Microsoft Technical Summit 2008

SQL Server
Locking, Blocking, Deadlocks

Siegfried Spuddig
consult Spuddig
info@spuddig.de

Einleitungstext

- Performance-Tuning von datenbankgestützten Applikationen ist mehr als nur Hardware- und Index-Tuning.
  Spätestens bei längeren Transaktionen müssen sich Entwickler und Administratoren zusammensetzen, um die Serialisierung der Abfragen zu vermeiden.
  Neben der Optimierung des Datenmodells für Mulit-User-Systeme können Sperrenmechanismen auf Query-, Datenbank- und Tabellen-Ebene eingesetzt werden.
  Serialisierung kann aber auch ausdrücklich gewünscht sein, um sicherzustellen, daß bestimmte Abläufe sich nicht überschneiden.
  Um die technischen Möglichkeiten mit SQL Server 2000, 2005 und 2008 und ihre Vor- und Nachteile geht es in diesem Vortrag.

Basics

Transaktionen

- Prozesse
  - Überweisung
  - Bankintern? Bankenübergreifend?
  - Workflows sind mehrere Einzeltransaktionen
- Technische Umsetzung
  - z.B. als T-SQL-Transaction, in der mehrere SQL-Statements zusammengefasst werden
  - Sperren auf Objekte & Daten durch den SQL-Server
  - Kompromiss zwischen Konsistenz und Multi-User-Betrieb

Kategorien von Transaktionen

- Normale Transaktionen
  - BEGIN TRAN MyTrans1
    SELECT...UPDATE... COMMIT TRAN MyTrans1
  - Nested Transactions, Distributed Transactions
- Einzelbefehle lösen auch Sperren aus
- Application Locks
  - Serialisierung marke Eigenbau / sp_getapplock
  - Serialisierung durch die Wahl des Datenmodells

Transaktionen → Sperren (=Locks)

- Sperren auf Zeilen
- Sperren auf Pages (8K Daten-Blöcke)
  - vereinfachte Verwaltung
- Sperren auf Tabellen (Tablock...)
  - noch einfacher / weniger Overhead
  - nötig bei CREATE PROC, Recompiles etc.
- Sperren auf Indices
  - (das sind ja im Grunde auch nur Tabellen)
- Sperren auf Datenbanken

Sperren-Typen (Locks)

- je nach Statement
- je nach Isolation-Level (=Kompromiss-Level)
- Intend Locks (IU, IX, ...) – Ausweitung
- eXclusive Lock (X)
- Shared Lock (S) – parallele Lesezugriffe
- Sch-S und Sch-M – Schema-Locks (Recompile)

Sperren kosten...

- Wartezeit bis zur Resourcen-Freigabe
- Zeit für die Verwaltung (=Overhead)
- Nerven

Quick Wins I

- Tuning verbessern
  - schlanke Prozesse
  - INDEX-Tuning & TABELLEN-Design
  - Stored Procedures reduzieren ggf. Roundtrips
  - Hardware / Hardware-Konfig / Scale-Out
- Eigentore vermeiden
  - Warten auf User-Input bei offenen Transaktionen
  - WebServices-Request in Transaktionen
- In Workflows denken - Transaktionen zerlegen

Öffnen einer SQL-Transaktion

- T-SQL:
  - BEGIN TRAN MyTran1
  - SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    BEGIN TRANSACTION MyTran1
- .NET:
  - conn.BeginTransaction();
  - conn.BeginTransaction(System.Data.IsolationLevel .Serializable);

Isolation Level & Kompromisse

- Isolation Level bis SQL 2000
  - Read Commited = default
  - Serializable (immer nur einer)
  - Repeatable Read (teuer)
  - Read Uncommitted (dirty read, immer möglich)
- Isolation Level SQL 2005 & 2008
  - Snapshot = Row Level Versioning
    → hervorragende READ-Performance
    → Wollen Sie das?

Probleme von Isolation-Levels

- Dirty Read
  - with (nolock)
  - Kraut und Rüben, keinerlei Lese-Sperren
- Non-repeatable read
  - zweiter Lesevorgang liefert veränderte Daten
- Phantom read
  - Daten tauchen „plötzlich“ auf

Problem-Tabelle

aus „SQL Server 2005 Administrator‘s Companion“

Table 17-2 Isolation Level Behaviors

Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read
Read uncommitted	Yes	Yes	Yes
Read committed without snapshot	No	Yes	Yes
Read committed with snapshot (statement level)	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot (transaction level)	No	No	No
Serializable	No	No	No

ISOLATION LEVEL auf Statement-Ebene: WITH ()

- SELECT ... FROM ... WITH (NOLOCK)
  - DIRTY READS
  - WHERE STATUS = ‚READ_ONLY‘
  - Vorsicht: Page-Splits können trotzdem zu fehlenden Daten führen!
- SELECT ... FROM ... WITH (NOLOCK)
  JOIN ... WITH (SERIALIZABLE)
  JOIN ... - - default = read committed
- WITH (HOLDLOCK, TABLOCKX, ...)

Sperren - Analyse

- Ask your admin!
- SPID = ID einer SQL_Connection,
  @@SPID = SPID der eigenen Conn.
- SQL Server 200x liefert detaillierte Informationen über Sperren:
  - select * from syslockinfo (SQL 2000)
    where spid = @@SPID
  - select * from sys.dm_tran_locks (ab 2005)
    where request_session_id=@@SPID

Werkzeuge zur Sperren-Analyse

- SQL PROFILER (Teil der SQL-Client-Installation)
- perfmon.exe (Windows)
  - LOCKs, DEADLOCKS / SEC...
- Management Views:
  - SELECT ... FROM ... SYS.DM_...
- Application_Process_Runtime_Tracing
  - An welcher Stelle gibt es Probleme?
  - Welche Einzelprozesse benötigen viel Zeit?

SQL Server 2005 Analyse-Features

- select * from sys.dm_os_waiting_tasks
  - Wer wartet? Wer blockt?
  - Monitoring: where wait_duration_ms > 2000
- select * from sys.dm_tran_locks
  - auf welche Sperren wird gewartet:
    select ... sys.dm_tran_locks as t1
    join sys.dm_tran_locks as t2
    ... (Query Tuning and Optimization S.345)

SQL 2005 Performance-Features

- Snapshot Isolation
  - row versioning
  - ALTER DATABASE MyDB SET READ_COMMITTED_SNAPSHOT ON
  - ALTER DATABASE MyDB SET ALLOW_SNAPSHOT_ISOLATION ON
- Indirekte concurrency features:
  - verbesserter Index Tuning Advisor
  - Online Index-Wartung – Deadlocks möglich!
  - Database Snapshot für Reports anlegen

Snapshot Isolation

Demo

- Aha-Effekt
- Einfache Analyse

Nachteile von Snapshot-Isolation

- ROW VERSIONING erzeugt Schreiblast – auch wenn kein Lesezugriff die Daten braucht
- TEMPDB bekommt mehr Last
  - Memory? Disk-Konfigs?
- Mehr Overhead
- Keine Unterstützung von Distributed Queries – aber die sollte man bei Sperren-Problemen sowieso vermeiden!

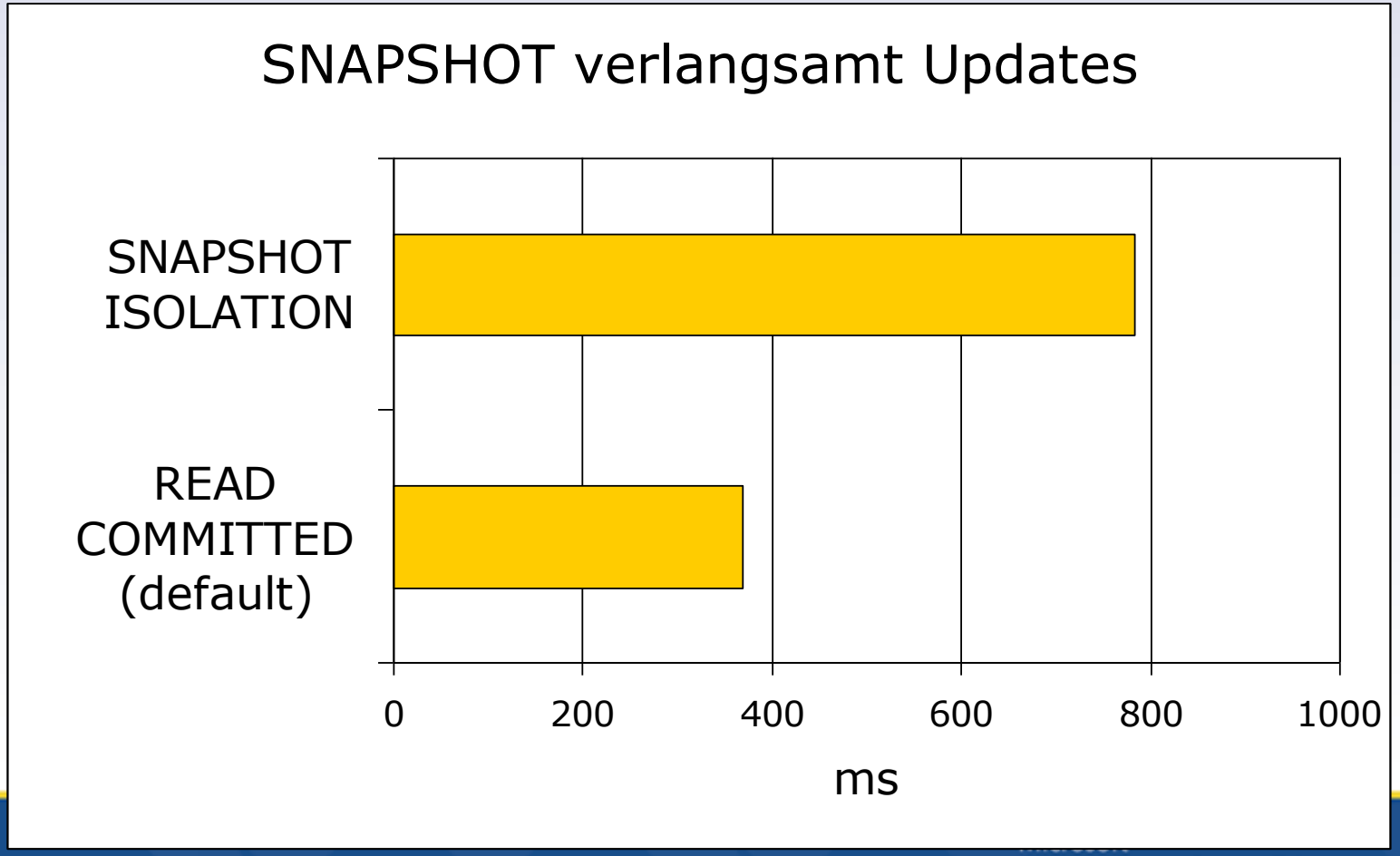
Snapshot Write-Performance? Es kommt darauf an...

- Update-Verhalten der Applikation?
- Transaktions-Länge?
- Hardware? DRAM? TempDB Storage?

- Schreibvorgänge profitieren auch von effizienteren Lesevorgängen
  - geringe I/O-Belastung
  - geringe CPU-Belastung
  - weniger Sperren durch die Lese-Vorgänge

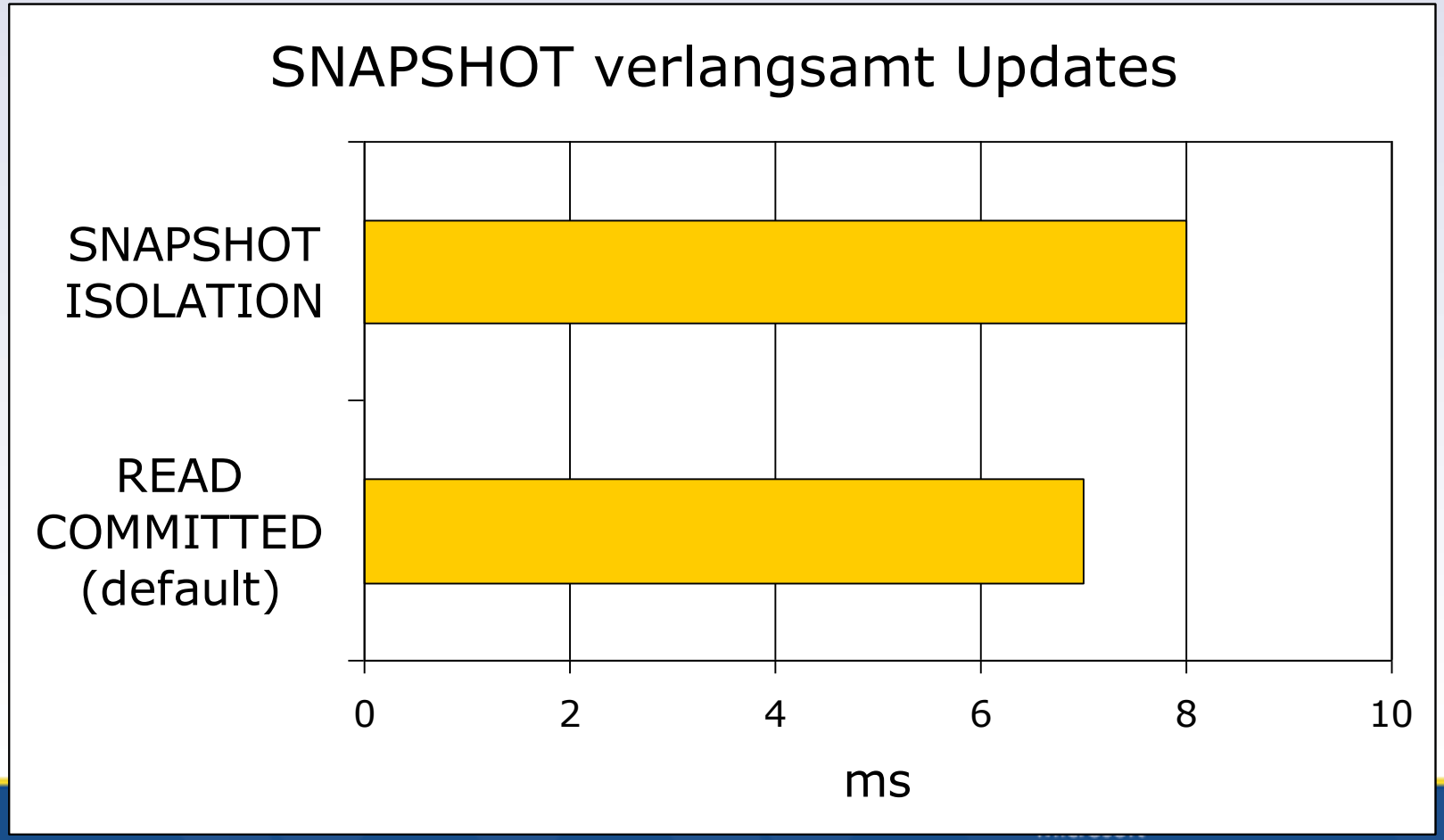
Snapshot Write-Performance I

WORST CASE SZENARIO: HEAP, Update, 100.000 Zeilen



Snapshot Write-Performance II

Update 1000 Rows auf einen Clustered Index



SNAPSHOT PERFORMANCE?

MESSEN, MESSEN, MESSEN

Zusammenfassung Snapshot Isolation

Specials

Lock Escalation

- SQL Server arbeitet mit Row-Level-Locking
  - Sperrenverwaltung erzeugt Overhead
  - Lock Escalation reduziert die Anzahl der Sperren
  - SQL 2008: Disablen der LOCK-Escalation ist auf Tabellen-Ebene möglich, sonst KB-Trick
    - select * from ... with (updlock, holdlock) where 1=0
  - DBCC TRACE 1211: Lock Escalation auf Instanz-Ebene ausschalten

SQL 2008 Features

- Lock Escalation Option
  - Abschalten mit ALTER TABLE möglich
- Filtered Indexes
  - für zusätzliche schlanke Indices
  - entspricht einem Index mit where-clause:

```
CREATE NONCLUSTERED INDEX MeinIndex
ON MyTable (Spalte_1, Spalte2)
WHERE Spalte2 IS NOT NULL;
```

Deadlocks

Deadlocks

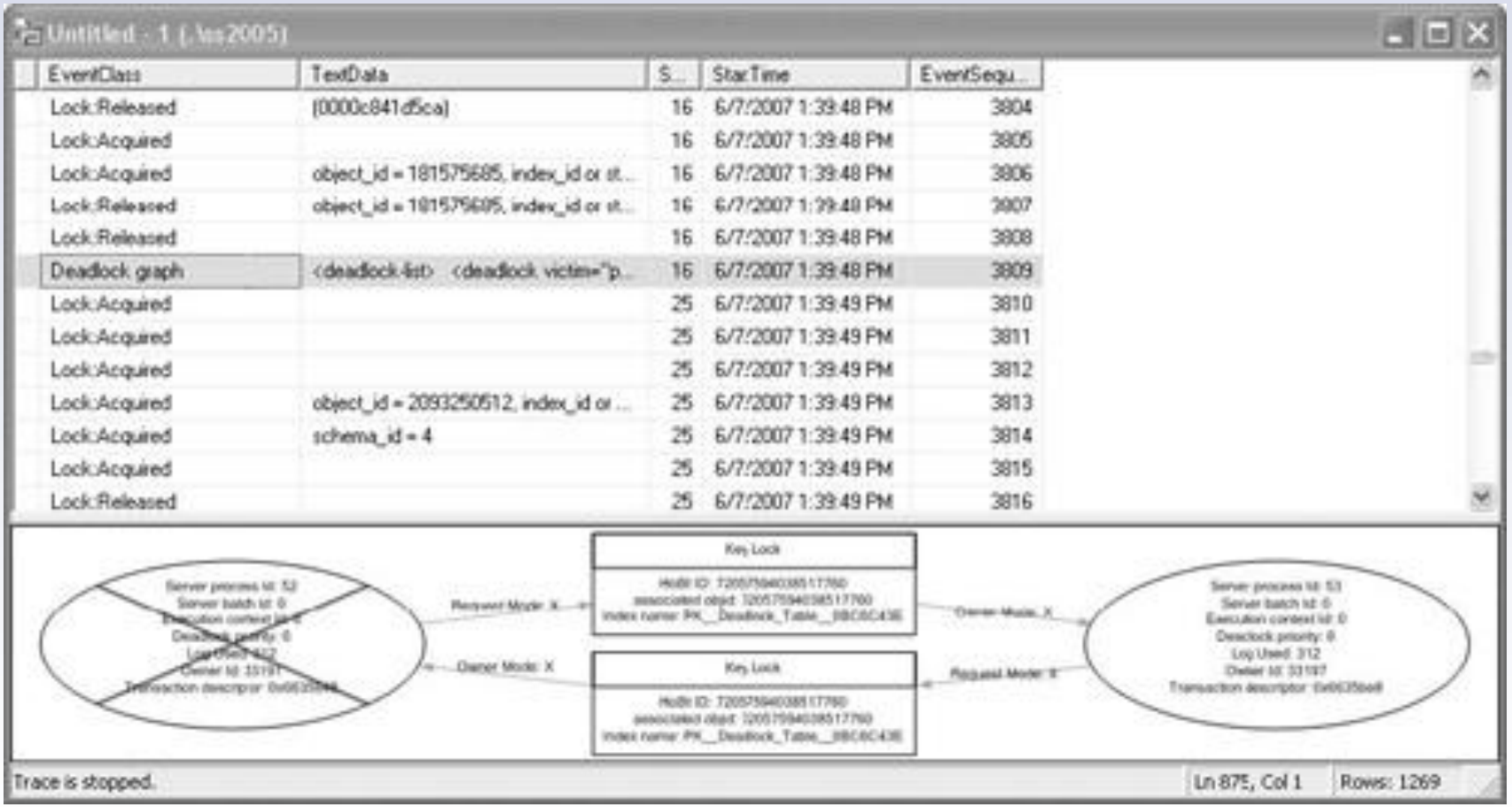
- „konventioneller“ DeadLock
  - zwei Prozesse, zwei Resourcen, unterschiedliche Reihenfolge der Allokation
- „conversion“ Deadlock
  - Shared Locks sind zunächst kompatibel
  - Shared Locks werden zu Exclusive Locks erweitert
- DEADLOCK VICTIM: Automatischer KILL & ROLLBACK
- ROLLBACK dauert z.T. sehr lange
  - Query läuft als parallelisierte Abfrage
  - Rollback läuft nie parallel -> relativ hohe Laufzeit

Deadlocks – Profiler & Traces

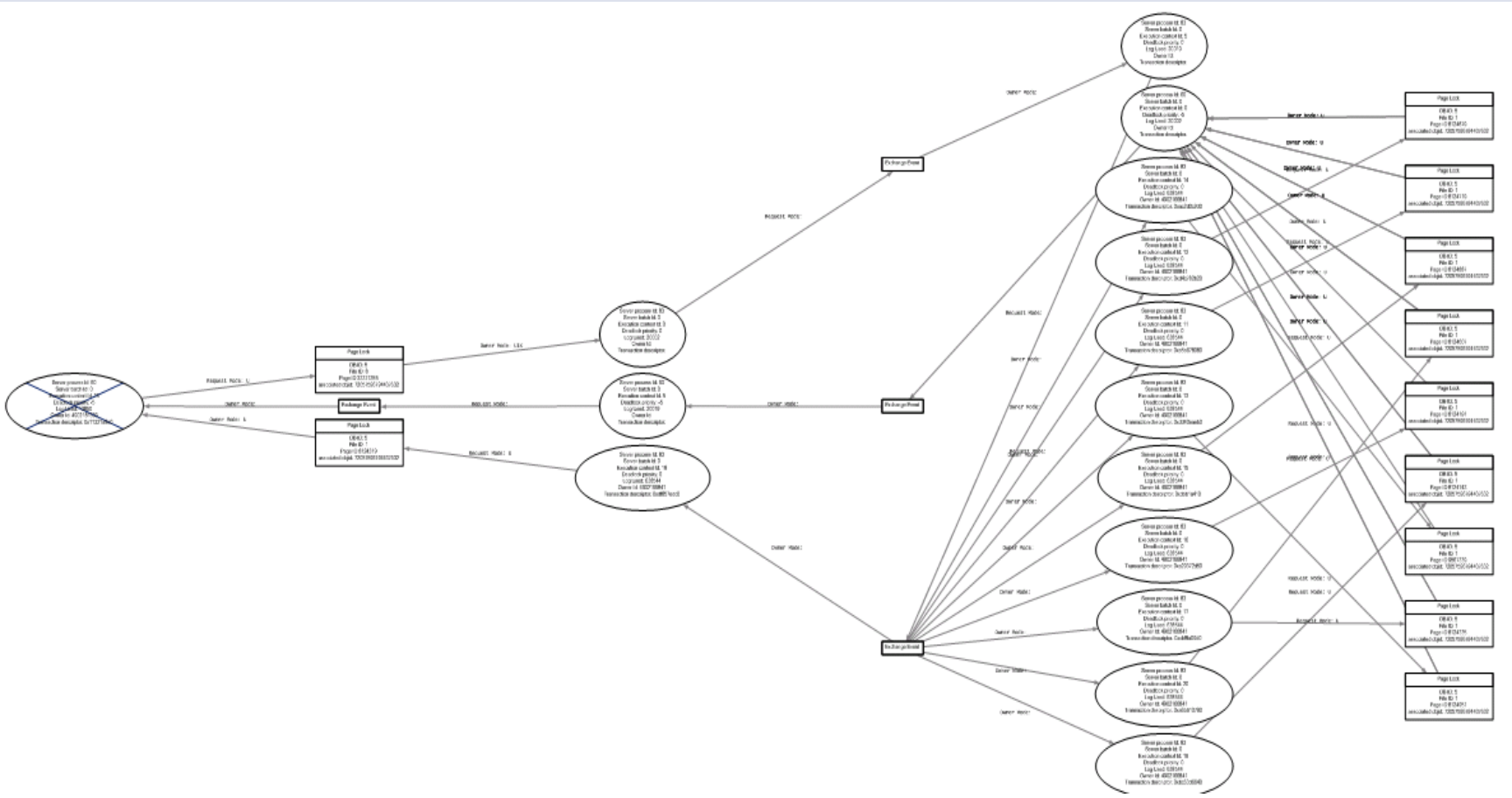
Lock:Deadlock	8:33371355
Deadlock graph	<deadlock-list> <deadlock victim="processfb7978"> <proce...
Lock:Deadlock Chain	Parallel query worker thread was involved in a deadlock
Lock:Deadlock Chain	Deadlock Chain SPID = 68 1:8124335 ...
Lock:Deadlock Chain	Deadlock Chain SPID = 118 1:8124335 ...
Lock:Deadlock Chain	Parallel query worker thread was involved in a deadlock
Lock:Deadlock Chain	Parallel query worker thread was involved in a deadlock
Lock:Deadlock Chain	Parallel query worker thread was involved in a deadlock
Lock:Deadlock Chain	Deadlock Chain SPID = 68 1:8124743 ...
Lock:Deadlock Chain	Deadlock Chain SPID = 118 1:8124743 ...
Lock:Deadlock Chain	Parallel query worker thread was involved in a deadlock

- DBCC TRACE 1204, 1205, 1206
  - http://www.sql-server-performance.com/rd_traceflags.asp

Deadlocks Analyse: Profiler SQL 2005



Deadlocks IV



Quick Wins II

- SET DEADLOCK PRIORITY LOW
  - SQL 2000: NORMAL oder LOW
  - SQL 2005+2008: -9,-8,...,8,9
- TRY CATCH – bis zu 25 Iterationen...
- Objektreihenfolge in Queries (...mmmh...)
- Index Tuning, zusätzliche Indices
- #TMP_TABLE statt TMP_TABLE
- Application_Process_Runtime_Tracing

Zum Schluß...

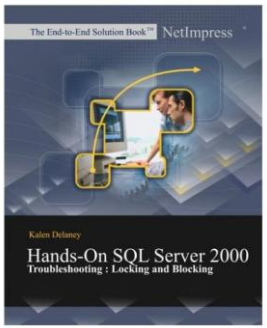
Wofür in 75 Minuten einfach keine Zeit blieb...

- SET XACT_ABORT ON / OFF
- SET LOCK_TIMEOUT 500 (ms)
- WAIT_STATS -> White Paper
- Ask your Admin: TABLE-PARTITIONING
- System.Transaction
- GUIDs als CLUSTERED_INDEX?
- Hardware: Server mit ½ TByte RAM (DL785)

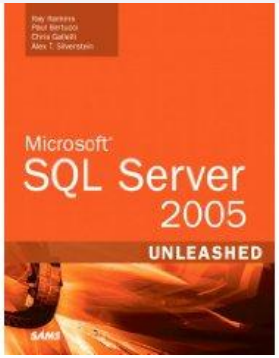
Quellenangaben

Quellenangaben I

- Kalen Delaney „Hands-On SQL Server 2000 : Troubleshooting Locking and Blocking [DOWNLOAD: PDF] “ eBook, $22.95

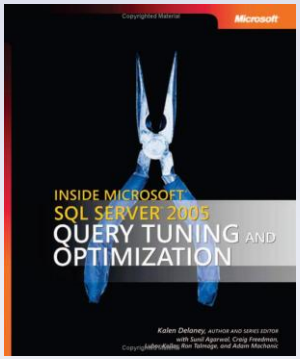


- Kalen Delaney „Inside Microsoft SQL Server 2000“ inkl. eBook PDF
- **Sams „SQL Server 2005 unleashed“ inkl. eBook (PDF) auf CD (Spitze!)**



Quellenangaben II

- MSPress Inside SQL Server 2005: Query Tuning and Optimization (kein eBook)
- Kalen Delaney and Fernando Guerrero: Database Concurrency and Row Level Versioning in SQL Server 2005 http://www.microsoft.com/technet/prodtechnol/sql/2005/cncrrncy.mspx
- MSPRESS Glenn Johnson „ADO.NET 2.0 Advanced Topics“



Quellenangaben III

- [ADMIN] MSPress „SQL Server 2005 Administrators Companion“ inkl. eBook (PDF) auf CD
- [ADMIN] Addison Wesley „SQL Server 2005 Practical Troubleshooting – The Database Engine“ Ken Hendersen

Ihr Potenzial. Unser Antrieb.

Vielen Dank

... und programmieren Sie fleissig große Applikationen und Datenbanken ☺