InnoDB table, or rebuilding an InnoDB table. The `OPTIMIZE TABLE` statement creates a new copy of the table specified, placing it at the end of the .ibdata file, but does not remove the old copy of the table in the .ibdata file. This behavior, and the fact that the aggregate tables being optimized were many GB in size, lead to an unsustainable increase in the file size. The larger file size and difference in what the `OPTIMIZE TABLE` command does for InnoDB tables compared to MyISAM table caused an increase in time for the command to run that was unacceptable to us.

To mitigate the unsustainable increase in file size and reduce the time for the command to complete, we set the 'innodb_file_per_table' setting to TRUE. The effect of this is that `OPTIMIZE TABLE` will create a new identical empty table in a new file and, then it will copy data row by row from the old table to the new one and then delete the old file. This prevents the unsustainable file size growth and allows the command to complete quicker. While it is still longer than when the tables were MyISAM, the time it takes to run is acceptable.

## 4 RESULTS

To measure the results of converting XDMoD tables to InnoDB, we used timing results from our regression tests and data from our log files that contained the time our data processing pipelines took to run. Our results are primarily concerned with the time impact of our changes, that is, is the XDMoD web portal faster or slower as a result of the conversion to the InnoDB table engine? Are the data ingestion processes faster or slower with tables converted to the InnoDB table engine?

### 4.1 XDMoD Web Portal

To measure the results of the InnoDB changes on the XDMoD web portal, we used data gathered while running XDMoD's regression tests. As stated in section 2.3, the regression tests ran under two scenarios. One with a version of XDMoD with MyISAM tables and the other with any MyISAM tables converted to the InnoDB table engine. In both scenarios, each test in our regression test suite submitted a request to the XDMoD API that consisted of a Time Period, Statistic, and Group By. When a response for this request was received, our regression tests recorded the time between when the request was sent and when the response was received.

First, we analyzed the overall time the test suites took to complete under each scenario. When using the MyISAM table engine, the test suite completed in 8014 seconds. When using the InnoDB table engine, the tests completed in 4517 seconds. This represents a 43% decrease in the time for the tests to complete when using the InnoDB table engine.

We also gathered information on the number of tests that saw their time to complete decrease and how the average amount of time to complete a test changed. As seen in Figure 2, out of 24401 tests run in each scenario, 87% of tests saw their time to complete decrease. The average change in the amount of time to complete a test was -33%, meaning tests ran an average of 33% faster when using the InnoDB table engine.

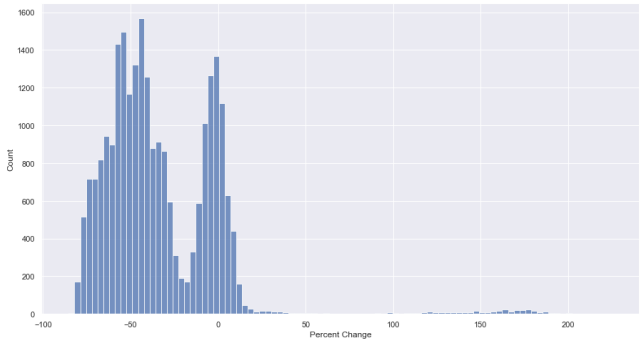


**Figure 2: Histogram distribution of percent change for individual tests**

While this was a significant result, we wanted to know how individual tests performed between the two scenarios. In particular, we wanted to find any trends related to the performance of specific Time Period's, Statistics, Group By's, or Realms. We again looked at the percent change but instead of the percent change of the complete regression test suite we looked at the percent change of each individual test.

When looking at the percent change of each tests on a scatter chart, Figure 3 , we noticed a concentration of tests that were much slower when using the InnoDB engine. Many of them were more than twice as slow.

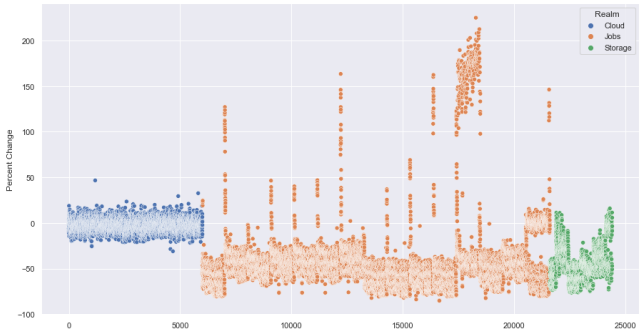


**Figure 3: Scatter of percent change for individual tests color-coded by XDMoD Realm**

To try and identify any pattern within those tests, we looked at color-coded scatter charts. We created multiple charts that were color-coded by features of our tests, such as the Group By, Statistic, Realm and Aggregation Unit, for example see Figure 3 and Figure 4.

When color-coding the charts by each of these features, we discovered that many slower tests requested the Queue Group By in the Jobs realm.

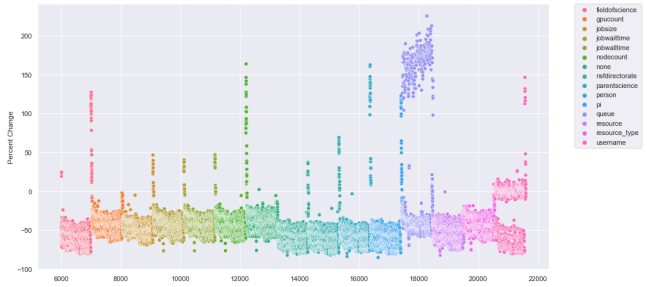


**Figure 4: Scatter of percent change for individual tests color-coded by XDMoD Group By**

With such a clear cluster of results, we invetigated the Queue Group By to see if other patterns existed. When we color-coded by the Aggregation Unit, Figure 5, we discovered that almost all tests performing worse were requesting the Year Aggregation Unit. These tests took almost twice as long to complete.

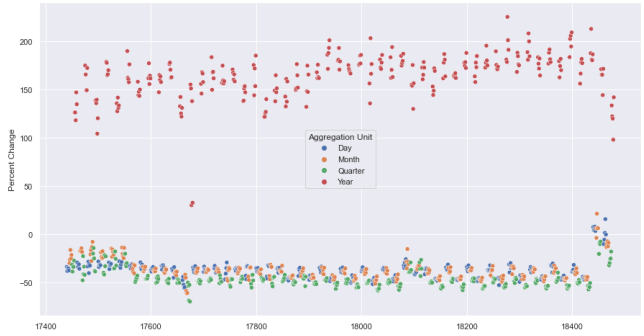


**Figure 5: Scatter of percent change for individual tests color-coded by XDMoD Group By**

Our final analysis of the effect that switching to the InnoDB table engine has on the XDMoD Web Portal is that it has a positive effect. As stated earlier, the XDMoD regression tests suite completed its tests 43% quicker when using the InnoDB table engine. A total of 83% of individual tests had their time to complete reduced, and the average time to complete a request decreased by 33%. Table 1 lists the mean time a test took for MyISAM compared to InnoDB and also show how the times differed by Realm.

## 4.2 Data Processing Pipelines

The improvements gained from converting XDMoD's tables to InnoDB and related optimizations were significant for our data processing pipelines as well. To measure the impact of this change on our data processing pipelines, we analyzed the XDMoD logs that contain the time to complete our data processing pipelines.

The data processing pipeline responsible for ingesting jobs data into XDMoD saw a significant decrease in time to complete with

**Table 1: Comparison of MyISAM and InnoDB Results**

| Tests | MyISAM mean time (sec) | InnoDB mean time (sec) | Mean percent change | Percentage of Tests Faster |
|---|---|---|---|---|
| All Tests | 0.33 | 0.18 | 33% | 87% |
| Jobs Realm | 0.43 | 0.23 | 43% | 95% |
| Storage Realm | 0.31 | 0.14 | 43% | 95% |
| Cloud Realm | 0.065 | 0.063 | 2% | 64% |

the conversion to InnoDB, the addition of indexes on the appropriate columns, and rewriting some queries to be more efficient as mentioned in section 3.3.

Data processing actions where we added a table index to the 'last_timestamp' column saw significant improvement. Processing steps that took around 200 seconds now take between 5 and 10 seconds, a roughly 95% time decrease.

The data processing step that ingests queues into a dimension table decreased as well, after rewriting the query to be efficient. The time for that processing action decreased from around 360 seconds to 2 seconds

The aggregation of jobs data also saw a significant decrease in its time to complete. We saw a 50% decrease in the time the XDMoD aggregation process took to run after changes to its table indexes. In some cases, the time decreased by up to 90% for the data aggregation process. Table 2 lists the time it took for data pipelines to complete.

# 5 DISCUSSION

## 5.1 Conversion to InnoDB

As stated in previous sections, the conversion to the InnoDB table engine provided a 33% time performance improvement for the XDMoD Web Portal. This improvement was gained solely through the conversion to InnoDB without any other optimizations. This performance improvement comes from InnoDB's use of clustered indexes on all of its tables. According to the book High Performance MySQL [4], clustered indexes have some very important advantages: related data is kept close together, data access is fast and queries that use covering indexes can use the primary key values contained in the index. In InnoDB, clustered indexes store a B-Tree index and the row data together in the same structure. Accessing a row through the clustered index is fast since the index search leads directly to the page that contains the row data. If a table is large, InnoDB's clustered index architecture often saves a disk I/O operation when compared to storage organizations that store row data using a different page from the index record.

For InnoDB tables, indexes other than the clustered index are known as secondary indexes. Each record in a secondary index contains the primary key columns for the row, as well as the columns specified for the secondary index. InnoDB uses this primary key value to search for the row in the clustered index which makes queries that use secondary indexes quicker as well.

The queries run for the XDMoD Web Portal make extensive use of indexes in order to provide better performance. Since the tables involved in the queries run for the XDMoD Web Portal already have appropriate indexes on them, no changes were needed to take advantage of InnoDB's clustered index's and associated performance gains.

## 5.2 Table Index Modifications

Changes to table indexes provided much of the improvement to XDMoD's data processing pipelines. This section includes more in-depth information about why these index changes help. These improvements were seen most prominently in the data ingestion and aggregation pipelines for XDMoD's Jobs realm.

The addition of a table index that includes the is_deleted, end_day_id, and start_day_id fields, in that order, provided the most significant performance impact. The order of the fields in the index is important because of how indexes work in MySQL, and the data access patterns of the data aggregation pipeline.

Our aggregation query contains the following WHERE clause:

```
WHERE start_day_id <= END_DAY_ID AND end_day_id >=
START_DAY_ID AND is_deleted = 0
```

In this WHERE clause, END_DAY_ID is the last day of records included in the aggregation and START_DAY_ID is the first day of records included in the aggregation.

A typical scenario may be where you are aggregating data for the previous day. An index with the order of deleted_id, start_day_id, end_day_id starts with is_deleted = 0 and then moves to the range query searching for all jobs that start before the current day, which is all the records in the job_tasks table, and then all records that have an end_day_id of today or later. Comparatively, an index of is_deleted, end_day_id, and start_day_id is much more efficient because it uses the end_day_id part of the where clause before the start_day_id part, which means orders of magnitude fewer rows need to be accessed. This is seen in Table 3. It is notable is that when using an index with the order of (start_day_id, end_day_id, is_deleted) the query planner does not use an index.

# 6 CONCLUSION

In this paper we show how converting to the InnoDB database engine can provide a significant performance impact when compared to MyISAM. By switching to InnoDB, we found that queries that read data from the XDMoD data warehouse for viewing in the XDMoD web portal decrease in time by 43% without any other optimizations. For XDMoD's data processing pipelines, switching to InnoDB did not initially provide a significant decrease in the time to ingest data. However, significant performance improvements was gained by adding appropriate indexes to some tables and rewriting queries to be more efficient.

**Table 2: Comparison of Data Pipeline Results**

| Pipeline Name | Pre-index (sec) | Post-index (sec) | Percent Change |
|---|---|---|---|
| hpcdb-xdw-ingest-jobs.HpcdbPostIngestJobUpdates | ~200 | ~5-10 | 95% |
| hpcdb-xdw-ingest-jobs.queue | ~360 | ~2 | 96% |
| jobs-xdw-aggregate.aggregate-days | ~2100 | ~960 | 54% |

**Table 3: Comparison of Rows Accessed Based On Table Index**

| Index Columns | possible_keys | key | rows |
|---|---|---|---|
| start_day_id, end_day_id, is_deleted | fk_resource, aggregation_index, deleted | NULL | 49363243 |
| is_deleted, start_day_id, end_day_id | fk_resource, aggregation_index, deleted | deleted | 24681621 |
| is_deleted, end_day_id, start_day_id | fk_resource, aggregation_index, deleted | deleted | 3096 |

InnoDB's use of a clustered index provides most of these performance improvements. As opposed to MyISAM tables where index data is held in a separate data structure, a clustered index stores index data with row data. This architecture leads to better performance as the index search leads directly to the page that contains the row data. If a table is large, the clustered index architecture often saves disk I/O operations when compared to storage organizations that store row data using a different page from the index record. Because InnoDB uses a clustered index, having appropriate indexes for queries that run against your database is important. Queries that do not have indexes or sub-optimal indexes may not see as much performance improvement when using InnoDB.

Evaluating XDMoD's table indexes and the order of columns in those indexes also provided information on how to improve the performance of XDMoD's data ingestion pipelines. The EXPLAIN EXTENDED command helped us evaluate the appropriate indexing strategy for our tables. As seen in section 5.2, the command showed that changing the order of the columns in an index can help MySQL execute a query that accesses drastically fewer rows to retrieve its data which results in a significant reduction in run time of the queries.

Overall, using the InnoDB table engine for has provided significant performance gains with minimal changes to the XDMoD data warehouse. As data from HPC systems increases in size, these performance improvements will help XDMoD continue to provide its service of supporting the storage and querying of CI data and an interactive web portal that facilitates rapid access to the data for HPC systems.

## 7 FUTURE WORK

During this work, we identified other areas and technologies that may be worth investigating in the future.

- We have not yet fixed the issue with queries involving the Queue Group By. This is something that we are planning to address in a future release of XDMoD.
- Load testing XDMoD. When testing the read performance of XDMoD, we tested in an ideal scenario of queries that involved one group by, one statistic, and a two-week time period on a server with no other traffic. A more real-world scenario with multiple requests for a wide variety of statistics, group by's, and time ranges would be beneficial.

## ACKNOWLEDGMENTS

The authors would like to thank CCR Systems Administration staff Sam Guercio and Andrew Bruno and staff members of the XDMoD development team, including Jeffrey T. Palmer, Ryan Rathsam, Nikolay Simakov and Hannah Taylor and former members Cynthia Cornelius, Jeanette Sperhac, Benjamin Plessinger, Rudra Chakraborty, Steven M. Gallo, Thomas Yearke, Amin Ghadersohi and Ryan Gentner.

This work was sponsored by the National Science Foundation (NSF) under award ACI 1445806 for the XD Metrics Service (XMS).

## REFERENCES

[1] Oracle Corporation. 2022. MySQL 5.7 Reference Manual. https://dev.mysql.com/doc/refman/5.7/en/.

[2] Ralph Kimball and Margy Ross. 2002. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling* (2nd ed.). John Wiley & Sons, Inc., USA.

[3] Jeffrey T. Palmer, Steven M. Gallo, Thomas R. Furlani, Matthew D. Jones, Robert L. DeLeon, Joseph P. White, Nikolay Simakov, Abani K. Patra, Jeanette M. Sperhac, Thomas Yearke, Ryan Rathsam, Martins Innus, Cynthia D. Cornelius, James C. Browne, William L. Barth, and Richard T. Evans. 2015. Open XDMoD: A tool for the comprehensive management of high-performance computing resources. *Computing in Science and Engineering* 17, 4 (2015), 52–62. https://doi.org/10.1109/MCSE.2015.68

[4] B. Schwartz, P. Zaitsev, and V. Tkachenko. 2012. *High Performance MySQL* (3 ed.). O'Reilly, Sebastopol.