<https://sqlundercover.com/2019/12/03/troubleshooting-resource-semaphore-and-memory-grants/>

Memory grants, now here’s a fun thing that can pretty much take your SQL Server to its knees.

**The Symptoms**

The first thing that you’re going to notice is that your SQL Server is going to be running slowly and I mean, slooooooow!

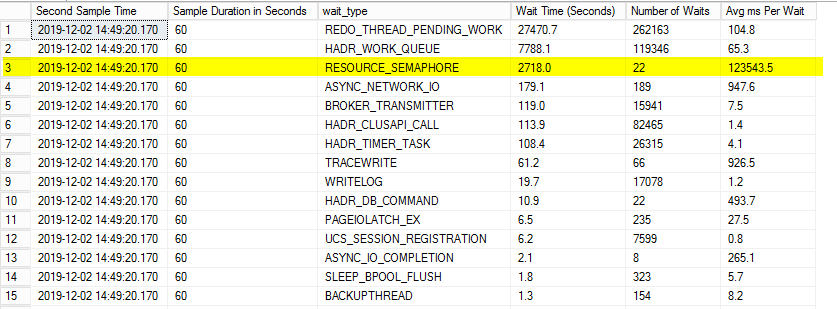
When I’ve got a server on a serious go slow, the first thing that I like to look at is the wait stats, I want to know what SQL’s waiting for.

dm\_os\_waitstats holds all the information that you need on wait stats, the downside however is that it gives you a running total since the server was started. That can be useful for some things but isn’t going to be great when we want to know exactly what’s going on now.

There are a few scripts out there that you can use for this, Brent Ozar’s sp\_blitzfirst ( <https://github.com/BrentOzarULTD/SQL-Server-First-Responder-Kit>) is a great go to (it will also give you details of any outstanding memory grants, very relevant to this particular post) but I tend to use a modified version of Paul Randal’s [wait stats script](https://www.sqlskills.com/blogs/paul/capturing-wait-statistics-period-time/).

Any monitoring tool will also give you this information.

If you’ve got an issue with memory grants then you’re very likely to see RESOURCE\_SEMAPHORE waits pretty high up on your wait stats list.

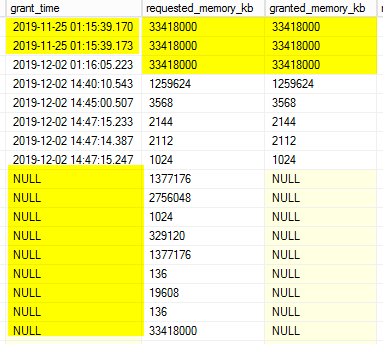
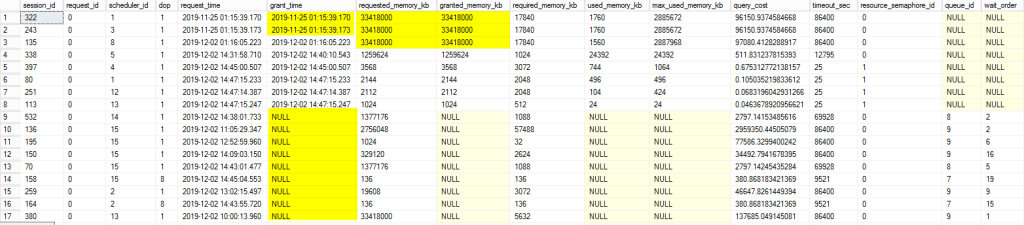


These aren’t something that I ever want to see and if I do I am straight away going to go looking for the cause. Basically what’s that’s telling you is that you’ve got processes waiting on a memory allocation. This is going to be because SQL Server hasn’t got enough memory to dish out to that particular process.

What’s going on here is that every time a query runs, it’ll ask for a certain amount of memory. A happy SQL Server will serve up a tasty slice of memory for that query to run in. The problem comes when the query is asking for a bigger slice of the pie than SQL has available. In that case the query will need to wait until SQL has enough free to give the query what it wants, that’s when the RESOURCE\_SEMAPHORE wait starts ticking up.

We can look a little deeper into this by querying sys.dm\_exec\_query\_memory\_grants to see exactly what has already been granted and what is waiting for a grant

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM sys.dm\_exec\_query\_memory\_grants  ORDER BY granted\_memory\_kb DESC |



This looks like a bad situation for a SQL Server to be in, there are a few things that catch my eye straight away.

Firstly, we’ve got a large number of processes with a grant\_time of NULL. That means the process is waiting on a memory grant, that goes hand in hand with the RESOURCE\_SEMAPHORE waits that we’ve been talking about. As soon as you see this going on, it’s a warning sign that SQL Server is in a spot of bother.

The next question is why hasn’t SQL Server got a big enough slice of the pie to give to these poor processes. My eye now turns to the top of the table and the granted\_memory\_kb column.

WOW, check out those top 3 rows, those processes have got around about a 32GB memory grant each, that’s 96GB granted to just those three queries.

*Interesting Fact:* Each of those grants happens to be 25% of SQL’s query memory. SQL Server will cap a query’s memory grant to 25% to prevent a single query for taking all the memory.

Unfortunately, in our case we’ve got three process all hogging a 25% slice. That means that SQL’s effectively running on 25% of it’s query memory, there’s not much of that pie left to go around, no wonder it’s struggling.

Another thing that I’ve noticed is the grant\_time, two of those queries have been holding their grants (total of 50% of the server’s query memory) for a week! There’s some real filth going on there.

**Further Investigation**

Now that we’ve identified the problem processes we can do a little more digging if we wanted to. Personally, in this particular case I knew exactly what those processes were and happily killed them off but you might want to find out more.

sys.dm\_exec\_query\_memory\_grants gives us some handy information, we’ve got the SPID so it’s easy enough to figure out who’s running the query.

We also get the plan handle and the query handle, pop these into the following queries and you can get the query text and the execution plan.

|  |  |
| --- | --- |
| 1  2 | SELECT text  FROM sys.dm\_exec\_sql\_text(<plan handle>) |
| 1  2 | SELECT query\_plan  FROM sys.dm\_exec\_query\_plan(<plan handle>) |

Thanks for reading and I hope that if you’re ever battling memory grant problems, you’ll find this useful.

<https://www.mssqltips.com/sqlservertip/2827/troubleshooting-sql-server-resourcesemaphore-waittype-memory-issues/>

# Troubleshooting SQL Server RESOURCE\_SEMAPHORE Waittype Memory Issues

##### Problem

Today, one of our SQL Server instances was performing very slowly. When I logged in to the database server to do some initial checks I noticed it was memory pressure from the initial observation. Next we had to find what out was causing our instance to have memory pressure. When I checked the wait types for the transactions, the RESOURCE\_SEMAPHORE wait was the issue for most of the transactions. In this tip I will describe this issues and how to find which query or transaction is causing the memory pressure

##### Solution

When I checked the wait types for all transactions, the RESOURCE\_SEMAPHORE wait was the wait type for most of the transactions in addition to some page IO waits. The page IO waits were also due to memory pressure because those transactions were not able to get enough memory to perform their operation.

### Resource Semaphore Wait

Before moving on, I would like to shed some light on the Resource Semaphore wait so you can better understand how memory is granted to SQL Server queries.

When SQL Server receives a user query, it first creates a complied plan, then an execution plan is created based on the complied plan. When SQL Server creates a complied plan it calculates two memory grant parameters called "required memory" and "additional memory". Required memory is the minimum memory needed to run a sort and hash join. It is known as required because a query would not start without this memory available. Additional memory is the amount of memory needed to store temporary rows in memory. This is known as additional because a query can be stored on disk if there is not enough memory available.

First, the server calculates how much memory is needed for any given query to execute. This is generally the sum of required memory and additional memory, but if your instance is using parallelism then the needed memory would be (required memory \* DOP) + additional memory. The server checks if the needed memory exceeds the per query limit, then the server reduces additional memory until the total fits within the limit. This revised size is called requested memory. There is an internal facility within SQL Server known as RESOURCE SEMAPHORE which is used to grant this requested memory to a query. If query is not able to be granted this requested memory by a Resource Semaphore, then that query will be in a waiting state with a RESOURCE\_SEMAPHORE wait type if you query the sysprocesses system table or sys.dm\_exec\_request DMV.

When a Resource Semaphore receives a new request, it first checks if any query is waiting or not. If it finds one, it puts the new query in the queue because the wait queue is designed on a first-come-first-served basis with a small weighting to favor small queries. Resource Semaphore attempts to grant memory when there is no waiting query or when a query returns reserved memory. If it finds enough memory, then the requested memory is granted and the query can start running and if it does not find enough free memory to grant the requested memory then it puts the current query into the waiting queue with a RESOURCE\_SEMAPHORE wait type and your server starts facing memory pressure.

### Identify RESOURCE\_SEMAPHORE Waits

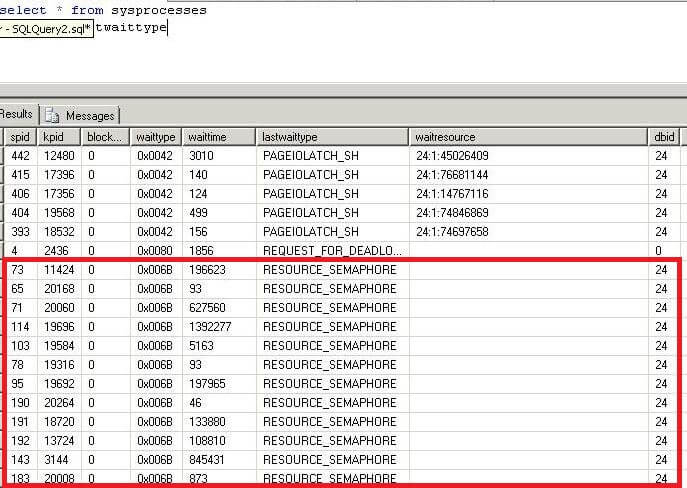
**Step 1**

First, we need to look into our instance to see why memory pressure is occurring within SQL Server. To get an initial overview of all transactions, we can query sysprocesses or we can use the sys.dm\_exec\_requests DMV.

SELECT \* FROM SYSPROCESSES

ORDER BY lastwaittype

Here we can see processes that have a RESOURCE\_SEMAPHORE wait type.

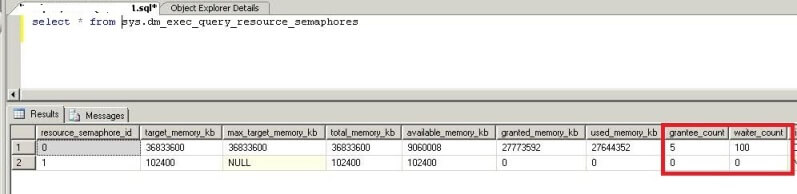


**Step 2**

From the above query, we can see a large number of transactions are waiting with Resource Semaphore wait type. Now we can run the below query to see the status of total number of queries which have been granted memory and the number of queries which have not yet been granted memory.

The output of this DMV returns two rows, one for large queries (resource\_semaphore\_id is 0) and another one for small queries (resource\_semaphore\_id is 1) specially less than 5 MB. Here you can get the total granted memory and total available memory for the instance. See the numbers on **grantee\_count** and **waiter\_count**, the grantee\_count is the number of queries which have their memory and the waiter\_count is the number of quires which are waiting in queue to get memory. So here we can see approximately 100 queries are waiting to get their requested memory.

SELECT \* FROM sys.dm\_exec\_query\_resource\_semaphores

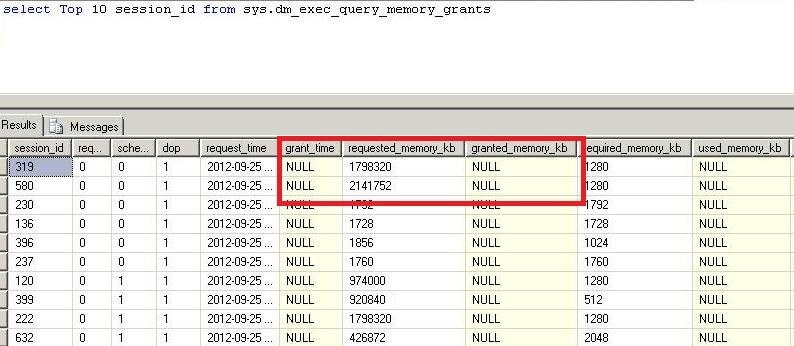


**Step 3**

Now we will get the details of all queries which are waiting in queue to get their requested memory. We will use DMV sys.dm\_exec\_query\_memory\_grants to get the total number of queries which are waiting in queue to get their memory along with the details. The columns **grant\_time** and **granted\_memory\_kb** will be NULL for those queries which are waiting to get their requested memory. You can see in the below screenshot the requested memory amount and their waiting state because their grant\_time and granted\_memory\_kb value is NULL.  We can also get the plan\_handle and sql\_handle of all queries with this DMV. We will use these values later to get the exact queries.

Note: there are too many columns to show, so this is just a partial listing of all columns.

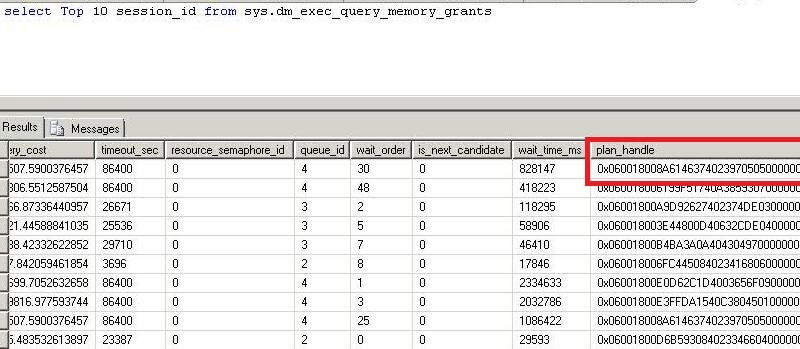
SELECT \* FROM sys.dm\_exec\_query\_memory\_grants



**Step 4**

Now we will find the memory intensive queries. We can see the requested memory for all waiting queries. Here we can see the requested memory is too large for most of the transactions. We will get the plan\_handle of all these queries to get the exact SQL text to look into the query plan.

select top 10 \* from sys.dm\_exec\_query\_memory\_grants

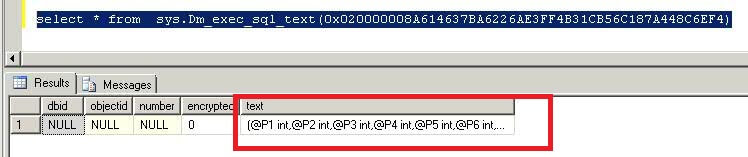


**Step 5**

Now we will use the above plan\_handle and sql handle to get the SQL code.

Run the below statement to get the SQL code using the sql\_handle from the above query.

SELECT \* FROM sys.dm\_exec\_sql\_text(sql\_handle)



We can also get the SQL plan using the plan\_handle from the query in Step 4.

SELECT \* FROM sys.dm\_exec\_sql\_plan(plan\_handle)

### Conclusion

Now that we have found the memory intensive queries and their execution plans, our next step is to look into these queries and figure out how to tune them.  We should look into bad or missing indexes used in the query and implement proper indexing.  In our case, it was bad indexes which were causing the memory pressure. After implementing proper indexing the same query was running with much less requested memory.

##### Next Steps

* Use this tip to identify the queries which are consuming more memory and putting remaining transactions into wait state due to lack of memory.
* Also look at the other columns of the above DMVs and correlate them to each other for better analysis and understanding of performance issues.
* These DMVs should provide an ample amount of information so you can identify the issue.
* Read more tips on [Performance Tuning](https://www.mssqltips.com/sql-server-tip-category/9/performance-tuning/) to improve your system performance.