Joost van den Dool is softwarearchitect bij DAF Trucks in Eindhoven

SQL Server voor developers

OPSPOREN EN VERHELPEN VAN PERFORMANCE BOTTLENECKS

Als ontwikkelaar van gedistribueerde software heb je vaak met databases te maken. Hoe klein het systeem ook is, de behoefte aan dataopslag komt al gauw om de hoek kijken. De database is ook vaak de eerste plek waar het mis gaat bij performanceproblemen. In dit artikel legt de auteur uit hoe je in een productieomgeving de oorzaak van performanceproblemen kunt achterhalen en welke mogelijkheden SQL Server biedt om de performance te verbeteren. Aan het einde kijkt hij hoe je in het ontwikkeltraject al rekening met performance kunt houden.

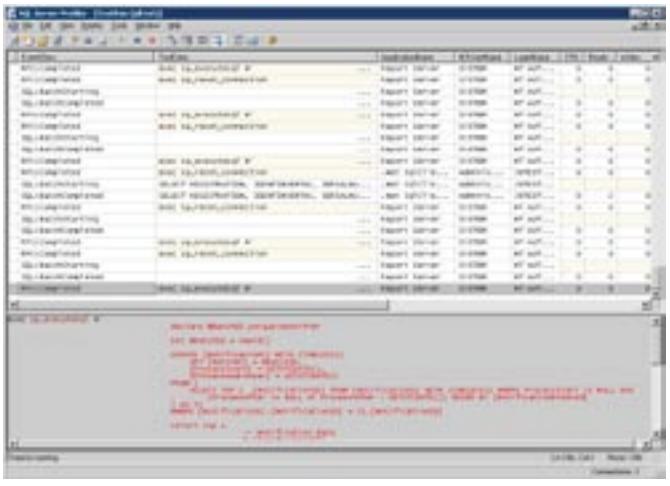
Ik ga er maar even vanuit dat ik niet de enige ben die wel eens een gebruiker of een beheerder aan zijn bureau heeft staan met klachten over de performance van een systeem. Bij voorkeur met termen als het systeem performt niet of het systeem is traag. Vervolgens begint de uitdaging om te achterhalen wat er nu werkelijk aan de hand is en wie of wat de boosdoener is. Gelukkig krijgen we bij SQL Server een hele stapel performancecounters cadeau die ons veel meer inzicht kunnen verschaffen. Mijn ervaring is dat je het beste een iteratief meetproces kunt starten. De eerste stap is het uitvoeren van een globale meting om een eerste indruk te krijgen van het systeem. Vervolgens analyseer je de meetgegevens. Let er op dat je de set van metingen en je analyse goed documenteert. Op basis hiervan kun je conclusies trekken en een specifiekere meting starten om zo via een iteratief proces steeds dichter bij het probleem te komen. Voor een overzicht van zo'n globale meetset verwijs ik je naar de site http://www.sql-server-performance.com/performance_monitor_tips.asp die een schat aan informatie bevat over performancecounters en hun mogelijke waarden. Een andere mogelijkheid in de SQL Server-toolset om inzicht te krijgen in de load op een databaseserver is de SQL Server Profiler; zie afbeelding 1. Met deze tool kun je zien welke queries op de database worden uitgevoerd, door wie ze worden uitgevoerd en wat de tijd en capaciteitsverbruikskenmerken van deze opdracht zijn. Hiermee is het ook mogelijk individuele queries uit deze log te halen en in de query-analyzer los te draaien. Als je in de query-analyzer gebruikmaakt van het query-plan krijg je veel inzicht over hoe je query zich gedraagt.

De oorzaak en wat nu

Na flink wat metingen en het uitproberen van queries denk je de oorzaak te hebben gevonden. Maar wat nu? Welke instrumenten heb ik om de performance te verbeteren? Het allerbelangrijkste zijn de indexen. Goede indexen schelen veel zoekwerk voor de database en zorgen daarmee voor een betere performance. We kennen twee typen indexen: de clustered en de non-clustered index. Bij een clustered index worden de data fysiek in de volgorde van de index gelegd, dit betekent dat er op een tabel maar één clustered index kan liggen. Denk voor een voorbeeld van een clustered index aan het telefoonboek. Hierin zijn de data gesorteerd op plaats, achternaam en vervolgens op voorletters. Je ziet daar dat de gegevens op volgorde van de index geordend zijn. Bij non-clustered indexen wordt een index op een of meer velden gelegd, maar de datavolgorde wordt niet aangepast. Je kunt dus meer non-clustered indexen op een tabel leggen. Denk voor een voorbeeld van een non-clustered index aan een trefwoordenregister, de trefwoorden staan handig op volgorde maar vervolgens moet je zelf nog naar de aangegeven bladzijde bladeren.

Bij het uitvoeren van een query maakt SQL Server een plan, dit plan bevat onder andere welke index hij moet gebruiken voor het vinden van de gegevens. Goede query-plannen leiden tot efficiënte queries. Als de databaseserver een minder goed plan kan maken, moet hij meer werk verrichten om de data boven tafel te krijgen. Om in het plan de juiste index te selecteren, maakt het systeem gebruik van statistieken. Statistieken zijn een soort histogrammen over de data in een tabel. Door de histogramgegevens kan het systeem beslissen welke index het beste resultaat oplevert. Het is belangrijk dat deze statistieken up-to-date zijn, anders krijg je minder efficiënte queries. Normaal staat je database zo ingesteld dat hij de statistieken zelf bijwerkt, maar soms kan dat niet omdat hij dat op een ongunstig moment doet, bijvoorbeeld net als het systeem het heel druk heeft. Het is heel belangrijk om goed over indexen en statistiek na te denken om een goede performance te waarborgen.

Nog een belangrijke performance-beïnvloedende factor is locking. Om locking-problemen goed te detecteren, kun je het beste gebruikmaken van de specifieke counters in SQL Server: Locks counterobject. Een database maakt altijd gebruik van locks om te zorgen dat data consistent blijven. Door ontbrekende indexen of door slechte queries kan het echter gebeuren dat de database locks moet escaleren. Dit betekent dat de database probeert een zo klein mogelijke lock te zetten, maar door de genoemde factoren is er een grotere lock nodig en wordt er dus een lock op meer data gezet,



Afbeelding 1. SQL Profiler (SQL Server 2005

bijvoorbeeld op een page of zelfs een tabel in plaats van op een regel. Dit kan er toe leiden dat andere processen moeten wachten tot de lock weer vrijgegeven wordt voordat zij data kunnen bewerken. Omdat die processen aan het wachten zijn, houden zij data ook weer langer vast en zo kan er een kettingreactie ontstaan die een grote invloed heeft op de performance van het systeem. Goede indexen en goed getunede queries kunnen een hoop leed voorkomen, maar je moet je vooral tijdens het ontwerp bewust zijn van mogelijke locking-problemen. Zodra er meer processen tegelijk moeten worden uitgevoerd, gaat locking een rol spelen. Hoe groter de opdracht, hoe groter de kans op problemen met locking. Het is dus belangrijk om transacties zo klein mogelijk te houden.

Voorkomen is beter dan…

Bij locking kwam het al ter sprake, in je design kun je een heleboel doen om de performance van je systeem te verbeteren. In deze paragraaf bespreek ik een aantal zaken die je kunnen helpen je ontwerp robuust te maken voor performance. Een van de keuzes is of je voor veel of weinig roundtrips ontwerpt. Bij veel roundtrips laat je veel werk door de database doen die erg goed is in het werken met data, het nadeel is dat je veel over de lijn stuurt. Minder roundtrips leiden tot minder netwerkload, maar je moet meer in je programma doen die dat vaak minder efficiënt doet dan een database. Een ander punt bij het ontwerp waarover nagedacht moet worden, is hoe om te gaan met parallelle processen. Zodra processen parallel gaan lopen is het belangrijk om te zorgen dat transacties zo kort mogelijk zijn en dat de volgorde waarin tabellen worden aangesproken over de processen gelijk blijft. Korte kleine transacties zorgen voor korte en meestal ook kleinere locks. Wanneer daarbij de tabellen steeds in dezelfde volgorde worden aangesproken, verklein je de kans op deadlocks aanzienlijk. Ook het ontwerp van je database is nog een belangrijke factor. Normaliseren is erg goed, maar je kunt overdrijven. Als je veel joins nodig hebt om bij je data te komen kost dat performance. Houd ook rekening met je datatypen. Kies ze zo klein mogelijk, want alle ruimte die je niet gebruikt, hoeft niet van schijf gelezen te worden en hoeft ook niet over het netwerk heen. Denk naast dit alles ook heel goed na over je indexen. De juiste indexen, afgestemd op de vragen van het programma, leveren veel performance op.
Bij het bouwen van je software is het erg goed om je queries in de query-analyzer te draaien met het query-plan aan. Het lezen van het query-plan levert veel informatie op over je query en hoe die gebruikmaakt van je indexen. Zo kun je jouw queries testen op het gegeven of ze voldoen aan je (performance) verwachtingen.

Tips & trucs

Tot slot wil ik nog wat handige features de revue laten passeren die je kunnen ondersteunen bij het tunen van de performance.

- Het is vaak lastig om een representatieve dataset te krijgen zodat je goed kunt testen. Sinds SQL Server 2000 is er naast de back-up- en restore-optie ook de mogelijkheid om een database te detachen en vervolgens (op een andere server) te attachen. Hierbij worden de datafiles van de database rechtstreeks gekopieerd en in een andere instance van SQL Server geladen. Het voordeel is dat je allerlei instellingen op databaseniveau meteen meekrijgt en dat je minder schijfruimte nodig hebt dan bij een restore. Bij een restore heb je ruimte nodig voor de restore-file, de targetdatabase en tijdelijke ruimte om de database te vullen.
- Bij het testen van je queries in de query-analyzer krijg je alleen de eerste keer een objectief resultaat. Als de query voor de tweede keer wordt gedraaid heeft SQL Server het query-plan al in de cache geladen evenals de data. Om steeds weer in een objectieve situatie te beginnen kun je het commando “DBCC dropcleanbuffers” gebruiken. Deze opdracht zorgt ervoor dat de cache leeggemaakt wordt, zodat je een objectief beeld van je query krijgt.
- Wanneer het aantal roundtrips beperkt moet blijven, of als je de data in XML binnenkrijgt, kan het handig zijn om vanuit de code geen SQL-queries te gebruiken, maar xml-documenten richting een stored procedure te streamen die de data insert in de juiste tabellen. Deze methode gebruiken wij onder andere als de data via een webservice binnenkomt. De webservice voert een schemacheck uit en vervolgens wordt de xml aan de database aangeboden. Voor een voorbeeld van deze constructie zie codevoorbeeld 1 en 2.
- Een andere mogelijkheid om het aantal roundtrips terug te brengen is gebruik te maken van stored procedures in .Net. Deze optie is mogelijk vanaf SQL Server 2005. Functies die niet in SQL Server zijn ingebouwd, denk bijvoorbeeld aan string-manipulatie, worden nu mogelijk doordat je .Net stored procedures kunt gebruiken. Het is een krachtig middel, maar je dient je wel bewust te zijn dat je code dicht tegen SQL Server aandraait en dus robuust moet zijn.

```
/*
  Voer dit script uit op de northwind database
*/

CREATE PROCEDURE xs_InsertRegion
        @RegionXML text
as
BEGIN
        declare @xml int

        exec sp_xml_preparedocument @xml out, @RegionXML

  INSERT INTO Region (RegionID, RegionDescription)
  SELECT  RegionID,
    Description
  FROM OPENXML(@xml, '/ROOT/Region')
  WITH
  (
    RegionID  int  './RegionID',
    Description  varchar(255) './Description'
  )

        exec sp_xml_removedocument @xml
END
GO
```

Codevoorbeeld 1.

```
/*
  Voer dit script uit op de northwind database nadat
  de xs_InsertRegion is toegevoegd. Het resultaat is data de
  Region die in de xml is vastgelegd wordt opgeslagen in de Region tabel
*/
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
  <Region>
    <RegionID>5</RegionID>
    <Description>Europe</Description>
  </Region>
</ROOT>'

exec xs_InsertRegion @doc
```

Codevoorbeeld 2.

Dit artikel is verre van volledig. Er is nog veel meer te zeggen over performance en over performance in relatie met databases. Ik heb geprobeerd een inleiding te geven op databases voor developers en dan specifiek op het gebied van performance. Ik hoop dat dit artikel helpt je bewust te maken wat je kunt doen om je ontwerp robuust te maken voor performance en als het mis gaat een idee te geven wat je kunt doen om inzicht te krijgen in wat precies het probleem is.

Joost van den Dool is softwarearchitect bij DAF Trucks in Eindhoven. Hij is MC DBA en MCSD .NET-gecertificeerd. Voor vragen is hij te bereiken op Joost.van.den.dool@daftrucks.com

Referenties

- www.sql-server-performance.com
- http://www.microsoft.com/sql/