SQL Server Performance Tuning Imp points: (Author: Steve Scheffler)

Performance Tuning-1: https://gist.github.com/evenkiel/a05d3dbdc29b17ac2284
Performance Tuning-2: https://gist.github.com/evenkiel/e3c048a6046f6be96d88
Performance Tuning-3: https://gist.github.com/evenkiel/52fb3259eaed51caf735
Performance Tuning-4: https://gist.github.com/evenkiel/e218c595f76794dcc022

Performance Tuning-1:

Getting ready steps
- ~~Write down 3 biggest pain points~~
  - "IS" it the database that's slowing things down or environmental? Is perfmon the answer?
  - Deadlocks / locking. Should we be running in a differnet isolation mode?
  - Audit solution - triggers vs in-app, etc.
- ~~Gather SQL server stats~~
- ~~Gather Wait statistics~~

Day One

How To Triage
- Batch Requests per Second (measure of workload)
- "SQL Query Stress" tool
- rollback will give you a % complete status while 'kill' will not
- rollback only ever executes on a single thread, regardless of DOP
- `sp_WhoIsActive` only shows you the last command in the transaction in the session
  - make sure you're using `@get_locks` command to see the locks held by the session
- `sp_AskBrent` will diagnose whether or not there's really a problem with the SQL Server
- `SOS_Scheduler_Yield` points to a CPU issue
- `Page IO Latch` should return in less than 100ms otherwise there may be a storage issue
- `exec sp_configure 'remote admin connections', 1` This will enable Direct Admin Connections
- `sp_BlitzCache`
  - cpu weight is a measure of cpu load
  - focus on `read weight` to see who is doing the most work
  - `exec sp_BlitzCache @top = 10, @sort_order = 'reads'` Will also run sort by 'executions'
- Run `sp_BlitzCache` once per week and review to get a feel for what the top queries should be
- Triage tools in order
  1. Monitoring software (flatline)
  2. `sp_WhoIsActive`
  3. `sp_AskBrent`
  4. `sp_BlitzCache`
  5. `sp_BlitzIndex`
  6. `sp_Blitz`

T-Sql Anti Patterns
- `set statistics io on`
- when using profiler, set 75 millisconds as a threshold to filter out the small statements
- Extended Events - uses event tracing for windows. lighter weight than a server side trace
- Single Statement Table Value Function - SQL Server can optimize and process this much more efficiently because SQL will inline the function
- SQL Server 2014 estimates that 100 rows will come back from a TVF. 2012 and earlier return 1.
- By pushing data into a temp table first, statistics are going to be accurate
- Look for usages of fn_split in our larger procedures. SQL server does a terrible job of estimating when this function is involved
- *DON'T JOIN TO TABLE VALUE FUNCTIONS* (or table variables for that matter). use temp tables instead
- CTEs are executed every time they are referenced.
- Dynamic SQL can result in a large plan cache if you're not using parameters properly
- SARG (search argumentable). non-SARGable means it won't use an index. this can happen with optional sproc parameters

- `WHERE LEFT, UPPER, LIKE, YEAR, VARCHAR = @NVARCHAR` all of these result in a non-sargable queries. all of these will result in full table scans
- Correlated subquery issues:
  - has some of the same problems as scalar functions
  - can use windowing functions ('OVER') instead
- Nested Views - can cause performance issues. Be careful

Understanding Parallelism

- `CXPACKET` means there were inefficiencies in how work was handed out
- `Cost Threshold for Parallelism` - default of 5 is typically too low these days starting point should be 50 or 100.
- Amdahl's law - adding more workers becomes counter productive at some point
- `CXPACKET` should be below 50%, but don't drive to 0
  - usually means there's missing indexes or query opportunities
  - this ensures that only big queries go parallel
  - this is not fixed by setting max DOP, you need to index and tune queries
- Don't look at %s when looking at wait types, look at clock time
- Monitoring software will measure hours of wait time per hour. Very useful

Parameter Sniffing

- "Density Vector" average number of rows for any given value
- If an execution plan gets seeded with a small number of estimated rows, then when the sproc runs again with different parameters it won't necessarily use statistics to find the plan to run
- Look on the actual execution plan XML and you'll see ParameterCompiledValue differences from ParameterRuntimeValue. If it's zero, then it's not using statistics it's just using the cached item.
- With procedures, look for differences in the avg duration between statements and the procedure. This could be an indication that parameter sniffing is an issue.
- stored procedures use statistics differently from direct statements. This can be a real issue when you're trying to trace down perf issues with stored procs inside of mgmt studio.
- `exec sp_helpstats 'dbo.Posts', 'All'``
- `DBCC SHOW_STATISTICS('Posts', 'kl_posts_OwnerUserId')`
  - Steps will show you the number of histogram blocks. 200 is the max, 201 steps with nulls
- If you use a local variable in a sproc then it uses a different set of statistics - uses density vector calculations instead. Talked without someone else at the conference who said they used this across all their sprocs to avoid parameter sniffing issues. This could be used instead of `WITH RECOMPILE`
- You won't be able to reproduce the problem oustide of a sproc
- *INSTEAD*, create a temp sproc to find out what's going on...
- The `WITH RECOMPILE` option, the exec plan will be re-compiled and then not stored in cache
  - Note that this only applies to the outer most statement
  - Rather than 'WITH RECOMPILE' at the sproc level, you should use 'OPTION (RECOMPILE)' at the statement level within the sproc
  - *DON'T put RECOMPILE hints in the header of the sproc*
- `OPTION (OPTIMIZE FOR UNKNOWN)`` as an alternative to recompile. Forces it to caculate estimated rows based on averages
- Local variables inside a sproc CANNOT be sniffed
- `OPTION OPTIMIZE FOR (@UserId=557499)`` this will give you a consistent execution plan
- Another option would be to have different versions of the sproc for small / large customers
- Prepared statements act just like spocs
- NonClustered indexes always include the clustered index key
- When testing slow running sprocs, create a temp sproc, look at the 'compiled for' value from the slow running instance and then `OPTIMIZE FOR` that value.
- Updating statistics forces a recompile. This doesn't mean that the stats were bad, it could be that the cached plan was bad.

Find and Fight Blocking

- Read committed is the current default pessimistic isolation level in the boxed version of sql server
- Use monitoring tools or perfmon alerts to identify blocking issues
- Create alert in SQL Agent - `General Statistics:Processes Blocked`. You can also set the response values to run `sp_WhoIsActive` when this happens. Make sure you set a soak time of like 5 minutes to prevent multiple alerts from coming in
- `LCK_M_IX`, `LCK_M_S`, `LCK_M_U`, `LCK_M_IS` - locks are either row or object locks
- `LCK_M_SCH_S`, `LCK_M_SCH_M` - Index rebuilds will also grab a schema lock when indexing occurs

- `ALTER TABLE DISABLE TRIGGER` - nobody can use the table while you're running
- Blocked process report (look for links in materials) can help you visualize the blocking chains every 5 seconds
- Deadlock monitor runs every 5 seconds. Also reports on blocking that can be picked up by a trace which can be picked up by extended events.
- 'Lock Escalation' SQL Server will escalate to a larger scoped lock once you cross a certain threshold. If you do any type of clustered scan there's a risk that the lock could escalate
- Look at wait stats to see what types of locking are occuring
  - Identify the root of the blocking chain
- Optimistic Concurrency
  - Read Committed Snapshot Isolation (SNAPSHOT)
    - most recent committed version at the statement level
  - Snapshot Isolation
    - transactionally consistent within a transaction
  - Use Snapshot isolation for reports can help without blocking the rest of the application. Make the big read only reports use snapshot isoluation. Best to use only on big read queries, not updates
- You can run RCSI and SNAPSHOT isolation together. SNAPSHOT is great for long running reports, just be sure to wrap the logic for the report in a transaction and set SNAPSHOT isolation
  - With RCSI and SNAPSHOT, performance of tempdb becomes vital.
  - Monitor tempdb perf and look for abandoned transactions (look back through notes on how to do this)

Performance Tuning-2:

Finding Queries and Bottlenecks

- RECOMPILE hints will cause the sproc to not show up in the DMV (Dynamic Management Views)
- DMVs and profiler will only catch perf issues for a window of time. They are passive tools that have to be running when the issue occurs
- `sp_Blitz` will look across a wider window of time.
- ClearTrace as scriptable (and more powerful) version of SQL Server Profiler.
  - Recommended practice is to create a Tools database to store scripts like ClearTrace and the BrentOzar scripts. May shops will not allow anything to be installed to master. Plus, master won't be restored typically during a rebuild
- Extended events
  - Can save to Disk, Activity Counters, or Histograms
  - Very flexible filtering mechanism and supports random sampling
  - Extended events are lighter weight than profiler
  - However, extended events only works (well) on 2012+. Sorry 2008R2
- DMV
  - Collect data since the sql server started up
  - Metrics are very coarse. Anything finer requires active sampling
  - `sys.dm_exec_procedure_stats`
  - `sys.dm_exec_trigger_stats`
  - `sys.dm_exec_query_stats`
  - `CROSS APPLY` is an inner join to a function
- `sp_BlitzCache` - Batteries for DMVs
  - Does not have sampling, so it's looking across all time.
  - For 2008R2 and earlier you'll need to set a flag to pull back trigger stats b/c there's a bug in the way microsoft maintains the counters
- Profiler and Server Side Trace are really the only tools availalbe for 2008R2 and earlier. ClearTrace would be the preferred way to work with this trace info in these environments.
- Activity Monitor can conflict with SQL Server internals. Avoid.
  - Also shows a limited set of information

Solve a Performance Crisis with Plan Guides

- SqlServer:SqlStatistics:Batch Requests/sec
- Processor:% Processor Time
- SqlServer:Sql Statistics:SQL Compilations/sec

- `SET PARAMETERIZATION FORCED WITH NO WAIT` database level switch which will force parameterization across all simple queries even if the queries aren't using parameters. `SIMPLE` would be the normal option.
- Template plan guides can force queries to act as if they have parameters
- Hint plan guides allow you to force OPTIONS or Hints
- Freezing plan guides. Forces a plan in cache to be used. SQL engine will have to do exactly what the frozen version indicates.

Watch Brent Tune Queries

- Read metrics - look at logical reads, CPU time, duration, and query cost
- Identify the biggest problem: reads, CPU, or duration?
- Run sp_Blitz
- End user requirements gathering
  - What's the target
- CREEPY Method
  - Capture query metrics
  - Read the metrics and plan
  - Experiment with query cost:
    - Remove the ORDER BY
    - Change the list of fields in SELECT to just be SELECT 1
      - This makes a big difference in how the query optimizer will look at indexes
    - Switch table variables to temp tables
    - Question any non-INNER joins
  - Execution plan review
    - Look at big differences between estimated and actual row counts
    - Tune off the actual execution plan rather than the estimated
      - Are stats out of date
      - Parameter sniffing issues?
    - Identify implicit conversions or SARGability issues
    - Split query into temp tables and replace joins
  - Parallelism opportunities
  - Index improvements
- Always question `ORDER BY` clauses
- Focus on logical reads first and get that number down. Stats will give you this information. Plug into statisticsparser.com to help with output.
- Statistics get updated by SQL Server when the amount changes by ~20%
- GROUP BY can perform better than COUNT(DISTINCT(x))
- 4 metrics to measure performance improvements (show % change of each)
  - Logical reads
  - Duration
  - CPU time
  - CXPACKET
- `SET STATISTICS TIME, IO ON` this will dump run stats to the messages window. You can then pipe this to the statisticsparser.com site for easier to read details
- SQL Server always thinks that 1 record will come back from a table variable, UNLESS you use `OPTION (RECOMPILE)`. However, that spikes CPU. This is why temp tables are preferred over table variables.
  - You'll get better statistics if you use temp tables
  - CTEs can give good statistics, however they will get rerun every time they're referenced

High Performance Reporting Services Tuning

Performance Tuning-3:

Performance Tuning Indexes

- `sp_BlitzIndex`
- If you don't create a clustered index then data is stored in a structure called a 'heap'
- Usage stats shows you what was in the plan, Op Stats will tell you what was acutally used.
- A 'scan' isn't necessarily a full scan. You need to look at the actual execution plan to see
- Don't rebuild indexes with a page count < 2,000 since it won't make a difference

- Index maintenance won't apply to heaps unless you actively force it
- Goal is to create the least indexes to solve the most problems
- The key of an index should be as specific as possible. If you have a very broad index field then that will require more scans
- Look at Equality vs Inequality lists.
- Indexes are ordered by the key columns, include column order doesn't matter at all
- `(ONLINE = ON)` is only available in Enterprise Edition
- Plan cache review is the best way to measure improvements by looking at CPU and logical read differences
- Look at the plans of all top queries to see where more indexes can help
- Trivial optimization never does an index request - there's only one way to execute
- Look for the key lookup pattern and see if widening an existing index via includes could help
- If you don't make a key unique, then SQL Server will make it unique for you with the UNIQUIFIER
- `sp_BlitzCache` look for high average reads in the plan cache
- If you see a column only in the Output columns and not listed as a predicate then it's a good candidate for adding as an INCLUDE to an index rather than being added as a key.
- Good clustering key properties
  - Unique (doesn't have to be absolutely unique if you have a good natural key and can't change)
  - Narrow
  - Static (don't want rows to change this value and then get moved around a bunch)
  - Supports queries
  - Auto incrementing
- Look for high average reads and average CPU. Also look for scans with predicates
- If you have a scan that is going seek backwards then this will inhibit parallelism
- A constraint (as in uniqueness) will always create an index behind the scenes

Signs You Need to Rethink Schema (partitioning)

Bulk Loads

- Filtered indexes to separate out workloads
- Frequent maintenance
- Loading / deleting data in chunks
- Switching recovery modes (bad idea)
- Filtered index is just an index with a WHERE clause
- Filtered indexes won't be used if you use a parameter!
- Partitioning is generally only done in data warehouses. Doesn't help in OLTP systems.
- Load & Archival
  - Load data into an unindexed heap
  - Clean it up, add the right indexes
  - Swap the scratch space into the active table
  - SQL Server 2014 introduced WAIT_AT_LOW_PRIORITY to avoid blocking chains
    - Regular queries can continuing working while you're waiting to swap out
  - Partitioning is a data management function and not a performance function. Helps with concurrency
- You can perform maintenance on individual partitions
- 2014 introduced incremental statistics tracking - can then just update at the partition level

Concurrent Data Access

- SQL Server 2008+ can use multiple threads per partition
- SQL server will push the partition ID as a secret key at the front of your indexes. Microsoft calls this a 'skip scan'

Partitioning Benefits

- Swapping
- Partition level maintenance
- Partition level lock escalation
- Partition aware seeks

Column Store

- This was designed to be used in conjunction with partitioning
- NOT for OLTP. This is tailored for Data Warehouse installations only

Indexed Views

- Lots of limitations. Can be useful if you have lots more reads than writes
- With standard edition, you must force the hint as the optimizer will not use it
  - `OPTION(NOEXPAND)`, `OPTION(EXPANDVIEWS)`

Filtered Indexes
- If you have known criteria then you can see some dramatic improvements with query performance
- You're moving the work to the write rather than the read.
- Doesn't allow for parameters so you'll have to use dynamic sql to pass literals in the sql string
- Helps with low cardinality indexes
- DO create a filtered index per queried value (e.g. status codes). DON'T create a filtered index per value.
- Optimizer considers more cases to apply filtered indexes than indexed views
-

Diagnosing Problems with In-Memory Tables (Hekaton)
- Lots of limitations. Very very early access at this point.
- "RPO" - Recovery Point Objective
- "RTO" - Recovery Time Objective

How to Write a Prescription
- Start with symptoms and perceived root cause
- Choose a metric
- Hours of wait time per hour is a good general 'how busy am I' metric for a server. Most monitoring packages will track this out of the box.
- Blocked queries per hour via Perfmon. Extended events would be a more detailed mechanism for identifying blocking chains.
- LCK* wait time per hour
- Percentage waits and measures aren't really helpful as they can be misleading

Performance Tuning-4:

How to Measure Your Server
- 3 Key Numbers
  - How busy is your server
  - How hard is it working
  - Ho wmuch data do you have

How Busy is Your Server
- Perfmon `SQLServer:SQL Statistics - Batch Requests/sec`. Trend this on an hourly basis and break it out by Weekday/Weekend
  - Do NOT look at transactions per/sec, since not all statements are transactions
  - 0 - 1,000 easy to handle with commodity hardware. MAY be possible to run this load on a VM
  - 1,0000 - 5,000 be careful, because a table lock or bad query can knock the server over
  - 5,0000 - 25,0000 You should have several full time DBAs performance tuning. Every index matters crticially
  - 25,000 + requires lots of attention, but you need to have always on availability groups.
  - Record is 150,000

How hard is it working
- Hours of wait time per hour
  - brentozar.com/go/getwaits - trend this on an hourly baiss
  - The more the server works the more it waits
  - 0 - Server isn't doing anything
  - 1 hour of waits - still not doing much
  - 1 hour of waits X # of cores - working hard, look at tuning

How much data do you have
- All numbers below assumce 'commodity' hardware which is 2CPU and 250GB + of RAM
- 1 - 150GB Std Edition
- 150 - 500 GB Enterprise
- 500GB + OLTP vs. Analytical?
- 1TB OLTP data - very challenging

What If It's Hardware

- http://cpuid.com can be used to look at exactly what the CPU is doing
- Get a faster CPU. Pay attention to the number of cores
- 7k per core with Enterpise. 2k per core with standard
- PAGEIOLATCH - make storage faster. Add more memory
- Perfmon `Phsical Disk: Avg Sec/Read`. 20 - 100 ms is good. > 100ms means you have a storage issue
- Don't ever change storage in an existing server. Buy new hardware and test/tune there
- WRITELOG
  - Commit is only completed when the transaction log is written to.
  - Perfmon `Physical Disk: Avg Sec/Write`. 3-20ms is good
  - Perfmon `Physical Disk: Reads/Sec, Writes/Sec`

Reporting in Production

- You need more RAM than data. If you have 128GB of data then you should have roughly 192GB of RAM
- If you can fit data in memory, then most of your tuning problems go away
- By default, SQL Server will allow one query to allocate up to 25% of available memory. 4 users with aweful reports could swamp a server
- Can use filtered indexes to get best performance for recent history
- Indexes views are a view with a where clause - has a clustered index
- `DBCC TRACEON(610)` minimal logging - useful for DW transfers to significantly drop logging
  - Almost impossible to run if you are running FULL recovery. Need to be running in SIMPLE recovery mode
- Use SIMPLE recovery model for datawarehouse databases. Pair this with `DBCC TRACEON(610)` to get super minimal logging and better speed for writes
- `WITH INDEX(1)` will force the clustered index to be used

How to Tell When tempdb is a Problem

- The 'Version Store' uses tempdb
- Temp tables and temp variables live here as well
- SQL server can round robin and distribute amongst the available temp db files
- For best results, have equally sized files
- For lots of small allocations, like you have with tempdb, having multiple files helps because you get multiple PFS (page free space) and SGAM (shared global allocation map)
- PAGELATCH_UP - tempdb waits. means the page is in memory
- PAGE_IO* - means that page is pulling from disk
- You can see this with the `sp_WhoIsActive` script
- Look at wait stats to see if adding tempdb files can help distribute workload
- SQL 2014 has greatly reduced IO in tempdb
- Latency thresholds for concern
  - Read latency - 30ms for data files, 5ms for log files
  - Write latency - 30ms for data files, 2ms for log files
- Microsoft's guidelines - start with the # of tempdb files as you have physical cores, up to 8. From there you need to measure waits and test
- They like to pre-grow out tempdb and fill up the drive
- Simple approach - start with 4 equally sized tempdb data files and then watch
- Don't need to necessarily put tempdb on a separate physical drive unless you're seeing phpyiscal IO waits. Kendra recommended creating a separate logical volume to contain tempdb so that you can control it's growth. Also allows you to move it around easily
- PREEMPTIVE_OS_* means that it's a wait on something outside of SQL Server. Could also be related to encryption.

Scale Out for Success

- Content Addressable Storage as a replacement for storing images/blobs in a SQL Server. http://www.emc.com/data-protection/centera.htm

- ElasticSearch - full text search solution in use by Stack Overflow
- Redis for an open source caching layer

Identifying Configuration Problems That Burn Performance

- ostress.exe - simple tool for scripting queries to hit a dev SQL Server. Similar to hammerdb
- Set autoshrink to off in sys.databases
- Stats updates happen synchronously when the next read comes through after writes cross the registered threshold
- THREADPOOL waits are an example of a poison wait. Shouldn't ever really have these.
- SQL Server memory ools:
  - Execution Plan Cache
  - Buffer Pool Cache (aka Data Cache)
  - Query Workspace Memory (aka Memory Grant)
- RESOURCE_SEMAPHORE - Queries are asking for workspace memory and SQL server is unable to grant
  - Scanning large tables can lead to big memory grants
  - For 2012/2014 you need to set memory higher than max memory of 128 on Standard.
  - Make sure that SQL Server is under external memory pressure
- sp_Blitz checks for the poison wait types
- The memory limits for SQL Server apply to the Data Cache only. So, add 15% of so to the max edition limit and set the max sql server memory to this level.
- Leave 10% of 4GB (whichever is higher) for the OS