1. Use instant file initialization for data files (does not have to zero out the file). Helps with reducing time taken to auto-grow. Reduces time to restore from backups by a lot. Zeroing-out of file cannot be skipped for log files.
2. Use multiple files of equal size per file group (between 4 to 8 files per file group. 4 is a decent default) and equal auto-growth.
3. Although writes to log are sequential (written to serially. If concurrent transactions are running at the same time, they are interleaved), but if you place multiple logs files (from different databases) on the same volume (shouldn’t it be filegroup?), then the performance might come down (because trying to write to multiple of these log files at the same time would result in random IO). But it should not matter for SSD drives as they are efficient for random accesses?? Even though SSDs solve the random-access performance, we will have contention issues (IO throughput) on that drive if log file from multiple databases are stored on the same drive and being written to concurrently.
4. Data compression maybe is not suitable for OLTP workloads. CPU vs IO trade-off.
5. De-fragmentation can also reduce space used. Worthwhile to enable backup compression.
6. Use sp\_estimate\_data\_compression\_savings to estimate compression savings before turning on data compression.
7. Row compression makes everything variable length where possible (e.g. char to varchar). Page compression, loosely explained, extracts common substrings and stores them once and references them where needed.
8. Number of tempdb data files should be = no. of cores of cores<8 and =8 if no. of cores >8 (and keep increasing by 4 if still seeing contention). But PFS latch contention is fixed in SQL 2019. So only 1 data file for tempdb should work?? But SQL Server still created 8 data files by default on my 8 core machine. Should the extra tempdb data files be created on separate filegroups?
9. Versioning isolations (RCSI and Snapshot Isolation) will remove blocking for readers (not for writers which will still use locking). We can also have replication to have subscribers handle read queries. In that case, the queries served by subscribers can block the replication and vice-versa. Having versioning isolation on subscribers can also reduce blocking.
10. Use RCSI as default for new on-prem dbs (it is the default in Azure). Just make sure tempdb is configured right (on SSD and with right number of files and size of files). The issue with this could be that it is mostly write workload, then even though those writes would be versioned, no one would be reading the versioned data. Hence, we would do extra work for versioning (tempdb overhead) that no one uses. Better to start with RCSI only instead of RCSI in conjunctions with SnapshotIsolation.
11. A long running transaction would require an equal amount of free space in log file to be able to undo it. An insert into a table which inserts millions of rows is a long running transaction. So, is it better run batch the large insert into chunks and then every so often between these batch inserts, we take a backup to truncate the log or do we do checkpoint??? What type of backups we take to mitigate this? Log backups or database backups?? Checkpoint does clear out the log file by flushing the changes to data files. If the recovery model is simple, checkpoint will clear/truncate the log. If it is bulk-logged/full, then transaction log backup will clear/truncate the log. Log clear/truncate is misnomer as it neither clears the log nor truncates the log (shrink is used for truncation). It only marks log space not being used by open transactions to be available for future use. Log size would remain same. So make sure to either checkpoint or take log backups regularly to prevent auto-growing log scenario.
12. Lazywriter vs checkpoint. Checkpoint flushes changed data pages to data files on disk (does it also free up log file space). But lazywriter could also write the dirty page to disk when there is memory pressure.