Porsche mit Anhänger?

Diese Frage stellt sich beispielsweise bei der Kombination des SQL Servers mit Navision. Der Einsatz der Server-Datenbank von Microsoft veranlasst meist zu großer Hoffnung, wenn die native Datenbank des ERP-Systems (inzwischen auch Microsoft) schlapp macht. Doch Ernüchterung macht sich breit, wenn unerwartete Performance-Probleme auftreten: Die Zahl gelangweilter Anwender, verärgerter Chefs und ratloser Administratoren steigt. Was tun? Tuning!

Von Bodo Michael Danitz

Performance Tuning heißt also das helfende Zauberwort, und weil es für große Datenbanken leider keinen Tuning Wizard mehr gibt, ist in diesem Fall tiefer gehendes Know-how, vor allem Erfahrung gefragt. Systematisch muss der „Flaschenhals" im System gefunden und eliminiert werden, und das in der Regel schweißtreibend flott, denn die Produktion soll laufen und die Geschäftsleitung, die gerade eher physisch als psychisch hinter einem steht, soll wieder zufrieden in ihrem Büro verschwinden.

Die Analyse

Was eigentlich ist „Performance"? Wie misst man sie, und wer misst sie? Stellt man diese Fragen der Berufsgruppe der Administratoren, dann erhält man viele in das Fachgebiet der Philosophie abgleitende Antworten, die da etwa beginnen mit „Nun, das hängt ganz davon ab …"

Nun, besinnen wir uns auf das, was wir gelernt haben: Dann wissen wir aus Erfahrung, dass es regelmäßig der User ist, der Performance bewertet – und meist am Telefon auch artikuliert. Sieht er nämlich die Eieruhr länger als etwa fünf Sekunden, gerät er bereits in einen wahrnehmbaren Zustand der Ungeduld. Für uns EDV-ler ist also die Antwortzeit des Systems ein Maß für Performance. Doch auch wenn ein „Job" innerhalb von fünf Sekunden antwortet, kann ein User nervös werden: Wenn der Job nämlich „nicht fertig" wird. Durchsatz wäre also auch noch gefragt.

Antwortzeit und Durchsatz liefern uns also ein Maß für die Performance des Systems. Und woraus besteht das System? Es beginnt bei der Clientanwendung und reicht bis zum Datenhalter im Back-End (den Benutzer als Performance bestimmendes Element vernachlässigen wir …):

- NavClient
- Client-Betriebssystem
- Client-Hardware
- Netzwerk
- Server-Hardware
- Server-Betriebsystem
- SQL Server

Der gesunde Menschenverstand rät: Am NavClient beziehungsweise an seiner Abfragestrategie wird keiner so schnell wirksam etwas ändern, ebenso wenig am Client-OS. Daher können wir uns Zeit raubende, akademische Betrachtungen dieser Elemente ersparen. Die Client-Hardware dürfte ursächlich ebenfalls vernachlässigbar sein (es sei denn, es finden sich unter ihnen noch 486er ...), und unterstellt man ein gut funktionierendes Netzwerk, dann landet man schon an dieser Stelle auf dem Server. Nehmen wir ferner an, dass es sich um einen dezidierten Datenbankserver handelt – dass also niemand auf die folgenschwere Idee gekommen ist, ihn beispielsweise zusätzlich mit Exchange auszulasten – dann dürften sich darauf also nur das Betriebssystem und die SQL Server-Installation sowie die wertvollen Unternehmensdaten aus Navision befinden.

Um herauszufinden, womit dieser Server seine Rechenzeit verbringt, bemühen wir den System Performance Monitor mit einer Sammlung der erfahrungsgemäß aufschlussreichsten Indikatoren:

- Processor (für jeden) – % Processor Time
- Physical Disk – Disk Read Bytes/sec, Disk Write Bytes/sec
- SQLServer:Access Methods – Full Scans/sec, Index Searches/sec, Probe Scans/sec, Range Scans/sec
- SQLServer:Buffer Manager – Readahead Pages/sec, Buffer cache hit ratio
- SQLServer:Locks – Average Wait Time (ms), Lock Waits/sec

Mit all diesen Indikatoren erhalten wir ein beeindruckend farbenfrohes Bild, das jedoch – systematisch betrachtet – gute Hinweise liefern kann.

Zum leichteren Verständnis folgt nun ein einfaches Interpretationsrezept:

Processor

Wir wünschen uns eine gleichmäßige Auslastung über alle vorhandenen CPUs, jeweils im Mittelwert nicht größer als 70-75 Prozent mit hoher Dynamik (das heißt, dass die Spitzenwerte die 100-Prozent-Marke durchaus kurzzeitig erreichen, dann aber wieder auf Werte deutlich unterhalb der 50-Prozent-Marke abfallen). Ein anderes, höher ausgelastetes Verhalten deutet auf eine Unterdimensionierung der CPUs (Anzahl, Taktung) oder auf eine unerwünschte Grundlast hin.

Physical Disk

(zur Aktivierung gegebenenfalls diskperf -y ausführen!)
Die Durchsatzgrenzen des Subsystems kann man sich ausrechnen, indem man die Bandbreiten der verwendeten Controller und Festplatten sowie des implementierten RAID-Levels bewertet. Die maximal mögliche Kapazität sollte auch hier höchstens zu etwa 70-75 Prozent im Durchschnitt erreicht werden.

SQL Server: Access Methods

Von allen Zugriffsmethoden sollte „Full Scans“ den deutlich geringsten Anteil haben und „Index Searches“ den höchsten. Zwischen diesen Eckwerten sollten sich die Anzahlen für Probe- und Range Scans befinden. Ist dies nicht der Fall, ist beispielsweise das Verhältnis Full Scans zu Index Scans größer als eins, dann dürfen wir davon ausgehen, dass eine suboptimale Indizierung der Tabellen oder eine ungeschickte Nutzung der vorhandenen Indizes eine Ursache darstellt.

SQLServer: Buffer Manager

Wir möchten möglichst selten, und wenn, dann nur kurzzeitigen Readahead (ungefähr < 5 sec) sowie möglichst viel Cache Hit Ratio (> 95%) sehen.

Häufiger Readahead kann auf ungeschickte Indizierung der Tabellen oder auf zu klein dimensionierten Hauptspeicher hindeuten, insbesondere, wenn die Cache-Trefferrate unter 95 Prozent sinkt. Lang andauernder Readahed bei relativ geringem Plattendurchsatz deutet auf ein langsames Festplatten-Subsystem hin.

SQLServer: Locks

Im konkurrierenden Zugriff vieler Benutzer sind Sperren unbedingt notwendig – Navision sperrt sogar gern auf dem höchsten Transaction Isolation Level „serializable“.

Entscheidend für die Performance ist nun, wie viele Client-Verbindungen wie lange auf ihre Sperranforderung warten müssen, weil die erforderlichen Daten gerade von anderen Benutzern im Zugriff sind. Wartezeiten von mehr als 1-2 Sekunden sind für die meisten Anwender an dieser Stelle nur schwer erträglich, so dass eine generelle Beschleunigung der Vorgänge, sei es durch geschicktere Indizierung, sei es durch leistungsfähigere Hardware, angezeigt ist.

Natürlich hat hier auch das Verhalten der Anwender Einfuss, denn je mehr Daten auf einmal gesperrt werden sollen – zum Beispiel Lesesperren bei Auswertungen oder Schreibsperren bei Rechnungsstellung für einen Großkunden – desto höher ist die Wahrscheinlichkeit, in den „Sperrkonflikt“ mit anderen Anwendern zu geraten. Weil wir aber den Anwendern nicht vorschreiben wollen, wann sie wie zu arbeiten haben, wollen wir uns auf technische Lösungen konzentrieren.

Der Autor

Bodo Michael Danitz beschäftigt sich seit fast zwei Jahrzehnten mit Datenbankdesign und Anwendungsentwicklung auf Oracle sowie mit der Konzeptionierung, Realisierung und Optimierung großer SQL Server-Architekturen. Innerhalb der PTecS GmbH, die sich auf Server-based Computing, Software- und Server-Management sowie Strategieberatung spezialisiert hat, betreut er federführend den Bereich „SQL Server-Systeme“ – schwerpunktmäßig mit der Performance-Optimierung solcher Systeme.

Die Kunst ist es nun, die so erhaltenen Informationen unter Berücksichtigung der grundsätzlichen Leistungsfähigkeit der verwendeten Hardware richtig zu interpretieren. Wir stellen fest, dass das Performance-Problem in der Abarbeitung der Abfragen und Transaktionen zu liegen scheint. Wir bemühen also weitere Werkzeuge – den SQL Server Profiler und den Query Analyzer.

Ein Licht geht auf!

Mit dem Profiler zeichnen wir eine Workload auf, um die Langläufer heraus zu fischen. Jeden dieser mutmaßlichen Übeltäter sehen wir uns im Query Analyzer genauer an, und was stellen wir fest? Genau! Die Abfragepläne könnten schöner aussehen. Es zeigen sich aufwändige Table Scans, wo bereits ein einfacher Index auf den Filterausdruck der Where-Clause deutlich Speed brächte, da werden Indizes ge-joined, wo ein Covering Index dies überflüssig macht, und in roter Farbe sticht ins Auge, dass angeblich Statistiken fehlen würden… Volltreffer!

Wenn man nun inne hält und in sich geht – wofür in der akuten Situation natürlich keine Zeit ist –, wenn man sich den Werdegang der Datenbank einmal vor Augen führt, dann wundert es auch nicht übermäßig: Da wurden die Daten vom Nativsystem migriert, indem die Objekte samt Indizes erzeugt wurden, um abschließend mit den Daten der letzten Jahre gefüllt zu werden, gefolgt von der Erfolgsmeldung „fertig“.

Der erfahrene SQL Server-Profi sowie der aufmerksame Leser ahnt es bereits: Es ist aber noch lange nicht „fertig“. Es fehlen mit an Sicherheit grenzender Wahrscheinlichkeit die Statistiken, auch Optimizer-Pages genannt, die dem Abfrageprozessor des SQL Servers den kostengünstigsten Weg zur Abarbeitung einer Abfrage zeigen.

Wie erzeugt man nun alle notwendigen Statistiken? Ganz einfach: Man führe die von Microsoft implementierte System Stored Procedure mit dem Namen „sp_createstats“ auf der Navision-Datenbank aus. In der Regel genügt es, Statistiken nur für indizierte Spalten zu generieren, und das erreicht man mit dem Parameter „Indexonly“:

```
sp_createstats ‚Indexonly'
```

Allein diese Maßnahme wirkt schon Wunder.

Jetzt widmen wir uns den mit dem Profiler ermittelten Langläufern, sofern sie nicht bereits Heilung durch die aktualisierten Statistiken erfahren haben.

Wir sorgen dafür, dass die Abfragen bezüglich ihrer Where- und Order-By-Clause Index-unterstützt werden. Sollen breite Tabellen (viele Spalten) abgefragt werden, und werden nur wenige Spalten selektiert, dann überlegen wir uns einen Covering-Index.

Ein Covering-Index beinhaltet Spalten der Where-Clause und der Order-By-Clause, ergänzt um die noch fehlenden Spalten der Select-Liste, so dass der Abfrageprozessor sämtliche referenzierten Felder aus dem Index beantworten kann und er nicht mehr per Bookmark-Lookup auf die Tabelle zugreifen muss – das spart wieder Zeit.

Jeder Gewinn ist ein Verlust

Was jedoch an Abfrageleistung gewonnen wird, geht wieder an Schreibleistung verloren, denn mit jedem Index bürdet man dem Server zusätzlichen Pflegeaufwand auf. Den kann man ihm aber teilweise erleichtern, indem man die Indizes mit einem angemessenen Füllfaktor erstellt. Er sorgt dafür, dass auf den Indexseiten noch Platz für künftige Eintragungen bleibt, so dass möglichst keine neue Seite eingefügt werden muss, was mit aufwändiger Baumberechnung verbunden wäre.

Übrigens: Ein Füllfaktor auf einem Index mit fortlaufenden Spaltenwerten (etwa eine Identity-Spalte) ist ziemlich sinnlos, im Gegenteil, er ist schädlich!

Es geht also darum, den goldenen Mittelweg zu finden – und der befindet sich genau dort, wo eine für das jeweilige Unternehmen repräsentative Workload in kürzester Zeit abgearbeitet werden kann. Man optimiere also die Gesamtlast, weniger einzelne Abfragen, um sich nicht zu verzetteln. Und diese repräsentative Last will zuerst gefunden werden!

Die richtige Indizierung zu finden, erfordert deshalb eine andauernde Beobachtung des Systems sowie tief gehende Kenntnis des SQL Server-Abfrageprozessors.

Das Licht am Ende des Tunnels…

Nehmen wir an, wir hätten sie gefunden und auch implementiert, die optimale Indizierung, dann werden wir nach einiger Zeit feststellen, dass die User schon wieder anrufen, weil das System ganz plötzlich, scheinbar über Nacht, langsam geworden ist.

Wahrscheinlichste Ursache: Die Indexseiten sind voll. Abhilfe: Re-Indizierung unter Berücksichtigung der Füllfaktoren. Zweit-wahrscheinlichste Ursache: Die Statistiken sind veraltet. Abhilfe: Statistiken aktualisieren.

Somit hätten wir einen schönen Wochenend-Wartungsjob gefunden. Und wie machen wir das? Microsoft scheint mit „dbcc (reindex)“ und den Datenbankoptionen „auto create statistics“ beziehungsweise „auto update statistics“ Lösungen für diese Aufgaben anzubieten – jedoch zeigt die Praxis, dass diese nicht

hinreichend zufrieden stellen: Tests haben ergeben, dass die Abfragepläne häufig suboptimal bleiben. Löscht man aber alle Indizes, erzeugt sie anschließend neu, und lässt man dann per „sp_updatestats" sämtliche Statistiken aktualisieren, dann erhält man wieder die optimalen Abfragepläne von einst – warum das so ist, wissen nur die Entwickler in Redmond.

Ein Beispiel:
Wir generieren deshalb mit dem Enterprise Manager ein Skript zur Erstellung aller Indizes und Primary Key beziehungsweise Unique Constraints. Dabei achten wir peinlichst darauf, dass der Gruppierte Index (Clustered Index) zuerst erstellt wird. Dann erstellen wir ein zweites Skript, das sämtliche Indizes oder Indexbehaftete Constraints löscht – und zwar den Clustered Index zuletzt!

Auf einer Testmaschine prüfen wir die Laufzeit der Skripten, und wir stellen fest, dass es etwa 12 Stunden für eine 50-Gigabyte-Datenbank benötigt ... – definitiv ein Wochenendjob! Im Anschluss daran aktualisieren wir noch zwei Stunden lang die Statistiken und stellen fest, dass unser verfügbares Wartungsfenster zu klein ist. Was nun?

Die Tuning-Eskalation

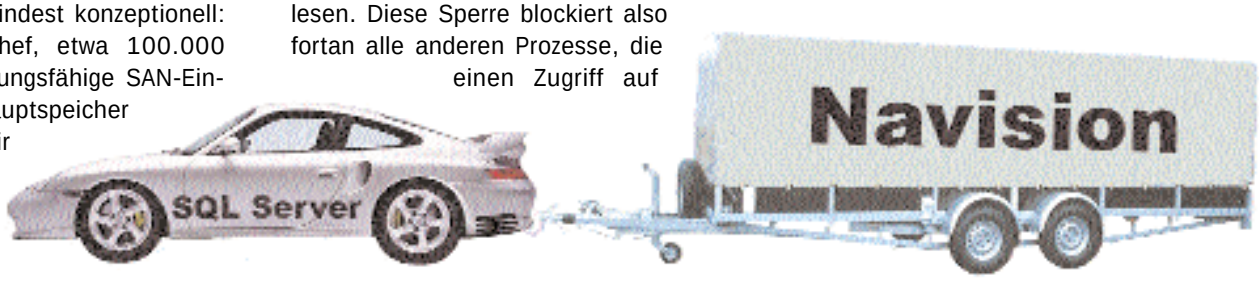
Ganz einfach, zumindest konzeptionell: Wir bitten den Chef, etwa 100.000 Euro für eine leistungsfähige SAN-Einheit und mehr Hauptspeicher zu investieren. Wir wollen mit AWE bis zu acht Gigabyte RAM nutzen (Windows 2000 Advanced Server und SQL Server 2000). Wir überlegen uns ferner eine sinnvolle Plattenaufteilung, indem wir Sequential I/O und Random I/O physikalisch voneinander trennen: Eigene Partitionen für Transaktionslog und Tempdb. Große Objekte, in der Regel betrifft dies 3-4 Tabellen, erhalten eigene Filegroups auf eigenen Partitionen, die gesammelten Non-Clustered-Indizes ebenfalls. Wir teilen unser Skript in mehrere parallele Jobs auf, die wir gegebenenfalls auch über mehrere Wartungsfenster verteilen können, und erhalten akzeptable Skript-Laufzeiten.

Im Produktionsbetrieb stellen wir fest, dass die Schreib-/Lesezeiten deutlich kleiner geworden sind, was wiederum zufriedenere User zur Konsequenz hat.

Bei dem Stichwort „Eskalation" sollte dringend noch ein Aspekt diskutiert werden, den kaum jemand kennt: Lock-Escalation – ein Mechanismus, der die Sperrenverwaltung von SQL Server beschleunigt. Jedoch: Bei Schreibvorgängen eskaliert SQL Server seine Satzsperren gerne zu einer exklusiven Tabellensperre, wenn nur eine hinreichende Anzahl Satzsperren erreicht ist. Die Eskalationsschwelle war bis zur Version 6.5 noch konfigurierbar, seit der Version 7.0 ist sie „hart verdrahtet".

... der herannahende Zug

Folgende konkrete Beobachtung kann gemacht werden: Ein Update-Statement, welches mehr als 764 Datensätze in einer Tabelle mit mehreren Millionen Sätzen verändern will, verursacht unter Umständen eine exklusive Tabellensperre – und das für die restliche Dauer der Transaktion, in der dieses Update durchgeführt wird. Andere Benutzer haben während dieser Zeit keine Chance, die Daten auch nur zu lesen. Diese Sperre blockiert also fortan alle anderen Prozesse, die einen Zugriff auf diese Tabelle wünschen, und das wird den einen oder anderen User schon wieder veranlassen, zum Telefon zu greifen ...



Und was macht man da als Admin? Die Prüfung mehrer Möglichkeiten hat die nachstehende als geeignetste identifiziert: Man identifiziere solche Objekte (Tabellen), erzeuge nach Möglichkeit einen Dummy-Datensatz und halte diesen in ständigem Lesezugriff durch einen SQL Server Agent Job – es muss schon eine andere Verbindung als die der ändernden Transaktion sein. Dann nämlich kann kein Prozess mehr eine exklusive Tabellensperre erhalten und setzt stattdessen brav seine Satzsperren – ausgetrickst!

Konkreter Jobtext:

```
begin transaction
  select * from tabelle
    where SatzNr = DummyID (holdlock)
```

(die Where-Clause soll also auf den Dummy-Datensatz zeigen)

Diese Lese-Transaktion bleibt offen, um die Sperre dauerhaft aufrecht zu erhalten.

Es wäre natürlich schöner, wenn sich die Entwickler bei Microsoft für solche Fälle eine andere Lösung überlegen könnten, vielleicht uns die Möglichkeit der konfigurierbaren Lock-Escalation zurückgäben, aber dieser Work-Around tut's zunächst auch. Wir warten auch schon ganz gespannt auf Yukon ...

Der Porsche mit Anhänger fährt!

Wer all diese Dinge in Erwägung gezogen hat, quasi die Anhängerkupplung seines Porsches auf die richtige Höhe gekurbelt und die richtigen Reifen aufgezogen hat und dann noch mit richtigem Drehmoment anfährt, rechtzeitig schaltet und regelmäßig Ölwechsel durchführt, der kann auch seinen Anhänger auf Touren bringen – schauen Sie doch mal in unserer SQL Server Tuning-Werkstatt vorbei – auf der CeBIT in Halle 4 / A38, MS-Partner Stand 64!

Für weitere Information siehe auch www.ptecs.de und www.sql-server.de.