AlwaysOn Availability Groups - What Not to do when Adding Databases

10 March 2014

*by* [*Warwick Rudd*](https://www.simple-talk.com/author/warwick-rudd/)

*FROM:* [*https://www.simple-talk.com/sql/database-administration/alwayson-availability-groups---what-not-to-do-when-adding-databases/*](https://www.simple-talk.com/sql/database-administration/alwayson-availability-groups---what-not-to-do-when-adding-databases/)

SQL Server's AlwaysOn Availability Groups provide a very resilient way of providing High-availability for SQL Server databases, but there are inevitable limits to their capacity. How many databases can you reasonably add? It depends on the resources available and the workload, but you can come up with a reasonable estimate  as Warwick Rudd explains

AlwaysOn Availability Groups (AGs) were first introduced as a new High Availability (HA) / Disaster Recovery (DR) feature with the release of SQL Server 2012. In this article I am going to assume you already have set up and configured AGs in your environment. If you need to know how to setup and configure your AGs have a look at my previous article “[SQL Server 2012 AlwaysOn](https://www.simple-talk.com/sql/database-administration/sql-server-2012-alwayson/)”.

Like any new feature or technology without understanding how it works can make it easy to make mistakes and cause problems in your environment. This feature is definitely one that should not fall into the PILOT (Production In Lieu Of Testing) category. Careful planning and testing should be undertaken to ensure you have covered all of your bases.

Recently I was asked what issues might emerge from the process of adding a large number of databases to an AG. In this case, a two-node Windows Server Failover Cluster (WSFC), configured on virtual machines, was being used with four vCPUs on each of the cluster nodes. A script was put together to add over 300 databases to a single AG on this virtual machine setup. During the running of the script, some issues were experienced which left those databases that were added in a variety of states of play. The dashboard was showing some of the databases to be offline, others online, while some on the replica were not connected to the AG. This was a case of trying to over-utilise the resources that were available, and the WSFC not coping. The upshot was that all the databases could be recovered, because this was the initial addition of the replica databases, and all had full backups and associated transaction log backups. However, the design needed to be re-evaluated to meet the HA/DR requirements. This work led me to put this article together to mention some of the things you should not do with AGs.

When you implement AGs, you must remember that there is a limit to the number of AGs and availability database per server, either physical or virtual. The limit to the number of databases isn’t a hard-and-fast number because it depends on the resources available and the workload that will be undertaken. With the release of SQL Server 2012, Microsoft extensively tested a hundred databases spread evenly across ten AGs (ten databases per AG on average) per physical machine. The first signs of overload in your server environment from having too many databases configured will be Worker Thread exhaustion and a slow response times for system views or DMVs. -see MSDN’s ‘Prerequisites, Restrictions, and Recommendations for AlwaysOn Availability Groups (SQL Server)’.

**Worker Thread Exhaustion**

It is not just with AGs that you’ll see error messages related to Worker Thread Exhaustion, but when you do, it is likely to be a symptom of adding too many databases to your AG environment. Your system will become unresponsive and, if you try to open a new connection to your instance of SQL Server through SSMS, you will more than likely be met with an error message such as “Timeout expired. The timeout period elapsed prior to completion of the operation or the server is not responding”. Alternatively you could investigate the SQL Server error log and you may see a message like “New queries assigned to process on Node x have not been picked up by a worker thread in the last x seconds.”

If you do experience such an error, you may then have some difficulty in connecting to the affected instance. If you do, you could restart your instance of SQL Server: This will clear all connections and so should allow you to connect. If not, you can still gain access if you are administrator by using the Dedicated Admin Connection (DAC). Once you have a connection you can go looking for the ID of the troublesome session. More than likely, this will be your process that is trying to add the three hundred Databases.

**Worker Threads**

A worker thread is what is used by SQL Server to perform some form of processing. A number of factors will determine the number of worker threads available on your system. They are created automatically by SQL Server as required. SQL Server has an instance-level configuration that is used to limit the number of worker threads available to SQL Server processes.

**Max Worker Threads**

The ‘Max Worker Threads’ configuration option is at the Instance-level and is found in sys.configurations. It can be set or changed using the sp\_configure system stored procedure. This configuration option enables SQL Server to create a pool of worker threads that are then available to service a large number of SQL Server processes: This may improve performance. By default this value is set to 0. A value of 0 allows SQL Server to automatically configure the number of worker threads at start up time.

The following table shows the number of worker threads that are automatically-configured based on the number of CPUs present on the server:

|  |  |  |
| --- | --- | --- |
| **Number of CPUs** | **32-bit** | **64-Bit** |
| <= 4 Processors | 256 | 512 |
| 8 Processors | 288 | 576 |
| 16 Processors | 352 | 704 |
| 32 Processors | 480 | 960 |
| 64 Processors | 736 | 1472 |
| 128 Processors | 4224 | 4480 |
| 256 Processors | 8320 | 8576 |

As this article is looking at the use of AGs we will only be concerned with the 64-bit worker thread numbers.

**AG Worker Thread Requirements**

There are a number of factors to consider when undertaking the capacity-planning phase for your AG environment. One such factor is the amount of resources each replica will require to provide appropriate performance for the system. To work this out, you need to take into account not only the normal activity that your environment will undertake but also the number of databases that will participate in the data transfers from your primary replica to your identified secondary replicas. There are, of course, many other factors as well that need to be taken into consideration, and some of them are covered by Jeremiah Peschka’s article “AlwaysOn Availability Groups: The Average of its Parts” but for this article we are only concerned with working out the scale of the resources in terms of the Worker Threads.

AGs have the following worker thread requirements (As quoted from “Prerequisites, Restrictions, and Recommendations for AlwaysOn Availability Groups (SQL Server) – Thread Usage by Availability Groups”):

* An idle instance of SQL Server, AlwaysOn Availability Groups uses 0 threads
* The maximum number of threads used by Availability Groups is the configured setting for the maximum number of threads (‘**max worker threads**’) minus 40
* The availability replicas hosted on a given server instance share a single thread pool (HADR Worker Pool)
* Threads are shared on an on-demand basis :
  + Typically there are 3-10 shared threads, but this can increase depending on the primary replica workload
  + If a given thread is idle for a while it is released back into the general SQL Server thread pool. Normally, an inactive thread is released after ~ 15 seconds of inactivity. However, depending on the last activity, an idle thread might be retained longer
* In addition, availability groups use unshared threads as follows:
  + Each primary replica uses 1 Log capture thread for each primary database. In addition, it uses 1 Log send thread for each secondary database. Log send threads are released after ~ 15 seconds of inactivity.
  + Each Secondary replica uses 1 redo thread for each secondary database. Redo threads are released after ~ 15 seconds of inactivity
  + A backup on a secondary replica holds a thread on the primary replica for the duration of the backup operation.

Even though AGs are an extension of the database mirroring technology, the worker thread usage for AGs is different than with database mirroring. Database mirroring used dedicated threads per database compared to the request queue and worker pool (HADR Worker Pool) to handle the requests. Bob Dorr gives a really in-depth account of how the worker pool is utilised by AGs. You can read more in his article – [HADRON Learning Series: Worker Pool Usage for HADRON enabled Databases](http://blogs.msdn.com/b/psssql/archive/2012/05/17/alwayson-hadron-learning-series-worker-pool-usage-for-hadron-enabled-databases.aspx).

There are other worker thread requirements beyond these. The minimum number of work threads required just to facilitate having AGs configured can be calculated from

* the number of AGs you have configured in your instance of SQL Server
* the number of availability databases in each of the AGs
* the number of availability replicas (2-5 replicas. A maximum of 4 secondary replicas with SQL Server 2012)

To calculate the minimum worker thread pool size required, for this article our environment will be configured as follows:

* One Primary Replica
* One Secondary Replica
* One Availability Group
* One Hundred Availability Databases

The information that we will use in the minimum pool size calculation is:

* Database Count (DC)
* Secondary Replica Count (SRC)
* Log Capture Thread (LCT)
* Log Send Thread (LST)
* Message Handler Thread (MHT)

For each database participating in an AG, one LCT is used to capture the transactions occurring on the database. One LST is required for each secondary replica in the AG. To allow the process to work at least one MHT is required to handle the communication occurring between replicas.

The algorithm we are going to use as outlined by Bob Dorr’s Article - HADRON Learning Series: Worker Pool Usage for HADRON enabled Databases is:

* Minimum Pool size = (DC x (LCT + (LST x SRC))) + MHT

In a worst case scenario for our environment mentioned above all one hundred replica databases are actively being used.

* Minimum Pool Size = (100 x (1 + (1 x 1))) + 1
* Minimum Pool Size = (100 x (1 + (1))) + 1
* Minimum Pool Size = (100 x 2) + 1
* Minimum Pool Size = 201

The Minimum Pool size required for our environment in this article would be 201. If we then looked at the worker thread table above, we’d see that the number of threads available to the system on a 64 bit 4 cpu server would be 512. From the AG requirements outlined earlier in this article “The maximum number of threads used by Availability Groups is the configured setting for the maximum number of threads (‘**max worker threads**’) minus 40” we can calculate the maximum available for your AGs would be 472 (512 – 40). Our calculation shows we will need 201 threads to service AG workload however this is not taking into account the number of threads that would be required for the transactional activity that will be occurring on the system or the number required for performing backups on the system. So we can see that the number of threads available are being used up very quickly not leaving any available resources for workload utilisation spikes.

If you try to add too many availability databases to an AG via a script, you will run out of worker threads very quickly. Your script will lose its connection and will not complete the process. Your AG environment will have databases sitting in a state where they are not part of the AG. In order to get back online with at least your primary replica databases, you will need to drop your AG(s) and restore, where necessary any irrecoverable databases on the primary replica with the backups taken as part of the initial AG setup process.

Because of the risk of things going wrong, any deployment into an AG needs to be carefully planned, with a prior calculation of the resources required, especially when these AGs are part of your production environment. Make sure you understand your existing capacity requirements and forecast the potential growth in activity or utilisation so you are able to identify appropriate server resources to meet your needs.

**References**

* AG Prerequisites, Restrictions & Recommendations - <http://technet.microsoft.com/en-us/library/ff878487.aspx>
* Bob Dorrs Worker Threads - <http://blogs.msdn.com/b/psssql/archive/2012/05/17/alwayson-hadron-learning-series-worker-pool-usage-for-hadron-enabled-databases.aspx>
* Jeremiah Peschka’s Article - <http://www.brentozar.com/archive/2013/08/the-average-of-its-parts/>

**Other useful resources**

* Worker thread investigation - <http://blogs.msdn.com/b/sql_pfe_blog/archive/2013/07/01/are-my-actual-worker-threads-exceeding-the-sp-configure-max-worker-threads-value.aspx>
* AG Worker thread consumption - <http://blogs.msdn.com/b/sql_pfe_blog/archive/2013/07/15/monitoring-sql-server-2012-alwayson-availability-groups-worker-thread-consumption.aspx>
* Configuring the Max Worker Thread - <http://msdn.microsoft.com/en-us/library/ms190219.aspx>
* Max Work Thread Options - <http://technet.microsoft.com/en-us/library/ms187024(v=sql.105).aspx>
* Troubleshooting Threadpool Waits - <http://www.sqlpassion.at/archive/2011/10/25/troubleshooting-threadpool-waits/>
* Threadpool and SQL Server Threads - <http://www.sqlpanda.com/2013/07/threadpool-and-sql-server-threads.html>
* Failover and Failover Modes - <http://msdn.microsoft.com/en-us/library/hh213151(v=sql.110).aspx>