IDERA® | SQL Server Whitepaper

TIPS FOR MANAGING LARGE NUMBERS OF SQL SERVER JOBS

BY ROBERT L. DAVIS

Techniques for managing large numbers of SQL Server jobs across enterprise environments

INTRODUCTION

Managing a large number of SQL Server jobs across a large enterprise environment can be difficult and frustrating. Database Administrators are often asked to schedule jobs for a variety of SQL processes and SSIS packages with little or no insight into what the processes or packages will be doing. Scheduling packages blindly can often yield to performance problems from jobs that try to use the same resources or access the same data structures at the same time. Job overlap can be difficult to find and can cause a lot of problems.

This paper will share my three top tips for managing a large number of SQL jobs and SSIS packages. These tips will make administering the jobs easier and prevent common problems that are inherent in a large SQL job environment.

1. Centralize the SQL job servers on to dedicated servers
2. Utilizing the new SSIS catalog in SQL 2012 and newer
3. Use a third party tool that can give you the big picture and let you look at your SQL job environment as a whole

These tips will make administering a large number of jobs easier, alleviate some common resource issues with executing many SQL jobs, simply troubleshooting SSIS package failures, and bring the problems in your environment into a single view.

CENTRALIZE JOB SERVERS

A critical component of my enterprise SQL job environment is centralized job servers. Centralized job servers make management easier by limiting the number of places you have to look to find failed jobs or see job information. It makes things like configuring availability or disaster recovery easier because you don't have to worry about enabling or disabling jobs on local servers when the database or server fails over. And lastly, it is easier to see the big picture of the jobs that are running if you only need to look a few places instead of hundreds.

In addition to easier management, dedicated job servers take a lot of load off of the production servers. SSIS packages and SQL jobs use non-buffer pool memory and they can easily suffer performance problems if there is external memory pressure on the server. It's a very common issue to see jobs and packages run for a long time when external memory is low, often caused by the fact that most of the memory on the server is allocated to SQL Server which is wholly consumed by the buffer pool (note: buffer pool is interchangeably used to mean the entire allocation of internal memory that is split up amongst the different caches and specifically the data cache within the pool).

While slow performance of SQL jobs and SSIS packages is a symptom of external memory pressure, the jobs and packages may be the cause of the external memory pressure. When you run the jobs on a dedicated server, they only compete with one another and not with the database engine itself.

USE POWERSHELL

In the past, one of the obstacles to using a centralized job server was that executing SQL processes like stored procedures was difficult. Since SQL Server 2008, PowerShell has been integrated into SQL jobs. There is almost no limit to what you can perform with PowerShell, and it works very well for executing procedures locally on a remote server. There is where the invoke-sqlcmd cmdlet (pronounced commandlet) saves the day. Invokesqlcmd is a cmdlet that acts very similar to the sqlcmd utility for command-line execution of scripts or queries.

In its simplest form, I can use invoke-sqlcmd to execute a procedure on a remote server. Below is an example invoke-sqlcmd call for a job.

```
Invoke-SqlCmd -ServerInstance MyServer -Database MyDB -Query
    'EXEC dbo.MyProcedure @ Parm1 = 42;' -QueryTimeout 3600 -EA Stop
```

The key arguments for this simple call are:

ServerInstance The SQL instance to which we will connect.

Database The database in which the action will be performed. If not provided, will use the default database of the login executing the process.

Query The T-SQL command to execute. Alternatively, you could use the –InputFile argument to execute a script.

QueryTimeout The timeout value for the procedure. It is crucial to use this parameter because under certain circumstances, the cmdshell call made by the SQL job could fail and the SQL job not get notified of the failure and it would wait infinitely for some sort of notification. If this happens, the query timeout will fire and fail the execution of the job step. The default timeout is none or wait infinitely.

EA This is a "common parameter" that is not specific to the invoke-sqlcmd cmdlet. It is part of the common PowerShell interface. In this case, we want an error to stop the job and fail it. The default action is to continue. The available actions are stop, continue, silentlycontinue, and inquire (ask if you want to continue).

For the full syntax of the invoke-sqlcmd, run or import the sqlps module and use the get-help command.

You can do a lot more with PowerShell inside of a SQL job step than just execute a procedure or SQL query. Other common processes I use PowerShell for is to execute multi-step processes that may include dynamically generating a query to execute and then exporting the results to a file.

For example, the following PowerShell job step command generates the query to get a report ID and passes the ID to a custom utility that exports the report to an Excel spreadsheet and copies it to a designated share.

```
Query = @"
DECLARE @rptName varchar(50), @rptID int, @deliveryFrequency varchar(50);
SET @rptName='DailyExport';
SET @deliveryFrequency='Daily';
SELECT rptID
FROM dbo.reportDoc
WHERE deliveryFrequency = @deliveryFrequency
AND rptName = @rptName;
"@ $ExtractDirectory=" D:\SSISpackages\Extracts"
$ResultSet = invoke-sqlcmd -ServerInstance MyServer -Database MyDB -Query
    $Query -QueryTimeout 3600 -EA Stop [int]$RptID = $ResultSet.rptId
    Write-Output "Calling ExtractsDataExport.exe with ReportID: $RptID" &
    "$ExtractDirectory\ExtractsDataExport.exe" $RptId
```

The entire process could have been put into PowerShell script stored on the job server file system that gets executed in a CmdExec (operating system) job step that executes via a command like the below.

```
C:\Windows\System32\WindowsPowerShell\v1.0\Powershell.
exe -File “D:\PowerShellScripts\ ExtractsDataExport.
ExtractsDataExport.ps1 ‘DailyExport’”
```

One of the challenges with putting the jobs all on a centralized server is that people tend to want to go the easy route and use highly privileged accounts for processes that don’t need elevated rights. There is a very simple solution to this problem, proxy accounts.

PROXY ACCOUNTS

Whether you need to execute an SSIS package or a PowerShell command or script or some other type of process, you can still have granular control over the account the process uses. SQL jobs allow you to define proxy accounts that get used as the execution account for the process.

In order to use a proxy account, you must define a Credential and assign it to the appropriate SQL subsystem. Follow the below steps to create and use a proxy account via T-SQL.

1. Create a Credential

```
USE master;
     CREATE CREDENTIAL [SQLMCM\SQLSoldier]
     WITH IDENTITY = N'SQLMCM\SQLSoldier',
     SECRET = N'<Enter Password>';
```

2. Create a proxy

```
EXEC msdb.dbo.sp_add_proxy
     @proxy_name = N'SQLProxy_SQLSoldier',
     @credential_name = N'SQLMCM\SQLSoldier',
     @enabled = 1;
```

3. Assign the proxy to one or more subsytems (subsystem 11 = SSIS)

```
EXEC msdb.dbo.sp_grant_proxy_to_subsystem
     @proxy_name = N'SQLProxy_SQLSoldier',
     @subsystem_id = 11;
```

Now that the proxy is created, you can assign it as the run-as account for the job step for the assigned subsystems. To get a list of subsystem IDs for each subsystem, query dbo.syssubsystems in the msdb database. When granting permissions for the proxy account, permissions should be granted to the service account specified in the credential. Using the example above, I would grant permissions to SQLMCM\SQLSoldier.

USE THE SSIS CATALOG

One of the really cool management features introduced with SQL Server 2012 is the SSIS catalog, also known as SSISDB. The SSIS catalog offers some really cool development features as well, but for this paper, I want to focus on the manageability features. One of my personal pet peeves about managing SSIS packages that I know very little about is that when the package fails, it is very difficult for me to troubleshoot the failure unless the package has instrumented custom logging.

If you have an environment where the packages always log very good details of the execution, you are managing an anomaly. A lack of logging and error reporting is the norm for most environments. Fortunately, the SSIS catalog now logs execution of packages stored in it with great detail. Prior to SQL Server 2012, if I wanted to troubleshoot an SSIS package, it usually meant I had to open the package in Visual Studio and step through it until I found the failure. When you look at a failed job for an SSIS package stored in SSISDB, you will see a generic error message like the following:

```
Package execution on IS Server failed. Execution ID: 18717, Execution
Status:4. To view the details for the execution, right-click on the
Integration Services Catalog, and open the [All Executions] report.
```

To see the All Executions report mentioned in the error message, check the SSIS catalog. Follow the below steps to find the real error message:

1. Expand the Integration Services Catalog
2. Right-click on the SSISDB catalog
3. In the popup dialog, highlight Reports
4. Highlight Standard Reports
5. Click on All Executions
6. When the report loads, it will show a listing of all recent job executions
7. Scroll through the list until you found the failure
   (List auto-loads the next page of results if you scroll to the end of the page)
8. Click on Overview and got a breakdown of the execution of the package as a whole and each individual step inside of the package

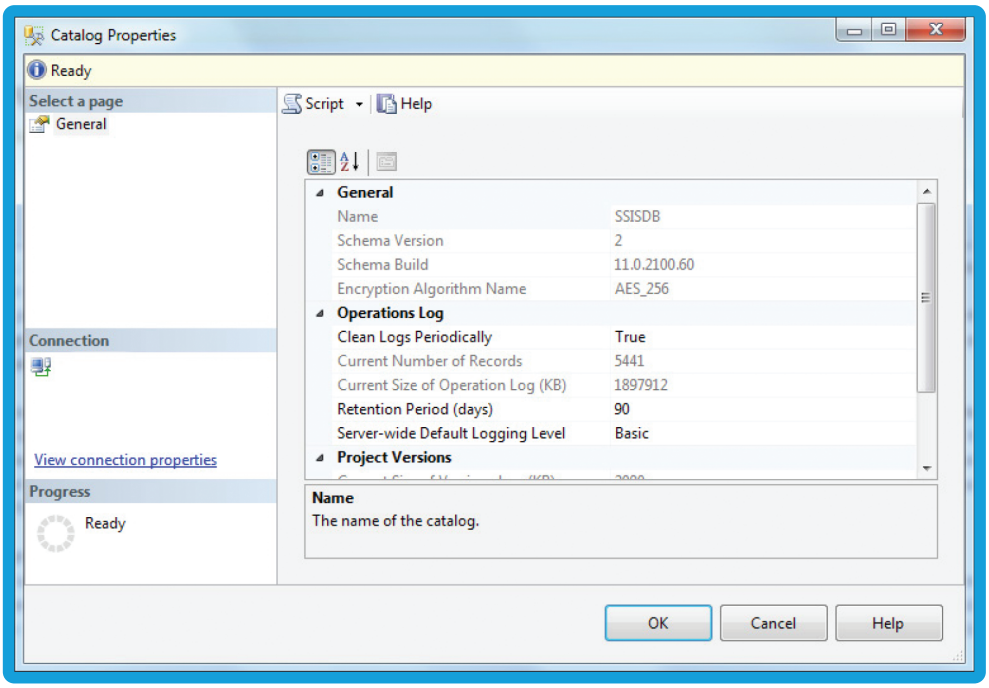
When you find the step or steps that the errors occurred on, you will see exactly what was being performed and what errors were generated. The steps inside the log are not broken down by step of the SQL job nor the SSIS package. The steps are the internal steps of the package. A single step in the package may be broken up into many sub-steps such as validate the package, re-execution, execution, etc. You should see an error similar to the following:

```
Error: SSIS Error Code DTS_E_OLEDBERROR. An OLE DB error has occurred.
Error code: 0×80004005.

An OLE DB record is available. Source: “Microsoft SQL Server Native
Client 11.0x Hresult: 0×80004005 Description: “Transaction (Process ID
87) was deadlocked on lock resources with another process and has been
chosen as the deadlock victim. Rerun the transaction.”.
```

You can right-click on the SSIS catalog and click Properties to see information about the database and get access to the settings for the internal package logging. You can see how many records are in the database and how much space it is using. You can edit the data retention, enable or disable automatic clean-up for the logs, and change the level of logging. Although higher levels of logging are available, you can usually get everything you need from the default, basic logging level.

See the image below for an example of the SSISDB properties.



SEE THE BIG PICTURE

If you have a large environment, you probably have a lot of SQL jobs as well. Managing the schedules for many SQL jobs across many servers can be difficult. I highly recommend a utility for seeing the big overall picture for the job schedules. Jobs overlapping can be the cause of resource bottlenecks, deadlocks, or extensive blocking. Changing the processes or packages that the jobs run is more than likely outside your realm of responsibility.

If jobs are colliding and causing problems, a developmental solution can take a while to get pushed through. Sometimes the problem can be alleviated by adjusting the schedules so that the jobs are not colliding, either as a long-term fix or a short-term solution until a permanent fix can be developed. But you should not blindly change job schedules and hope for the best. You need to see what other jobs are running at which times to ensure that you're not just going to collide with a different jobs.

With a little work, you can build your own report to show the job execution information. There are also some third party tools you can use to view he job schedules of the servers. Some tools are free, like SQL Job Manager from Idera. SQL Job Manager gives you a customizable calendar-view display of the SQL jobs across multiple servers to help you quickly find jobs that overlap, jobs that have failed, and see what is currently running.

SQL Job Manager also gives you a simple drag-and-drop interface to move jobs between servers to equalize the load or to remove contention on local resources of the job server. In order to take advantage of this feature, you really need to have jobs that are not tied to a particular server. In other words, dedicated job servers as described earlier allow us to take full advantage of the features of this tool.

There are also some paid third party tools you can get for more features to help you manage your SQL job environment. SQL Enterprise Job Manager (https://www.idera.com/productssolutions/sqlserver/sql-server-agent-job) includes the ability to send notifications for job failures or other events, a web-based interface to enable remote SQL job management, and the ability to create, delete, or edit a single job and push it out to multiple servers. The third feature makes it easy to ensure that every server you manage has the most current version of your maintenance jobs.

CONCLUSION

If you have a lot of SQL jobs and SSIS packages that you have to manage, you have a difficult task. The tips provided in this white paper will help ease your burden by simplifying the administration of the jobs and packages.

Using centralized job servers will bring your servers into focus while alleviating a lot of the persistent performance issues that accompany the execution of SQL jobs and SSIS packages on production servers. It also eases the configuration and administrations of many availability and disaster recovery scenarios.

The SSIS catalog in SQL Server 2012 and newer provides some new features that make administering and troubleshooting SSIS pages easier. This tool automatically performs extensive logging of the internal steps of an SSIS package that can tell you exactly what failed and why. There is no longer a need to rig custom logging or step through a package to troubleshoot package failures.

Lastly, look at the big picture. Use one of the tools available for managing and viewing jobs across your enterprise, like one of Idera's job management tools. Whether you choose to go with a free option or a paid option, you will undoubtedly get a tremendous amount of value out of these tools and be able to quickly identify and resolve SQL job collisions.

IDERA

IDERA understands that IT doesn't run on the network – it runs on the data and databases that power your business. That's why we design our products with the database as the nucleus of your IT universe.

Our database lifecycle management solutions allow database and IT professionals to design, monitor and manage data systems with complete confidence, whether in the cloud or on-premises.

We offer a diverse portfolio of free tools and educational resources to help you do more with less while giving you the knowledge to deliver even more than you did yesterday.

Whatever your need, IDERA has a solution.

SQL Enterprise Job Manager

Monitor and Manage Agent Jobs Across Multiple SQL Servers

SQL Enterprise Job Manager provides comprehensive monitoring of SQL Server agent jobs across the entire SQL Server landscape. See what is scheduled to run and when and get up to the minute status information. View jobs by instance or drill down into specific jobs to see alert settings, upcoming jobs, and history details.

- Monitor SQL Server agent jobs across the enterprise
- Easily view job status and details with familiar calendar layout
- Chain jobs together and build automated workflows
- Configure and manage jobs that run over multiple instances
- Set alerts to be notified of potential issues
- Web-based architecture simplifies deployment

Start for FREE

To learn more visit idera.com today!