Sample Azure Database Performance Report

ACMEGizmo

Public

Node4 Limited

NODE4
Empowering business to do more

Contents

Proprietary Notice

Document Change Details

Version	Date	Author	Role	Changes
v1.0	Today	D.B Expert	Senior DBA	First Draft

1 Summary

1.1 Process Overview

AcmeGizmo SQL Azure database server and a sub-set of three databases (Mydevice_Marketing, Events_Marketing, and SEOManagement_Marketing) as identified on the completed Node4 Scope of Works have been reviewed to help troubleshoot ongoing performance issues. From the review, a set of recommendations have been created that have been categorised into server level recommendations and database specific service optimisation / best practice.

1.2 Diagnostics

The table below summarises the key steps that have been taken to provide this report:

Step	Remarks
Diagnostic Scripts	RDBMS specific diagnostic scripts
Backup Analysis	Database backup analysis
Database Server	RDBMS specific database instance configuration review
High Availability	RDBMS specific review of any HA features used
Performance Baseline	Azure SQL Analytics and Query Store
Analysis	Review and analysis of collected metadata and outputs

1.3 Executive Summary

This report identified several configuration issues at both the Server level and individual database level. At the server level it has been identified that migrating to SQL Azure Always-ON failover group, SQL Azure elastic pool, Elastic jobs and SQL Azure ATP (Advanced Threat Protection) could improve service resilience, security and performance. There may also be an opportunity to help reduce costs by scaling up and down resource automatically based on the known workload demand/usage patterns.

At the database level, it was recognised that there are numerous opportunities to improve current performance particularly with the Mydevice_Marketing database. Such optimisations include; database compatibility settings, configuration of auto tuning options, specific index optimisations, remediating service impacting issues around blocking/locking, index maintenance, saving resources and improving performance through data compression, column-store indexes and other specific in-memory OLTP optimisations.

We would welcome the opportunity to support any agreed remediation steps.

1.4 Next Steps

Below are the suggested next steps;

Step	Remarks
Report Review	Discuss and agree findings and remediation approach
Implementation	Implementation of agreed specific changes
Validation	Incremental implementation and review of performance

2 Server Level Recommendations

2.1 ACME-SQLSRV-PROD

Risk	Previous Rec?	Amended?	Observation	Suggested Action
H	No	No	Consider migrating to SQL Azure Elastic pool in one Standard (or Premium – if using in-memory recommendations below) plan to better utilise purchased DTUs. Consolidating databases into an elastic pool allows un-used DTUs to be shared across multiple databases within the elastic pool. Its recommended that an elastic pool is considered once the higher impact recommendations detailed in this report have been considered and implemented. Running an elastic pool on a Premium service plan may be cost effective whilst unlocking the in-memory performance benefits identified below	Once the high impact recommendations have been implemented and performance (particularly with Mydevice_Marketing database) has improved, calculate the eDTUs required; MAX(<Total number of DBs X average DTU utilization per DB>, <Number of concurrently peaking DBs X Peak DTU utilization per DB) Calculate the storage requirement (implementing the compression and column-store indexes should reduce the storage requirement) for all candidate databases. Then determine the eDTU pool size that provides the required amount of storage. Finally, take the larger of the eDTUs required v's the storage requirement as the initial sizing for the elastic pool
H	No	No	Enable SQL Azure ATP (Advanced Threat Protection) for all databases. ATP provides; audit data (using log analytics), sensitive data management (detects tables with sensitive data), security baseline (50 checks), alerts via email for threats/breaches in real-time, SQL security insights dashboard	SQL Azure ATP is free for the first month and then chargeable thereafter. Given GDPR requirements, its strongly recommended that this service is implemented
M	No	No	Active Geo-replication enabled, but databases are not in an automatic failover group. If appropriate (consider the architectural design prior), configure automatic failover groups	Promote the databases into automatic failover groups. Note that Elastic pool also supports automatic failover groups and active geo-replication. Note: DTUs allocated should match across primary and secondary elastic pools
M	No	No	Ensure that App tier connection strings are correctly updated to use the SQL Azure Failover Group read/write connection end-point rather than individual server connections.	Check and validate App tier connection strings and update as required
M	No	No	Ensure that App tier connection strings are correctly updated to use the SQL Azure Failover Group read-only connection end-point rather than individual server connections. Off-loading reporting style workload to the asynchronous read-only database copy can help alleviate heavy workload / contention on the primary read/write node.	Check and validate App tier connection strings and update as required. Stored proc sp_wait_for database_copy_sync may be used to ensure committed transaction is received by the secondary prior to releasing the thread
M	No	No	Configure Elastic Jobs for Database Maintenance	Dependent on adoption of Elastic Pool
M	No	No	Enable monitoring and configure an alert to trigger auto-scaling of the elastic pool. Consider down-scaling at quieter times e.g. overnight and at weekends	Dependent on adoption of Elastic Pool OR can up/down scale individual databases (outside of an elastic pool) based on alerting or time-based scheduling
M	No	No	Enable Database Automatic tuning options at the server level and inherit down to database level	Server settings may be overridden at the database level as required

3 Database Level Recommendations

3.1 Mydevice_Marketing

Risk	Previous Rec?	Amended?	Observation	Suggested Action
H	No	No	Service impacting incidents recorded.	Please refer to Appendix B for further information
H	No	No	Query Store Analysis: Large index FULL scans processing > 2m records	Please refer to Appendix A and Appendix C for further information
H	No	No	Backup retention set to Point-In-Time for 35 days. Consider setting weekly, monthly, and annual backup retention policies based on organisation data retention policies. Reduce the PIT retention down to what is required e.g. 3 days	Implement once Recovery Point objectives clarified
H	No	No	Lock waits on indexes (as at “Today”) pk_Issues (Issues) 106 lock events, 73 secs wait uc_Issues_Uid (Issues) 3 lock events, 4 secs wait IX_ActorUid (AuthenticatedSessions)	See Appendix B for further information
M	No	No	Auto Tuning Options (partially enabled – create indexes only). Its recommended that all three SQL Azure tuning options are enabled to allow for plan optimisation, index creation and index deletion. The Auto Tuning advisor uses machine learning to determine the best optimisations and implements these optimisations during quite times only on the database. It also evaluates how successful the optimisation was and can back-out an optimisation if it was not successful in improving performance. There is a 2 hour back-off limit for implementations	Enable all three auto-tuning settings (force plan, create index, drop index). If indexing divergence is not acceptable, then disable create/drop index options and ensure that recommendations are regularly reviewed, evaluated and factored into future agile development cycles
M	No	No	Database compatibility level is set to 120 (SQL 2014). The current default compatibility level is 140 (SQL 2017). Compatibility level 140 enables improved query memory grants (more accurate), join optimisation (based on runtime row counts), and interleaved execution that improves performance of queries using table-based functions. Compatibility level 140 includes improvements enabled with Compatibility 130. These comprise; multi-threaded insert statement (or parallel plan), parallelism with memory-optimised tables, cardinality estimation improvements, and more aggressive stats collection on large tables via multi-threaded process	Smoke (or regression test) a copy of the production dataset on a test database with compatibility level 140 set; ALTER DATABASE [Mydevice_Marketing] SET COMPATIBILITY_LEVEL = 140; GO Ideally, a full stress test with benchmarking would be appropriate, but since the compatibility level may be set back down dynamically, the risk associated with not performing a full set of non-functional tests is mitigated. As such, this dynamic change may be applied to production with limited non-functional performance testing

M	No	No	Parameter [Auto Update Statistics Asynchronously] is set to FALSE. Given the size of the database and some of objects within, it might prove beneficial to enable this parameter (set to TRUE) when revising the database compatibility level considering the more aggressive auto update statistics sampling	ALTER DATABASE [Mydevice_Marketing]
				SET
				AUTO_UPDATE_STATISTICS_ASYNC ON;

Risk	Previous Rec?	Amended?	Observation	Suggested Action
M	No	No	Heap tables detected SEO.IssuesPriority and mm.StatusChanges	Consider adding a clustered index on these tables to avoid any potential sort activities
M	No	No	Regular stats and index maintenance and user defined stats detected	Its recommended to use an intelligence statistics and index maintenance solution such the Node4solution that is based on Ola Hallengren's open source solution. Such maintenance is best scheduled daily on SQL Azure using Elastic Jobs
M	No	No	Buffer memory consumption: dm.DocumentContent (47GB out of 56GB) SEO.IssuesNotes (1.4GB out of 56GB) SEO.Issues (1.3GB out of 56GB) SQL Server offers Row compression (compression of fixed-width columns not fully populated) or Page compression (like row but with repeating pattern compression). Careful use of compression can; reduce the amount of disk space consumption, improve i/o performance (more data, less blocks), and improve cache performance (data is compressed/uncompressed in L2 cache and then fed to/from the SQL Buffer pool)	Consider using Page or Row compression on selected indexes (clustered indexes included). Indexes with high data volatility should be avoided. Compression is most suited to read biased objects with less writes. Varbinary(max) is supported (dm.DocumentContent) but nvarchar(max) data types and heap tables are not currently supported
M	No	No	High average_read_stall_ms (183.5ms). Important as database is read biased (97%) versus write (3%)	Consider implementing the in-memory (see Appendix C) and data compression recommendation to reduce physical disk read overhead
M	No	No	Wait events: ASYNC_NETWORK_IO (55%), SOS_SCHEDULER_YIELD (13%), PAGEIOLATCH_SH (12%)	Consider implementing the in-memory (see Appendix C) recommendations to reduce locking and blocking overheads
M	No	No	New candidate indexes: [Mydevice_Marketing].[SEO].[UserAttachments].[DocumentMetaDataId] [Mydevice_Marketing].[dash].[Dashboards].[TenantUId] [Mydevice_Marketing].[SEO].[Exports]. [UId] [Mydevice_Marketing].[SEO].[ReferralAgencies].[IsShared]	Its recommended to review and evaluate these index recommendations using hypothetical indexes prior to full implementation
M	No	No	Top logical reads: SEO.usp.UserTransfers_Filter / usp_UserTransfers-FilterMany Top physical reads: usp.UserTransfers_Filter	Please review the recommendations in Appendix A and Appendix C
M	No	No	Indexes with no read activity: IX_DateDeletedWithRefs (NodeReference) IX_DateDeleted (NodeReference) IX_NoteTypeId (ConcernNotes) IX_DateCreated (ConcernNotes) IX_IssuesStatus_ProdileId (IssuesStatus) IX_OccurenceDateTime (Issues) IX_DateCreated (Issues) uc_Issues_ReferenceNumber_VenueId (Issues) uc_Users_PersonUId (Users) IX_UsersStatus_UserId (UsersStatus) uc_ExternalIdentities_SourceTypeId_ExternalUId (ExternalIdentities)	Its recommended to review and evaluate these suggested index drops in case they are required. Also – consider enabling Auto Tuning option drop index

Risk	Previous Rec?	Amended?	Observation	Suggested Action
L	No	No	QUERY_OPTIMIZER_HOTFIXES not enabled. This database level scoped parameter controls the usage of the latest optimizer-related hotfixes released after SQL Server 2016 RTM independently of the current database compatibility setting	-- Enable query optimizer fixes for Primary database ALTER DATABASE SCOPED CONFIGURATION SET QUERY_OPTIMIZER_HOTFIXES = ON; GO -- Enable query optimizer fixes for Secondary database ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY SET QUERY_OPTIMIZER_HOTFIXES = ON; GO
L	No	No	MAXDOP unlimited. The optimiser can parallelise all queries across all schedulers where the cost threshold (default 5) is exceeded. It can be useful to cap the degree of parallelism where excessive parallelism and locking waits are detected with high query latency. Currently, there is no evidence to justify capping parallelism, but should this become an issue, suggested settings are provided	Unable to adjust Cost Threshold for parallelism so can cap MAXDOP; -- Set MAXDOP for Primary database ALTER DATABASE SCOPED CONFIGURATION SET MAXDOP = 8; GO -- Set MAXDOP for Secondary database(s) ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY SET MAXDOP = 8; GO
L	No	No	High VLFs – 2475 (4 vlfs per 16mb growth increment). This could cause a performance degradation on inserts, updates and deletes	The default auto growth size for the transaction log is 16mb which is arguably too small for this database workload. Unfortunately, it's not possible to change this setting or resize the transaction log in SQL Azure without dropping and recreating the database. Other performance recommendations will help mitigate this current constraint
L	No	No	PLE 50 minutes, 56GB memory, 12 cpu/sched, avg load 20 tasks per sched, avg task count 14	Other performance recommendations will help increase PLE and thus buffer cache hits

3.2 Events_Marketing

Risk	Previous Rec?	Amended?	Observation	Suggested Action
H	No	No	Query Store Analysis: Large index FULL scan processing > 3m records	Please refer to Appendix C for further information
H	No	No	Lock waits on indexes (as at 31st Jan) uc_EventLog_TenantUId_AggregateUId_AggregateVersion (ad.EventLog) 10 lock events, 20 secs wait	Please refer to Appendix C for further information
H	No	No	Backup retention set to Point-In-Time for 35 days. Consider setting weekly, monthly, and annual backup retention policies based on organisation data retention policies. Reduce the PIT retention down to what is required e.g. 3 days	Implement once Recovery Point objectives clarified
M	No	No	Auto Tuning Options (partially enabled – create indexes only). Its recommended that all three SQL Azure tuning options are enabled to allow for plan optimisation, index creation and index deletion. The Auto Tuning advisor uses machine learning to determine the best optimisations and implements these optimisations during quite times only on the database. It also evaluates how successful the optimisation was and can back-out an optimisation if it was not successful in improving performance. There is a 2 hour back-off limit for implementations	Enable all three auto-tuning settings (force plan, create index, drop index). If indexing divergence is not acceptable, then disable create/drop index options and ensure that recommendations are regularly reviewed, evaluated and factored into future agile development cycles
M	No	No	Regular stats and index maintenance and user defined stats detected	Its recommended to use an intelligence statistics and index maintenance solution such the Node4solution that is based on Ola Hallengren's open source solution. Such maintenance is best scheduled daily on SQL Azure using Elastic Jobs
M	No	No	High average_read_stall_ms (559.8ms).	Consider implementing the in-memory (see Appendix C) and data compression recommendation to reduce physical disk read overhead
M	No	No	Wait events: PAGEIOLATCH_SH (38%), LOG_RATE_GOVERNOR (26%), ASYNC_NETWORK_IO (11%)	Consider implementing the in-memory (see Appendix C) and suggested remediation (in Appendix B) recommendations to reduce locking and blocking overheads
M	No	No	Top logical reads: g.usp.EventLog_GetPageByTenant Top physical reads: *.usp_EventLog_GetPageByAggregate	Please review the recommendations in Appendix C
M	No	No	Buffer memory consumption: Ad.Blobs (4.5GB out of 16GB) *.EventLog (5.5GB out of 16GB) SQL Server offers Row compression (compression of fixed-width columns not fully populated) or Page compression (like row but with repeating pattern compression). Careful use of compression can; reduce the amount of disk space consumption, improve i/o performance (more data, less blocks), and improve cache performance (data is compressed/uncompressed in L2 cache and then fed to/from the SQL Buffer pool)	Consider using Page or Row compression on selected indexes (clustered indexes included). Indexes with high data volatility should be avoided. Compression is most suited to read biased objects with less writes. Varbinary(max) is supported (ad.Blobs) but nvarchar(max) data types and heap tables are not currently supported

Risk	Previous Rec?	Amended?	Observation	Suggested Action
M	No	No	Low DTU usage	Current database service plan under-utilised. Recommend migrating to SQL Azure Elastic pool
L	No	No	QUERY_OPTIMIZER_HOTFIXES not enabled. This database level scoped parameter controls the usage of the latest optimizer-related hotfixes released after SQL Server 2016 RTM independently of the current database compatibility setting	-- Enable query optimizer fixes for Primary database ALTER DATABASE SCOPED CONFIGURATION SET QUERY_OPTIMIZER_HOTFIXES = ON; GO -- Enable query optimizer fixes for Secondary database ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY SET QUERY_OPTIMIZER_HOTFIXES = ON; GO
L	No	No	MAXDOP unlimited. The optimiser can parallelise all queries across all schedulers where the cost threshold (default 5) is exceeded. It can be useful to cap the degree of parallelism where excessive parallelism and locking waits are detected with high query latency. Currently, there is no evidence to justify capping parallelism, but should this become an issue, suggested settings are provided	Unable to adjust Cost Threshold for parallelism so can cap MAXDOP; -- Set MAXDOP for Primary database ALTER DATABASE SCOPED CONFIGURATION SET MAXDOP = 4; GO -- Set MAXDOP for Secondary database(s) ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY SET MAXDOP = 4; GO
L	No	No	High VLFs – 1097 (4 vlfs per 16mb growth increment). This could cause a performance degradation on inserts, updates and deletes	The default auto growth size for the transaction log is 16mb which is arguably too small for this database workload. Unfortunately, it's not possible to change this setting or resize the transaction log in SQL Azure without dropping and recreating the database. Other performance recommendations will help mitigate this current constraint

3.3 SEOManagement_Marketing

Risk	Previous Rec?	Amended?	Observation	Suggested Action
H	No	No	Backup retention set to Point-In-Time for 35 days. Consider setting weekly, monthly, and annual backup retention policies based on organisation data retention policies. Reduce the PIT retention down to what is required e.g. 3 days	Implement once Recovery Point objectives clarified
M	No	No	Wait events: PAGEIOLATCH_SH (58%), WRITE_LOG (32%), ASYNC_NETWORK_IO (4%)	Consider implementing the in-memory (see Appendix C) recommendations to reduce locking and blocking overheads
M	No	No	New candidate indexes: SEOManagement_Marketing].[sm].[YearGroups].[LearningCentreId] [SEOManagement_Marketing].[sm].[RegistrationGroups]. [LearningCentreId]	Its recommended to review and evaluate these index recommendations using hypothetical indexes prior to full implementation
M	No	No	Top logical reads: usp_Apprentices_FilterByWildcard Top physical reads: usp_Apprentices_GetByIds	Please review the recommendations in Appendix C
M	No	No	Indexes with no read activity: IX_SourceId (Contacts) Uc_Contacts_Uid (Contacts) IX_ForApprenticeId (Contacts) uc_Apprentices_UPN_LearningCentreId (Apprentices)	Its recommended to review and evaluate these suggested index drops in case they are required. Also – consider enabling Auto Tuning option drop index
M	No	No	Buffer memory consumption: dm.DocumentContent (24GB out of 28GB) sm.ApprenticeAttendances (0.5GB out of 28GB) SQL Server offers Row compression (compression of fixed-width columns not fully populated) or Page compression (like row but with repeating pattern compression). Careful use of compression can; reduce the amount of disk space consumption, improve i/o performance (more data, less blocks), and improve cache performance (data is compressed/uncompressed in L2 cache and then fed to/from the SQL Buffer pool)	Consider using Page or Row compression on selected indexes (clustered indexes included). Indexes with high data volatility should be avoided. Compression is most suited to read biased objects with less writes. Varbinary(max) is supported (dm.DocumentContent) but nvarchar(max) data types and heap tables are not currently supported
M	No	No	Regular stats and index maintenance and user defined stats detected	Its recommended to use an intelligence statistics and index maintenance solution such the Node4solution that is based on Ola Hallengren's open source solution. Such maintenance is best scheduled daily on SQL Azure using Elastic Jobs
M	No	No	Database compatibility level is set to 120 (SQL 2014). The current default compatibility level is 140 (SQL 2017). Compatibility level 140 enables improved query memory grants (more accurate), join optimisation (based on runtime row counts), and interleaved execution that improves performance of queries using table-based functions. Compatibility level 140 includes improvements enabled with Compatibility 130. These comprise; multi-threaded insert statement (or parallel plan), parallelism with memory-optimised tables, cardinality estimation improvements, and more aggressive stats collection on large tables via multi-threaded process.	Smoke (or regression test) a copy of the production dataset on a test database with compatibility level 140 set; ALTER DATABASE [SEOManagement_Marketing] SET COMPATIBILITY_LEVEL = 140; GO Ideally, a full stress test with benchmarking would be appropriate, but since the compatibility level may be set back down dynamically, the risk associated with not performing a full set of non-functional tests is mitigated. As such, this dynamic change may be

Risk	Previous Rec?	Amended?	Observation	Suggested Action
				applied to production with limited non-functional performance testing

M	No	No	Parameter [Auto Update Statistics Asynchronously] is set to FALSE. Given the size of the database and some of objects within, it might prove beneficial to enable this parameter (set to TRUE) when revising the database compatibility level considering the more aggressive auto update statistics sampling	ALTER DATABASE [SEOManagement_Marketing]
				SET
				AUTO_UPDATE_STATISTICS_ASYNC ON;

L	No	No	QUERY_OPTIMIZER_HOTFIXES not enabled. This database level scoped parameter controls the usage of the latest optimizer-related hotfixes released after SQL Server 2016 RTM independently of the current database compatibility setting	-- Enable query optimizer fixes for Primary database ALTER DATABASE SCOPED CONFIGURATION SET QUERY_OPTIMIZER_HOTFIXES = ON; GO -- Enable query optimizer fixes for Secondary database ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY SET QUERY_OPTIMIZER_HOTFIXES = ON; GO
L	No	No	MAXDOP unlimited. The optimiser can parallelise all queries across all schedulers where the cost threshold (default 5) is exceeded. It can be useful to cap the degree of parallelism where excessive parallelism and locking waits are detected with high query latency. Currently, there is no evidence to justify capping parallelism, but should this become an issue, suggested settings are provided	Unable to adjust Cost Threshold for parallelism so can cap MAXDOP; -- Set MAXDOP for Primary database ALTER DATABASE SCOPED CONFIGURATION SET MAXDOP = 4; GO -- Set MAXDOP for Secondary database(s) ALTER DATABASE SCOPED CONFIGURATION FOR SECONDARY SET MAXDOP = 4; GO
L	No	No	High VLFs – 790 (4 vlfs per 16mb growth increment). This could cause a performance degradation on inserts, updates and deletes	The default auto growth size for the transaction log is 16mb which is arguably too small for this database workload. Unfortunately, it's not possible to change this setting or resize the transaction log in SQL Azure without dropping and recreating the database. Other performance recommendations will help mitigate this current constraint

4 Appendix A: Query Store Analysis

4.1 Mydevice_Marketing

usp_UserTransfer_Filter

```
INSERT INTO @Ids
        SELECT pt.Id
        FROM SEO.UserTransfers pt
        INNER JOIN SEO.Users fp
                ON fp.Id = pt.FromdUserI
        INNER JOIN SEO.Venues fe
                ON fe.[Id] = fp.VenueId
        INNER JOIN SEO.Venues te
                ON te.[Id] = pt.ToVenueId
        LEFT OUTER JOIN SEO.Users tp
                ON tp.Id = pt.ToUserId
        WHERE
                (@FromUserUId IS NULL OR fp.[UId] = @FromUserUId)
                AND (@ToUserUId IS NULL OR tp.[UId] = @ToUserUId)
                AND (@FromVenueUId IS NULL OR fe.[UId] = @FromVenueUId)
                AND (@ToVenueUId IS NULL OR te.[UId] = @ToVenueUId)
                AND (@CreatedAfter IS NULL OR pt.DateCreated > @CreatedAfter)
                AND (@CreatedBefore IS NULL OR pt.DateCreated < @CreatedBefore)
                AND ((@IncludePendingAccept = 1 AND DateAccepted IS NULL AND DateCancelled IS NULL)
                OR (@IncludeAccepted = 1 AND DateAccepted IS NOT NULL AND DateCancelled IS NULL)
                OR (@IncludeCancelled = 1 AND DateCancelled IS NOT NULL AND DateAccepted IS NULL))
```

Index FULL scan (> 2m rows) against Mydevice_Marketing.SEO.Users using non-clustered non-unique index;

```
CREATE NONCLUSTERED INDEX [IX_Users_VenueId_Inc_TaskListId_UId] ON [SEO].[Users]
(
       [VenueId] ASC
)
INCLUDE (      [TaskListId],
       [UId]) WITH (STATISTICS_NORECOMPUTE = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON
[PRIMARY]
GO
```

Consider creating a unique index on ***Mydevice_Marketing.SEO.Users.Uid and include [Id] and [VenueId] to facilitate an index seek operation (traverse the b-tree index and the leaf node) and avoid scanning the entire index. This may be validated using a hypothetical index implementation (create index stats only) and then validating using DBCC AutoPilot prior to implementing the physical index. usp_UserTransfers_FilterMany may also benefit from this optimisation.***

usp_NodeReference_Filter

```
SELECT
              fn.[UId] AS FromUId,
              fnt.Name AS FromType,
              tn.[UId] AS ToUId,
              tnt.Name AS ToType,
              nr.DateCreated,
              nr.CreatedBy,
              nr.DateDeleted,
              nr.DeletedBy
       FROM NodeReference nr
       INNER JOIN Node fn
              ON fn.Id = nr.FromId
       INNER JOIN NodeType fnt
              ON fnt.Id = fn.TypeId
       INNER JOIN Node tn
              ON tn.Id = nr.ToId
       INNER JOIN NodeType tnt
              ON tnt.Id = tn.TypeId
       WHERE
              (@ReferringFromType IS NULL OR fnt.Name = @ReferringFromType)
              AND (@ReferringToType IS NULL OR tnt.Name = @ReferringToType)
              AND (  ((@CreatedAfter IS NULL OR nr.DateCreated > @CreatedAfter)
                     AND (@CreatedBefore IS NULL OR nr.DateCreated < @CreatedBefore))
                            AND ((nr.DateDeleted IS NULL AND @DeletedAfter IS NULL AND @DeletedBefore IS NULL)
                                   OR ((@DeletedAfter IS NULL OR nr.DateDeleted > @DeletedAfter)
                                   AND (@DeletedBefore IS NULL OR nr.DateDeleted < @DeletedBefore))))
```

Index FULL scan (> 1.8m rows) against Mydevice_Marketing.hm.Node using non-clustered non-unique index;

```
CREATE NONCLUSTERED INDEX [IX_TypeIdWithUId] ON [hm].[Node]
(
       [TypeId] ASC
)
INCLUDE (      [UId]) WITH (STATISTICS_NORECOMPUTE = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON
[PRIMARY]
GO
```

Consider creating a composite unique index on ***Mydevice_Marketing.hm.Node [Id, Uid, TypeId] to facilitate an index seek operation (traverse the b-tree index and the leaf node) and avoid scanning the entire index. This may be validated using a hypothetical index implementation (create index stats only) and then validating using DBCC AutoPilot prior to implementing the physical index.***