Por Eladio Rincón

Eventos Extendidos en la Práctica

Averigüe donde se gasta el tiempo de una consulta

En general, mientras se procesa una consulta, puede estar en uno de dos estados: usando recursos de servidor o esperando por recursos de servidor. Utilice los Eventos Extendidos para ver dónde se gasta la mayor parte del tiempo de sus consultas.

Como DBA, debiéramos de tener control de todo lo que pasa en el servidor. Sin embargo, en el mundo real, a veces las cosas se descontrolan, y necesitamos ser capaces de identificar qué es lo que sucede rápidamente y de forma fiable. En SQL Server 2008 R2 existe un conjunto funcional llamado *Extended Events* que puede ayudarnos.

Se trata de un sistema de manejo de eventos que puede utilizarse en una amplia gama de escenarios de monitorización y resolución de problemas. El sistema suministra sentencias T-SQL del tipo DDL (*Data Definition Language*) para crear y modificar una sesión "Extended Events", así como obtener datos y metadatos de la sesión mediante Vistas Dinámicas (DMV) y Vistas de Catálogo.

En este artículo no damos cobertura a conceptos introductorios, de forma que, si no le resulta familiar el trabajo con *Extended Events*, tómese nos minutos para hojear la información de los siguientes enlaces:

- Introducing SQL Server Extended Events
- SQL Server Extended Events Concepts
- Using SQL Server Extended Events
- SQL Server Extended Events How-to Topics

Este artículo está enfocado a un caso concreto: Averiguar dónde se gasta en tiempo de una consulta. El enfoque es similar al de la metodología *Waits & Queues*: mientras se procesa una consulta, puede encontrarse en uno de dos estados: usando recursos de servidor o esperando para poder utilizarlos. Veamos cómo los *Extended Events* pueden ayudarnos a encontrar la información que necesitamos.

El enfoque Waits & Queues

Como DBA experimentado, probablemente le resulte familiar la metodología *Waits & Queues* que sigue el principio de:

> *Tiempo de respuesta de la conexión =Tiempo de servicio + Tiempo de espera*

donde

- *Tiempo de servicio (Service time)* es el tiempo que SQL Server necesita para procesar la consulta, y
- *Tiempo de espera (Wait time),* es el tiempo a esperar para usar los recursos y procesar la consulta

Aunque la primera vez que escuche algo acerca de estas técnicas, fue en 2003, en el artículo de **Tom Davidson** de SQL Server Magazine, "*Opening Microsoft's Performance-Tuning Toolbox*", cuando realmente se popularizaron fue cuando SQL Server 2005 presentó el soporte DMV incluyendo *sys.dm_os_wait_stats* y *sys.dm_io_virtual_file_stats*, y la publicación del nuevo enfoque basado en supported and documented queries. Puede leerse más sobre éste enfoque en el artículo sobre buenas prácticas de MSDN SQL Server 2005 Waits and Queues.

La ventaja de esta técnica de resolución de problemas es que somos capaces de identificar cuellos de botella en los recursos del servidor a través de un enfoque global. Se pueden obtener resultados rápidamente consultando las DMV relacionadas o generar un informe más elaborado usando algo como la herramienta interna que hemos desarrollado para nuestro servicio SolidQ Health Check, que produce un informe Excel como el del ejemplo de la Figura 1.

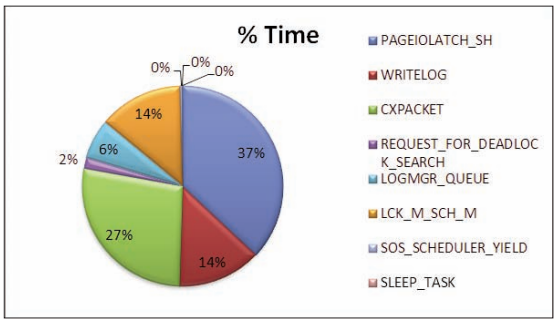


Figura 1: Informe Excel del servicio *SolidQ's Health Check*

En otros casos, ya conocemos qué sesión o consulta está originando los problemas en el servidor y no necesitamos el enfoque a nivel de instancia. En este escenario, podemos usar las DMV de tipo sys.dm_exec_* o *SQL Server Profiler* para obtener esa información, como el tiempo de CPU, las lecturas físicas y lógicas, el tiempo de CLR y el tiempo total. Sin embargo, todavía necesitaríamos algo a nivel de consulta para identificar dónde transcurre el tiempo de la consulta en sí.

Configurando el *Extended Event*

Para crear una sesión "*Extended Event*", primero necesitamos identificar la sesión a analizar. Más tarde, podemos ejecutar el *script* del Listado 1, donde *session_id* 54 es la conexión que necesitamos analizar.

Observe que establecemos el filtro para la *session_id* y el destino de la captura es un fichero (*asynchronous_file_target*). Además, hay un filtro en *sqlos.wait_type (opcode =1)* que indica que queremos capturar solamente eventos finales para *wait_info*.

Tras crear la sesión *Extended Event* comenzamos por ejecutar la siguiente sentencia:

Listado 1: Creación la sesión de Extended Event

```
CREATE EVENT SESSION xe_waits
ON SERVER
ADD EVENT
    sqlserver.sql_statement_completed (
                    ACTION (sqlserver.sql_text)
        WHERE  (sqlserver.session_id = 54)
    ) ,
ADD EVENT
          sqlos.wait_info (
       WHERE  (sqlserver.session_id = 54 AND opcode =
1)
          )
ADD TARGET package0.asynchronous_file_target
    (set filename=N'c:\temp\xe_waits.xel',
    metadatafile=N'c:\temp\xe_waits.xem')
WITH
(
    MAX_MEMORY = 4096KB,
    EVENT_RETENTION_MODE = ALLOW_SINGLE_EVENT_LOSS,
    MAX_DISPATCH_LATENCY = 1 SECONDS,
    MEMORY_PARTITION_MODE = NONE,
    TRACK_CAUSALITY = ON,
    STARTUP_STATE = OFF
);
```

```
ALTER EVENT SESSION xe_waits
ON SERVER
STATE = START;
```

Consultando los resultados

Para analizar los datos registrados en los ficheros de seguimiento (trazas) en el *Extended Event*, necesitamos detener la sesión y ejecutar unas consultas que mezclan lenguaje T-SQL con extensiones *XQuery*. No soy un experto en XML, y me llevó un rato escribir las consultas para obtener los resultados que buscaba; recomendaría usar las siguiente consultas como plantillas y personalizarlas para sus necesidades.

Listado 2: Stopping and Dropping the Extended Event Session

```
ALTER EVENT SESSION xe_waits
ON SERVER
STATE = STOP;

DROP EVENT SESSION xe_waits
ON SERVER;
```

Listado 3: Consulta para devolver estadísticas de tiempo de espera

```
– wait stats
SELECT
        COUNT(*) count
       , k.map_value wait_type
       , SUM(wait_time) wait_time
       , SUM(signal_wait_time) signal_wait_time
       from (
SELECT
        t.x.value('(data/value)[1]', 'int') as k
      , t.x.value('(data/value)[3]', 'int') AS
        wait_time
      , t.x.value('(data/value)[6]', 'int') AS
        signal_wait_time
from (
select CONVERT(xml, event_data) as xml_data
from sys.fn_xe_file_target_read_file
('c:\temp\xe_waits*.xel', 'c:\temp\xe_waits*.xem',
  null, null)
) v
CROSS APPLY xml_data.nodes ('//event[@name=''wait_info''']')
  as T (x)
) v
join sys.dm_xe_map_values k
on  v.k = k.map_key
where k.name = 'wait_types'
group by k.map_value
```

	cpu_time	duration	reads	writes	sql_text
1	16	162009	932	0	SELECT * from HumanResources.vEmployee
2	0	145008	0	0	select object_name as event, CONVERT(xml, event_data) a...
3	15	300017	1566	0	SELECT * from HumanResources.vEmployee
4	62	122007	0	0	-- wait stats SELECT COUNT(*) count , k.map_value wait_...
5	16	52003	0	0	-- query time SELECT t.x.value('(data[@name="cpu"]/valu...
6	0	4000	0	0	DBCC DROPCLEANBUFFERS

Figura 2: Resultados de la consulta anterior

Normalmente, antes de analizar los datos, debemos detener la sesión *Extended Event* para consolidar los datos en un fichero y entonces destruirla (mediante DROP, si es adecuado), ejecutando los comandos que se indican en el Listado 2, pág. 41.

Para obtener las estadísticas de espera a nivel de consulta, utilizamos la consulta del Listado 3, que devuelve el conjunto de información que se muestra en la Figura 2.

Además, podemos obtener información detallada acerca de las consultas ejecutadas si lanzamos la consulta que muestra el Listado 4. La Figura 3 correspondiente, muestra sus resultados.

Listado 4: Consulta para obtener información detallada sobre consultas ejecutadas

```
– query time
SELECT
    t.x.value('(data[@name=''cpu'']/value)[1]', 'int') as cpu_time
  , t.x.value('(data[@name=''duration'']/value)[1]', 'int') AS duration
  , t.x.value('(data[@name=''reads'']/value)[1]', 'int') AS reads
  , t.x.value('(data[@name=''writes'']/value)[1]', 'int') AS writes
  , t.x.value('(action/value)[1]', 'varchar(max)') AS sql_text
from (
select CONVERT(xml, event_data) as xml_data
from sys.fn_xe_file_target_read_file
('c:\temp\xe_waits*.xel', 'c:\temp\xe_waits*.xem', null, null)
) v
CROSS APPLY xml_data.nodes ('//event[@name=''sql_statement_completed''']') as T (x)
```

Results | Messages

	count	wait_type	wait_time	signal_wait_time
1	2	WRITELOG	1	0
2	58	NETWORK_IO	633	1
3	632	SOS_SCHEDULER_YIELD	1	0
4	49	PAGEIOLATCH_SH	338	1

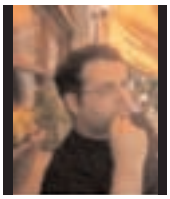
Figura 3: Resultados de la consulta anterior correspondientes al Listado 4

Conclusión

Ahora parece un buen momento para familiarizarse con *Extended Events*, y puede usar los *scripts* de este artículo como un punto de partida práctico para aprender cómo funcionan y suministrar funcionalidad administrativa muy útil al mismo tiempo. Como sucede con cada característica nueva, lleva tiempo y práctica aprender acerca de los *Extended Events*, ganar confianza con ellos y explorar cómo personalizarlos para que se adapten a nuestras necesidades.

En el artículo siguiente, veremos ejemplos de cómo dirigir los datos de sesiones *Extended Event* a un recolector que recibe menos información, porque agrega los datos capturados. Esta es una estructura ideal para mantener almacenes en memoria, en lugar de ficheros, como veremos ya en el próximo número. ■

Sobre EL Autor



Eladio Rincón *(personal blog | SolidQ blog), es MVP de SQL Server desde 2003, CEO de Solid Quality Mentors en Brasil y Director de IT para tecnologías de bases de datos en España. Ha formado parte del equipo de SolidQ desde 2004, centrándose en administración, soporte, proyectos de consultoría y formación en tecnologías relacionadas con SQL Server. Ha dirigido o liderado proyectos por todo el mundo, incluyendo China, Costa Rica, Korea, Méjico, Perú, España, Reino Unido y EE.UU. Eladio colabora con Microsoft en la organización de eventos presenciales y "on-line", como los MSDays. Mantiene su blog personal, además del blog "El Rincón del DBA", junto a otros colegas de SolidQ.*