

Enspire for Enterprise

**Setting up Percona Xtradb Cluster in Ubuntu 24**

**Prepared By: Shehzaib Mudassar**

**Version: 1.0**

**Date: 13th-June-2025**

CONTENTS

[INTRODUCTION 3](#_Toc177986187)

[1. PERFORMANCE 3](#_Toc177986188)

[2. MEMORY 5](#_Toc177986189)

[3. CLEAN & DIRTY PAGES 8](#_Toc177986190)

[4. LOGICAL I/O 8](#_Toc177986191)

[5. DATA GROWTH 10](#_Toc177986192)

[6. DISK USAGE by TOP TABLES 11](#_Toc177986193)

[7. CPU USAGE 11](#_Toc177986194)

[8. Wait Statistics 12](#_Toc177986195)

[9. OUTAGE CAUSES 17](#_Toc177986196)

[10. REFERECNCES: 20](#_Toc177986197)

# INTRODUCTION

This document is about CarpetOne*, which is* one of the major databases we have on our instance in *Globenetix environment.*

The application was generating alerts, so this is the initial assessment of the database to identify problems and bottlenecks and to give recommendations for its optimal performance.

Each of the following areas were accessed and then recommendations for each parameter have been given.

1. CPU,
2. Disk I/O,
3. Network issue,
4. Memory issues

.

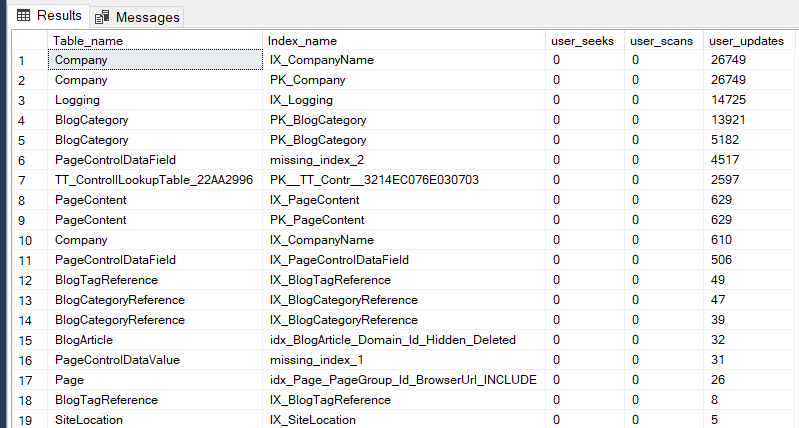
# PERFORMANCE

There are four major parts of SQL Server that needs to be looked for performance issues. Those are

1. *CPU*
2. *Disk I/O*
3. *Memory*
4. *Network*

But before discussing above mentioned parameters in detail, I would like to mention few other points which are as follows:

* Indexesare available in tables required by queries using *where, group by, having, order by* clauses. Index can also resolve the locking issue because transaction will find the required row without going through full table scan, because index will help find a row quickly, so recourses would not be locked for longer time. Following table list all the indexes in our CarpetOne database which was not used neither in seek nor scanned. We can drop the non-cluster index which are not in use leaving unique constraint index because query optimizer considers a ‘uniqueness guarantee’ to perform certain operations, but this is not revealed in index usage statistics. Harm of leaving unused index is the space used by non-cluster index and the increased work for DML statements as they must keep the indexes updated as well as the base table. Many indexes cause overhead for index maintenance during INSERT, UPDATE and DELETE operations. Below table shows unused index.



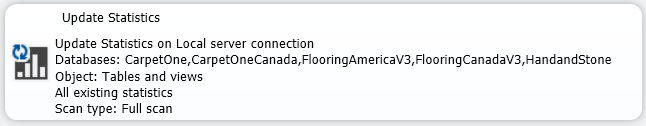
*Please refer to EXCEL Sheet for full list.*

* Statistics are required by query plan. Stats need to be refreshed frequently so that we can have good execution plan, especially on tables which are involved in select, insert, update, and delete operations. It is normally good practice to set *'Auto Update Statistics'* on and set '*Update Statistics Asynchronously'* on. This allows SQL Server to trigger a stats update when it feels it is needed, and the asynchronous stats update means the query that triggered the stats update is not waiting for the update to complete before it returns to the user. Auto Update value is true but asynchronous is false for CarpetOne.

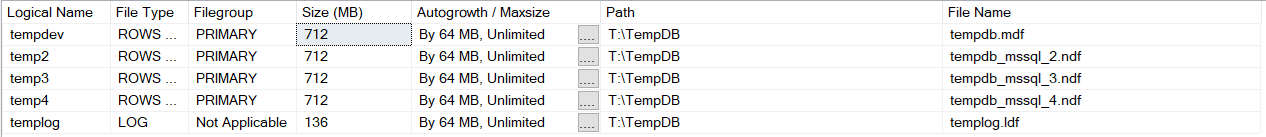


The important thing to note here is Auto update stats fire when 20% of data changes in table or will automatically update the next time a query plan uses it.

Following Maintenance Plan is created for CarpetOne in GNIX. This job is going to update stats of CarpetOne database.



* Avoid large insert, update, delete on tables where indexes are created. If required, drop the index before large insert data set and recreate later.
* *TempDB* database is a temporary working area for SQL Server to do such things as sorting, hash join, online indexing and grouping.  Sizing the tempdb depends upon existing workload, rebuilding indexes, and other factors. General practice is to have 8 datafiles and maintained on separate disk to avoid contention. Datafiles are less in our GNIX environment. Would like to recommend 8 datafiles.



* Fragmentation in the index happens because of data modifications (INSERT, UPDATE, DELETE). This led to an increase in the number of *logical reads* during the query execution, because the index utilizes more data pages to store data. Following Maintenance Plan is created to handle fragmentation issue in CarpetOne.



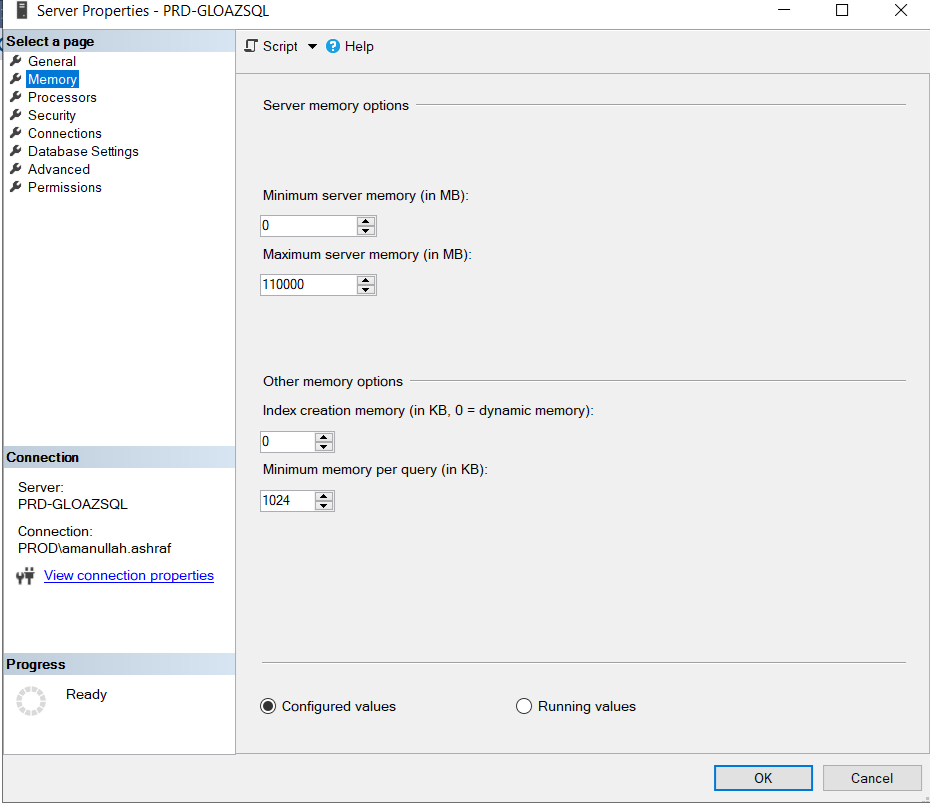
Tables having avg\_fragmentation\_in\_percent >5% and less than 30% are required to run REORGANIZE INDEX command. Tables where avg\_fragmentation\_in\_percent >30% we must REBUILD INDEX. I am rebuilding indexes where fragmentation is > 30%.

**NOTE**:

*Please avoid shrinking datafiles because it causes fragmentation.*

# MEMORY

SQL Server Max Memory needs to be set at 90% of the total memory available. Leaving 10% for the operating system to use. As we have 128GB, we have assigned 110000 MB/128000 MB (85%) to SQL Server and left the rest for OS. We are fine with this setting.



Two other parameters used to identify SQL Server memory issues are *‘cache hit ratio’* and *‘page life expectancy’*. Buffer cache hit ratio is the probability of query finding the requested data in the memory. In page life expectancy, if memory is not enough then page no longer stays in the buffer cache and data needs to be read from the disk which degrades performance. Let’s check these two values for our instance.

**Page Life Expectancy**



cntr\_value= 1574661 seconds is very good. Generally, it is 1000. But should not go down below 300, because below 300 means we have index issue, insufficient memory issues. So, there is no issue regarding staying of pages in memory in our case.

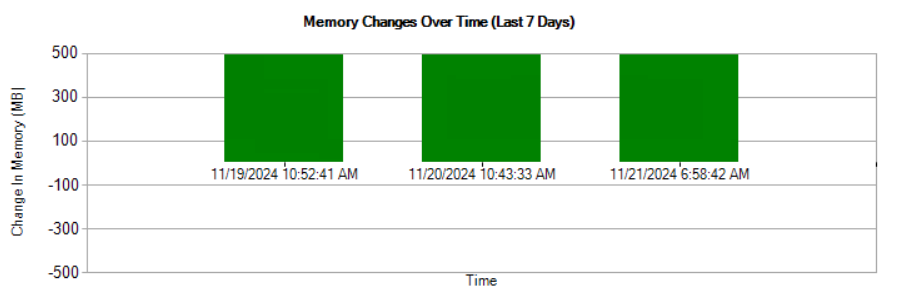
Let’s check the status of cache hit ratio.

**CarpetOne Cache Hit Ratio**

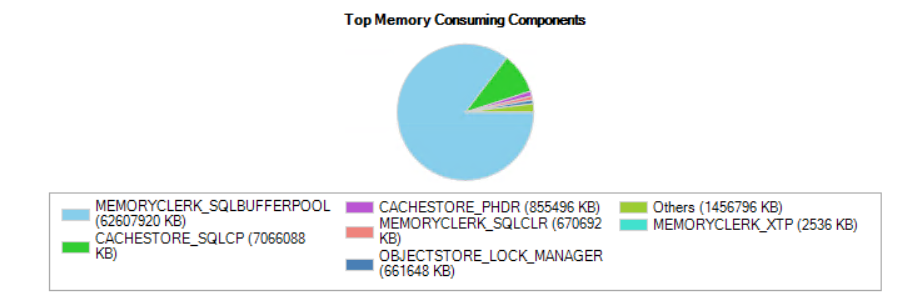


Cache hit ratio (100 %) is excellent, no action required.

**MEMORY USAGE**



Memory usage is fine.

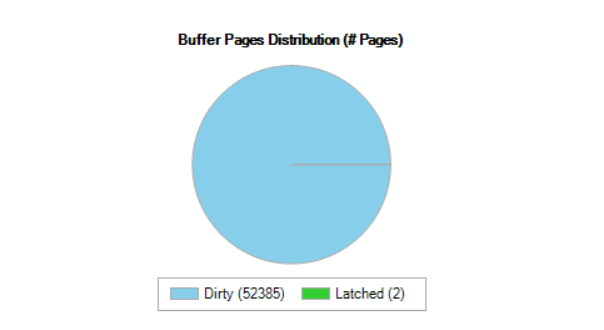


***MEMORYCLERK\_SQLBUFFERPOOL*** is showing all data pages in memory. High value is preferable. That’s why we have high ‘*page life expectancy*’ and high *logical reads* as can be seen from the below pie chart.

***CACHESTORE\_SQLCP*** is storing cache plans for SQL statement which are less to be reused than stored procedure.  This indicates that there are a lot of ad-hoc queries running on the server.

# CLEAN & DIRTY PAGES

Let’s look deeper into buffer cache regarding clean and dirty pages. Cleans pages are those which are not touched by insert/update/delete activity. On the other hand, pages are dirty due to insert/update/delete activity on the databases. Below is the pie chart and table showing the number of clean and dirty pages.





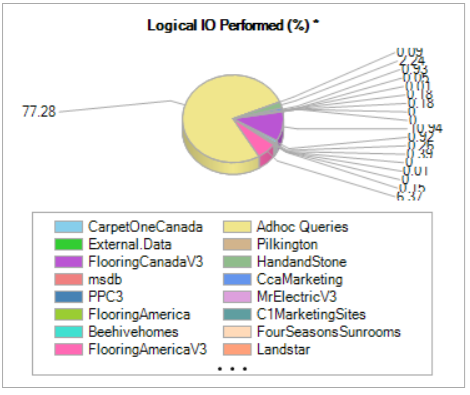
### Recommendations

The following are the recommendations for the dirty pages.

This value (0.05) is towards lower side. We are fine with this value.

# LOGICAL I/O

Logical IO means reading/writing operations performed in memory, rather than physical disk. Below pie chart, which shows the distribution of logical input/output (IO) operations across various databases in GNIX environment.



Major logical I/O portion was used by *Adhoc Queries*. Ad-hoc queries are created and executed dynamically or on-the-fly, rather than being pre-defined and stored in the database system. CarpetOne is not in the list, indicates its portion of logical IO is negligible.

### Recommendations

1. Developers can use stored produces instead of Ad-hoc queries if possible because stored procedure can be reused but SQL\_Query\_Plan might not be reused and might washed out from query plan.
2. Run the follow statement to flush the ad hoc plans (this will not flush store procedure plans):

USE master;

GO

DBCC FREESYSTEMCACHE ('SQL Plans');

GO

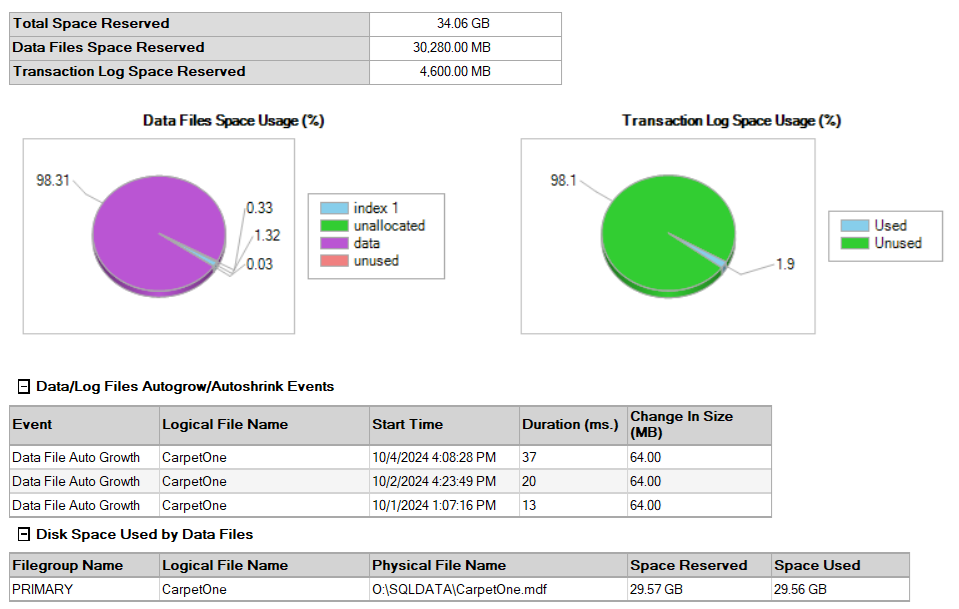
Secondly, we need to set ‘*Optimize for Ad hoc Workloads*’ parameter value to be ‘true’ from the server property (Advanced) settings.



This will allow SQL Server Engine to store small compile plan instead of full plan. This will release memory pressure by not allowing to store plan in cache that are not to be used again.

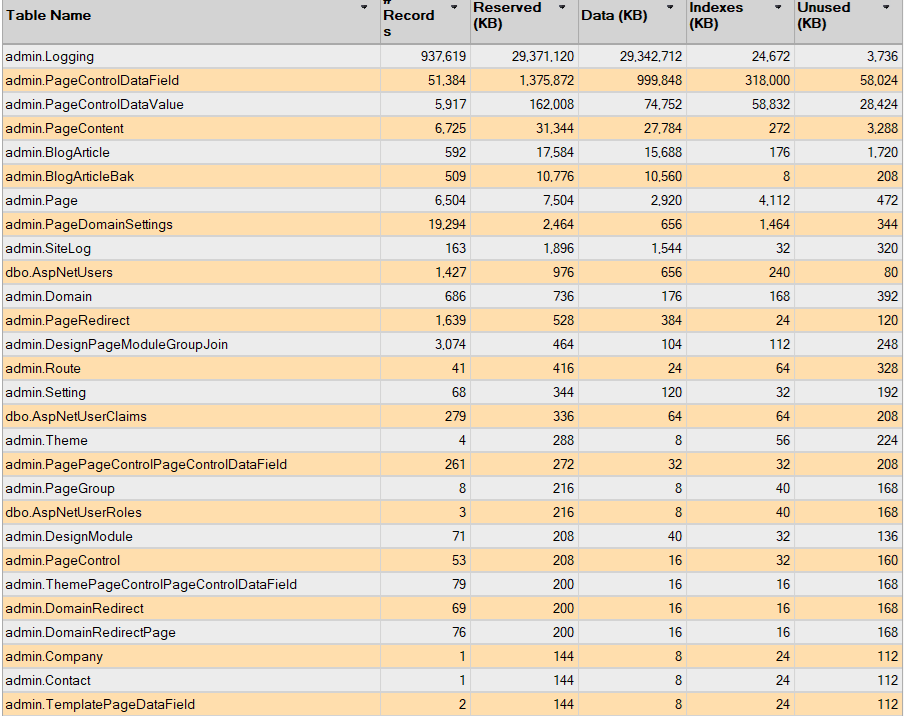
# DATA GROWTH

CarpetOne *Data & Log file* usage



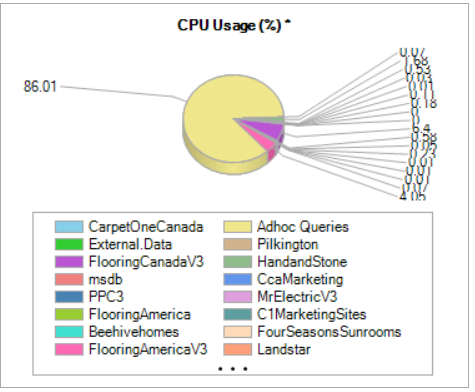
This datafile is almost filled with 99% usage. There are approximately 3 auto growth activity happened against the datafile. As auto growth is a resource consuming process, instead of 64 MB, it should be increased to 150 MB in a go.

# DISK USAGE by TOP TABLES



# CPU USAGE

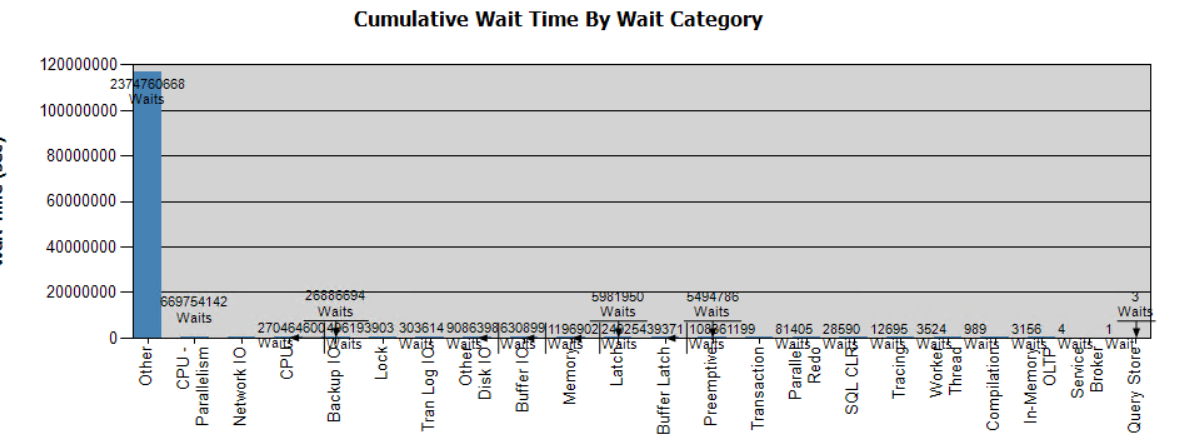
If CPU is below 80% of usage, that means we are fine. As Adhoc Queries are using 75% of CPU that means we are fine but we need to tune our queries.

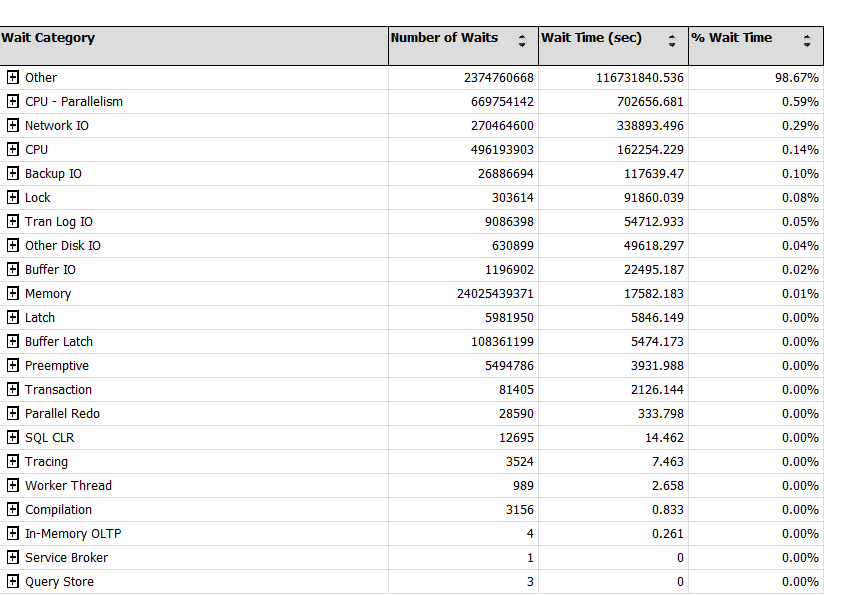


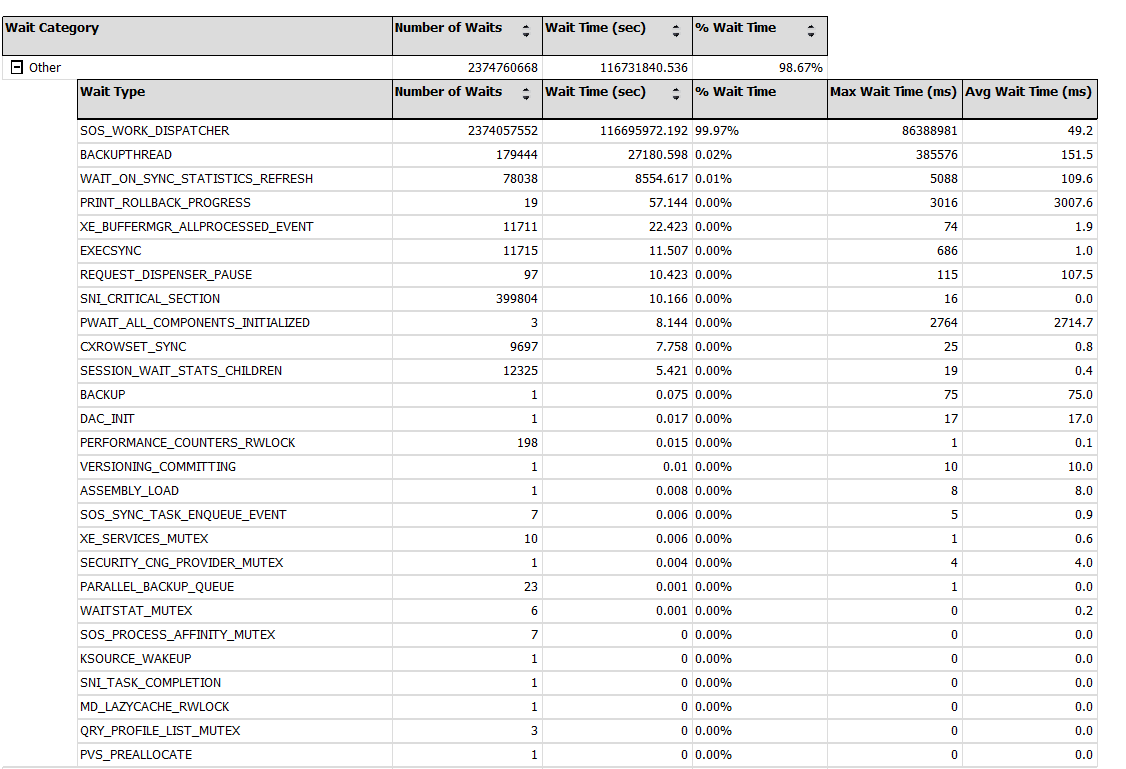
CPU spent most of the time on ad-hoc queries as can be seen from the above pie chart. Its most probably used by ‘CarpetOne’ databases as its CPU time is highest among the databases.

# Wait Statistics

Wait statistics are one of the most important indicators to identify performance issues in SQL Server. Following are the important parameters.







**SOS\_WORK\_DISPATCHER**

This wait is generally related to worker threads in SQL Server being dispatched for tasks. A high number could indicate contention or insufficient resources (like CPU).

#### **Recommendation**

:

* Add more CPU cores.
* Review queries for efficiency; optimize expensive or long-running queries.
* Increase the max degree of parallelism (MAXDOP) if appropriate.

**BACKUPTHREAD**

This wait is often seen during database backups. The large max wait time suggests backups may be affecting performance.

#### **Recommendation**

* Schedule backups during off-peak hours.
* Optimize backup strategies (e.g., use differential or incremental backups).
* Ensure the storage subsystem can handle the I/O demands of backups.
* Use less storage data type.

**WAIT\_ON\_SYNC\_STATISTICS\_REFRESH**

This occurs when automatic statistics updates are happening during query execution, and the query has to wait for the statistics refresh to complete before it can continue. This can impact query performance, especially if the query optimizer determines that outdated statistics need to be updated before executing a query.

#### **Recommendation**

Already created a task in SQL Agent for this purpose.

**ASYNC\_NETWORK\_IO**



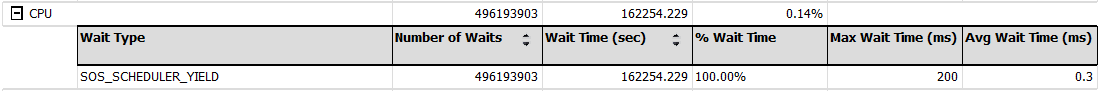
This wait type occurs when SQL Server is waiting for the client to acknowledge the data it has sent. This is often an indication of slow network speed or slow client processing. ASYNC\_NETWORK\_IO somehow indicates that the client application isn't processing results as fast as SQL Server sends them. This could be caused by an issue with the client application or with the network connection between the server and the client application. The ASYNC\_NETWORK\_IO wait indicates that one of two scenarios are happening. The first scenario is that the session is waiting for the client application to process the result set and send a signal back to SQL Server that it is ready to process more data. The second is that there may be a network performance issue.

#### **Recommendation**

:

* Check for network latency between the SQL Server and clients. If high latency is observed, network optimizations may be needed.
* Ensure clients consuming data are efficient and can process the data fast enough.
* Consider reducing the size of result sets if possible or implement more efficient data-fetching strategies on the client side.
* Check whether the application is requesting large data sets from a SQL Server instance, and then if it filters those data on the client side. Secondly, creating views, will therefore significantly reduce amount of data that will be sent to the client application. To reduce the ASYNC\_NETWORK\_IO, it is necessary to commit transaction in a timely manner and use *where* clause in a query if possible, to restrict the result set. Beside that check network bandwidth between the SQL Server and client.

**SOS\_SCHEDULER\_YIELD**



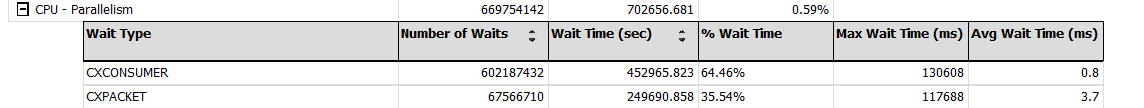
This wait indicates that a worker thread was yielding the CPU to other tasks.

#### **Recommendation**

:

* Adding more CPUs/cores.
* Tune the queries that consume the most CPU.
* Adjust the **Max Degree of Parallelism (MAXDOP)** setting.

**CXPACKET/CXCONSUMER**

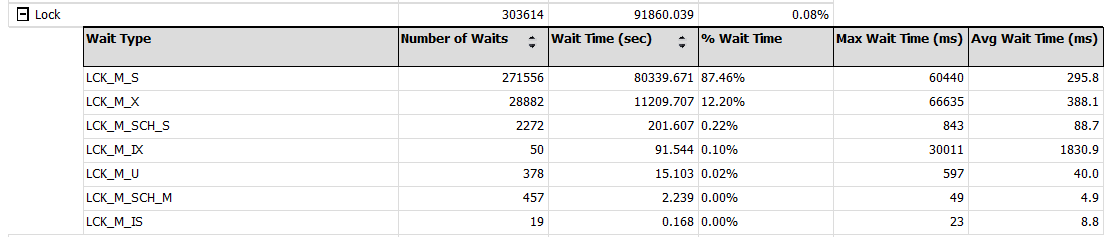


CXPACKET and CXCONSUMER waits are related to parallel query execution. When a parallel operation is created for SQL Query, there are multiple threads for a single query. Each query deals with a different set of the data (or rows). Due to some reasons, one or more of the threads lag behind, creating the CXPACKET Wait Stat. High values might indicate inefficiencies with parallel processing.

#### **Recommendation:**

* Adjust **Max Degree of Parallelism (MAXDOP)** setting. Often setting MAXDOP to a value based on the number of CPU cores (e.g., MAXDOP = number of physical cores minus 1) can help balance the workload.
* Reduce parallelism for smaller queries by setting the Cost Threshold for Parallelism to a higher value, which ensures only more resource-intensive queries run in parallel.
* Optimize queries to avoid unnecessary parallel execution by tuning indexes, reviewing query logic, and examining execution plans.

**LOCK**



LCK\_M\_S: This wait occurs when a request is waiting to acquire a shared lock. This typically happens when read requests are blocked by write transactions that have been kept open for long period of time.

LOCK\_M\_X: Occurs when a task is waiting to acquire an Exclusive lock. This exclusive lock is used by SQL Server when a transaction modifies data (INSERT, UPDATE, DELETE). It prevents other transactions from reading or modifying the locked resource until the lock is released

#### **Recommendation**

:

* Ensure that queries are properly indexed so that they only lock the necessary rows, not entire tables or large ranges of data.
* Use appropriate isolation levels. In some cases, you can use READ\_COMMITTED\_SNAPSHOT or SNAPSHOT isolation to reduce locking contention by using row versioning.
* Avoid long-running transactions. The longer a transaction holds locks, the more likely it is to cause blocking for other transactions.
* Break up large batch updates into smaller chunks to reduce the time locks are held.
* Use **appropriate indexes** to narrow the lock scope. With more targeted locks (e.g., locking specific rows instead of full table or page locks), you can reduce contention.
* Use the **NOLOCK** hint (or **READ UNCOMMITTED** isolation level) for read operations where it's acceptable to read uncommitted data.

# SUGGESTIONS

1. **Data type size is very important**. One of the major aspects affecting query performance is the amount of I/O involved. A query that reads less simply tends to run faster. The bigger the type that you use, the more storage it uses. Please use the smallest type that serves your needs. For example, nvarchar (MAX) can be replaced with nvarchar (255) etc.
2. **Partitioning big tables will help**. In order to improve query scan performance and lock wait time, we can consider partitioning ‘admin.Logging’ and ‘admin.PageControlDataField’ tables. As these tables are big in size. Partitioning will remain transparent to end user.
3. **Denormalized Schema.** It’s aprocess of optimizing a database schema by introducing redundancy to improve query performance. ‘admin.Domain’ and ‘admin.Page’ tables are involved in many Primary and Foreign key relationships. If we denormalized these two tables by creating a one single table, then this will reduce the need for joins. *Please refer to ERD document.*
4. **Enable adhoc queries:** This will store small compile plan instead of full plan. This will release memory pressure by not allowing to store plan in cache that are not to be used again.



1. **Use (NOLOCK) table hint:** This query hint is used to specify that a SELECT statement should not issue shared locks and, therefore, can read uncommitted data. This hint is commonly used to improve performance in read-heavy systems where data consistency is less of a concern. Since NOLOCK prevents locking, it helps reduce contention between transactions. For example:

*SELECT column1 FROM table\_name WITH (NOLOCK);*

1. **Rewrite CPU consuming queries:** Instead of SELECT, use CTE and then select from CTEs; For example:

*CREATE TABLE #ProductColorIDs (ProductColorID BIGINT);*

*INSERT INTO #ProductColorIDs (ProductColorID)*

*VALUES (291786150027319881), (291786150027320010),….. (329146400027919195);*

*SELECT*

*p.DivNumber,*

*p.MemberId,*

*p.LocationID,*

*p.Location*

*FROM*

*PriceFileData AS p*

*INNER JOIN*

*#ProductColorIDs AS pc ON p.ProductColorID = pc.ProductColorID*

*WHERE #*

*p.DivNumber = @\_\_division\_0*

*AND p.Location = @\_\_locationNumber\_1;*

Make sure the columns used in the WHERE clause (DivNumber, Location, ProductColorID) are indexed.

1. **Change MaxDOP and parallelism value at instance level**: This setting applies to *all databases and queries* running on the SQL Server instance, unless overridden by database or query-specific settings. It governs the **maximum number of CPU cores** that can be used to execute a single query in parallel across the entire instance. SQL Server will use multiple CPU threads up to the value set for MaxDOP.

*MaxDOP = 16*

*Cost Threshold for Parallelism = 40*

1. **Use MaxDOP table hint:** This setting controls the number of processors used for the parallel execution of queries in SQL Server. For example:*SELECT \* FROM Orders OPTION (MAXDOP 2);*
2. **Use smaller transaction**: Insert with multiple values in a single statement, is generally faster than executing multiple individual insert statements. For example*: INSERT INTO table (columns) VALUES (value1), (value2), (value3)* is faster than multiple individual separate insert values statements.
3. **Checkpoint:** When dirty pages that have been modified in memory but not yet written to disk.Checkpoints write dirty buffers to disk.
4. **Enable Asyn stat update:** This will update stats during query execution process without waiting for query to execute first.
5. **Use SP instead of Queries:** SQL Server caches the execution plan. This plan is reused each time the stored procedure runs, reducing the overhead of query parsing and optimization.
6. **Adding more CPUs/cores for better parallelism job.**
7. **Set auto growth to 150 MB for data files**
8. **Create Index on column used in where, join, group by conditions.** Avoid large insert, update, delete on tables where indexes are created. Create indexes on columns used in above mentioned clause. Delete Unused indexes in order lower down maintenance cost. Please refer to excel sheet for unused index details.

# REFERECNCES:

* SQL Server CarpetOne database Reports & DMVs
* <https://www.sqlshack.com/>
* <https://dba.stackexchange.com/>
* <https://www.sqlservercentral.com/>