Abstract

This paper will discuss tuning techniques for Microsoft SQL Server 2005 & 2008 to enable them to perform well on a multi-node server, such as the IBM x3950 and x3950 M2. Tweaks that are not specific to multi-node scalability, such as the use of large pages, are not discussed.

Introduction

A single-node or non-scalable system based on recent Intel x86 processors has all its memory, slots, and IO local to all the processors in the system. There is no performance penalty for using one area of memory over another.

The situation is different when we go to a multi-node system. In that case, some of the memory, slots, IO, and processors are local (on the same node) and some are remote (on a different node). Accessing remote resources incurs a performance penalty, which can be substantial. When the operating system and application software are not configured properly to deal with this fact, system performance can suffer dramatically.

The Problem

A 2-node system consists of two hardware nodes connected via scalability cables. In this configuration, 50% of the memory is local and 50% is remote, as viewed from each node. With no NUMA-awareness at all, when a program allocates memory, it has a 50/50 chance of allocating memory on the local node, and which memory is allocated (local or remote) can change between invocations due to the memory usage of other applications at invocation time. Application performance can be drastically different on the same system depending on where memory was allocated, locally or remotely.

A 4-node system has 25% of the memory local and 75% remote, relative to each node. This means that a program has a 75% chance of allocating slow memory on a non-NUMA aware OS. On an 8-node system, those chances go up to 7 out of 8—almost 88%. And on top of that, half of the nodes in an 8-node x3950 are two remote hops away from each node, meaning even longer latency. Poor performance is almost guaranteed, unless software is configured properly.

The Solution

To maximize performance, it is best to keep processing/memory/IO as local to each node as possible, to minimize scalability traffic (remote node access). The first step to doing this is to use a NUMA-aware operating system. Windows Server 2003 and Windows Server 2008 are both NUMA-aware and that awareness functions properly on the x3950 and x3950 M2. Windows Server 2003/2008 automatically detect each hardware node and sets them up, processors and memory, each as a software NUMA node. When an application that is running on a multi-node x3950 requests memory, the OS will automatically try to provide that memory on the same NUMA node that the application is running on. This keeps the application and its data all on the same node, reducing remote traffic and maximizing performance. Localizing the application's input and output, networking and storage, will further increase performance. This can be done with process affinity to pin an app to a specific hardware node where its networking and storage controllers physically reside.

The situation gets more complicated when the application requires the resources of more than one hardware node. The application itself then needs to be NUMA-aware and configured properly for the system it is running on. This is the case with SQL Server 2005 & 2008, which are both NUMA aware. However, this awareness isn't fully automatic. There are several configuration parameters that need to be set to optimize SQL Server's performance on a multi-node x3950.

Note that even though SQL Server 2000 has some NUMA-awareness as of SP4, it is very minimal. Users of NUMA systems are strongly encouraged to move off of SQL Server 2000 and onto SQL Server 2005 or 2008 for best performance.

Setting up the SQL Affinity Mask

The first step is to decide which processor threads in the system will be allocated to SQL Server and then set SQL's processor affinity mask, using **sp_configure**. This will lock SQL Server to the same specific processors all the time. For example, consider a 2-node 8P system using single-core processors without Hyper-Threading (so 8 total CPU threads). To assign SQL Server to all processors, give it an affinity mask of 0xFF. 0xF0 are the 4 processors on one hardware node (node 1) and 0x0F are the 4 processors on the other (node 0). The more processor threads (processors, cores, Hyper-Threads), the longer the mask will be. For example, for the same 8P 2-node system with quad-core processors and no Hyper-Threading (so 32 total CPU threads), the affinity mask would be 0xFFFFFFFF, to use all cores. 0xFFFF0000 are the cores on node 1 and 0x0000FFFF are the cores on node 0.

Setting up SoftNUMA Nodes

With that done, the next step is to configure SQL Server's softNUMA nodes, which essentially partition SQL Server. Work that comes into a certain softNUMA node is handled by that softNUMA node exclusively; the work will not spill over to other softNUMA nodes. This keeps the processors, caches, memory, and data related to the work all local to that softNUMA node. The higher the locality, the better the performance will be.

For starters, add one softNUMA node to SQL for each hardware node that has any processors affinitized to SQL. Continuing the 32-thread example above where the 2-node's SQL affinity mask is 0xFFFFFFFF, you would create two softNUMA nodes in SQL Server, one having the mask 0xFFFF0000 and the other having the mask 0x0000FFFF.

Creating softNUMA nodes is done via the registry. Open regedit and drive to `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\<#>\NodeConfiguration`, where <#> is **90** for SQL Server 2005 and <#> is **100** for SQL Server 2008. Once there, create a new key for each softNUMA node named "Node*X*", where *X* is the node number starting from zero and increasing by 1. Then for each of those keys, create a DWORD value "CPUMask" and set it to the processor affinity mask you want to assign to that softNUMA node. Continuing the example:

Key Name	DWORD Value Name	DWORD Value
Node0	CPUMask	0x0000FFFF
Node1	CPUMask	0xFFFF0000

Once you are done with that, start up SQL Server and watch the messages. You should see related lines as follows:

```
2007-10-11 12:48:36.45 Server      Processor affinity turned on: processor mask
0x00000000ffffffff. Threads will execute on CPUs per affinity mask/affinity64 mask config
option. This is an informational message; no user action is required.
```

```
2007-10-11 12:48:46.51 Server      Multinode configuration: node 0: CPU mask:
0x000000000000FFFF Active CPU mask: 0x000000000000FFFF. This message provides a
description of the NUMA configuration for this computer. This is an informational message
only. No user action is required.
2007-10-11 12:48:46.51 Server      Multinode configuration: node 1: CPU mask:
0x00000000FFFF0000 Active CPU mask: 0x00000000FFFF0000. This message provides a
description of the NUMA configuration for this computer. This is an informational message
only. No user action is required.
```

Check this information over carefully to be sure it is correct. SQL Server will not error out if it finds a problem- it will just change to different values. Check the following:

- Check the overall processor affinity mask and that processor affinity is enabled.
- Verify that the Active CPU mask for each softNUMA node is the same as the requested CPU mask.
- Sometimes SQL Server decides to reverse the requested CPU masks for softNUMA nodes 0 & 1. If you find this is the case, simply reverse them in the registry (rename the key "Node1" to "Node0" and vice-versa).

For every change you make, stop and restart SQL to verify that the change took properly.

You can create as many of these softNUMA nodes as you want- one per CPU thread if you wanted. Just note that softNUMA nodes can not have overlapping affinity masks and they should not be set to cross over hardware NUMA boundaries. The optimum number of softNUMA nodes to create varies by system and workload. Setting up one softNUMA node per hardware NUMA node is a good place to start. That will minimize internode traffic and show the biggest performance gains.

Setting up the SQL Server Network Connection Affinity

Now that the softNUMA nodes are configured, you need to assign work to each softNUMA node. This is done by configuring network connection affinity. The idea is that work coming into SQL Server is given to a certain softNUMA node based on what connection the work came from. Connection affinity can be set by IP address (which IP address you connect to determines your softNUMA node) or by IP port (which port you connect to determines your softNUMA node). For highest locality, it is best to have a network adapter physically residing in each hardware node, each with its own IP address.

Let's continue our example by setting port-based connection affinity. Start by opening the SQL Server Configuration Manager and expand the SQL Server 2005/2008 Network Configuration tree on the left. Double-click on TCP/IP to open its properties. If "Listen All" is enabled, then SQL Server will be listening for connections on all of the IP addresses the server has. In that case, you will set the connection affinities in the "IP All" section of the "IP Addresses" tab. If "Listen All" is disabled, you will set the connection affinities for the specific addresses that are enabled.

Assuming the use of "Listen All", go to the "IP All" section of the "IP Addresses" tab and go to the "TCP Port" line. This is where you will list the IP ports that SQL Server will be listening on- on all IP addresses defined on the server, including loopback. When you list the ports, you can provide an optional bitmask to affinitize each port to a certain softNUMA node. For example, say the "TCP Port" string was:

```
1433,1434[0x1],1436[0x2]
```

This means that users that connect to SQL Server via any server IP address on port 1433 can have their work assigned to any softNUMA node (port 1433 is not affinitized). Connections via port

1434 will be affinitized to softNUMA node 0 and connections via port 1436 will be affinitized to softNUMA node 1. Note that the softNUMA affinities are bitmasks, which means:

softNUMA node 0 = 0x1
softNUMA node 1 = 0x2
softNUMA node 2 = 0x4
softNUMA node 3 = 0x8
softNUMA node 4 = 0x10
softNUMA node 5 = 0x20
softNUMA node 6 = 0x40
and so on.

After that is all setup, SQL Server will now show messages like this when it is started:

```
2007-10-11 12:48:56.57 Server      Server is listening on [ 'any' <ipv4> 1433].
2007-10-11 12:48:56.62 Server      Server is listening on [ 'any' <ipv4> 1434].
2007-10-11 12:48:56.62 Server      SQL Network Interfaces initialized listeners on node 0
of a multi-node (NUMA) server configuration with node affinity mask 0x0000000000000001.
This is an informational message only. No user action is required.
2007-10-11 12:48:56.59 Server      Server is listening on [ 'any' <ipv4> 1436].
2007-10-11 12:48:56.59 Server      SQL Network Interfaces initialized listeners on node 1
of a multi-node (NUMA) server configuration with node affinity mask 0x0000000000000002.
This is an informational message only. No user action is required.
```

No message follows the line for port 1433, so it is not affinitized. The line following port 1434 shows that it is affinitized to node 0 (mask 0x1). The line following port 1436 shows that it is affinitized to node 1 (mask 0x2). Verify that the information in these messages matches your desired connection affinities. For every change you make, stop and restart SQL to verify that the change took properly.

Finally, the users need to be configured so they connect to the desired port. This is done with the SQL Native Client Configuration. Group users together who work on the same parts of the database (or the same database) to the same softNUMA node. This will keep that part of the database local to that softNUMA node. If database tables have to bounce back and forth between softNUMA nodes (hardware nodes in this case) because users on both softNUMA nodes are constantly using them, performance will suffer. The overall idea is to keep data local to each hardware node, minimizing slow remote data access.

Non-NUMA Commands

We've found that some SQL commands respect the SQL Server NUMA configuration, while others do not. Queries that manipulate data in the database work well with NUMA. Database maintenance commands, such as **reindex**, do not. Because of this, those commands can take a lot longer on a NUMA system than on a non-NUMA system. To work around this issue, to keep maintenance windows as short as possible, it is best to pin SQL Server to only one hardware NUMA node when running such tasks. Just use **sp_configure** to change the overall affinity mask to force SQL Server to reside on only one hardware NUMA node. Then run the maintenance tasks. When done with those, reset the SQL Server affinity mask for normal operations.

Other Notes

1) Once you have SQL Server configured properly for NUMA, it is best to reboot the server and then start SQL Server, especially if large pages are being used. Rebooting clears out memory, allowing SQL Server to grab large contiguous blocks of memory for best performance.

2) Don't allow SQL Server to grab too much memory, resulting in paging. Be sure there is at least a couple of GB free, or performance can get really poor. The **sp_configure** setting "max server memory" controls this.

Review

In order to get the best performance possible on a multi-node x3950 or x3950 M2, the amount of remote-node references must be minimized. The higher the data locality, the better the overall performance will be. In order to optimize SQL Server 2005 or 2008 on a multi-node x3950 or x3950 M2, the following must be done:

1) pin SQL Server to certain processors by settings its affinity mask
2) setup softNUMA nodes in SQL Server, to group local CPU threads together to handle incoming work
3) setup connection affinity in SQL Server, to assign networking connections to softNUMA nodes
4) assign users to specific network connections. Have similar users share the same network connections, meaning that they share the same softNUMA node, keeping their shared data local to that part of the server.