Radványi Tibor
Eszterházy Károly Főiskola, Számítástudományi Tanszék
dream@aries.ektf.hu

AZ MSSQL SZERVER HATÉKONYSÁGÁNAK VIZSGÁLATA ADAT-INSERT SZEMPONTJÁBÓL

Bevezető

Napjaink kutatási területei között fontos helyet foglal el az adatbázis rendszerek vizsgálata, ezek hatékonyságának mérése. [1] A nagy adatforgalmú rendszereknél az adatok felvitele, azok beillesztése jelentős és kifejezetten erőforrás-igényes művelet. A benchmark mérésnél szem előtt kell tartani mind a szerver, mind a kliensoldali szoftverek lehetőségeit. [2][3] A mérési eredményeket befolyásoló tényezők, mint a Dataset mérete, az SQL parancsok összetettsége, és a hardver/szoftver környezet rögzítése az első feladat.[1] A kliens program a Microsoft Visual .Net rendszerében készült, C# nyelven. Az adatbázis elérés, az ADO.NET technológia használata hatékony eszköz az adatbázisok eléréséhez. Az optimális teljesítmény eléréséhez felhasználtuk a Microsoft ajánlásokat és különböző kutatások eredményeit. [2][5]

A mérés során használt hardver és szoftver rendszerek

Az egri Eszterházy Károly Főiskola Számítástudományi Tanszékén került felállításra egy, a méréshez elengedhetetlenül szükséges szerver számítógép. Ez a gép biztosítja annak a lehetőségét, hogy megfelelő szerveroldali teljesítményt nyújtson, mely nem távolodik el a valós felhasználási környezetek biztosította lehetőségektől.

Szerver *(dragon.ektf.hu)***:**
Processzorok típusa: 2 db Intel Pentium III Xeon
Memória: 1024 MB
HDD: 2 db SCSI vezérlésű, 30 Gb méretű, de nem Raidbe kapcsolt.
Operációs rendszer: Microsoft Windows 2003 szerver
Adatbázis szerver: Microsoft SQL Server Enterprise Edition
Verziószám: 8.00.760 (SP3)

Munkaállomások *(csoportos terhelés esetén)***:**
Processzor: Intel Pentium 4 (1600 MHz)
Memória: 256 MB
HDD: 1 db 40 Gb méretű IDE vezérlésű 7200 ford/perc
Operációs rendszer: Microsoft Windows XP professional SP1

Hálózat:
Belső hálózat: 100 Mbps, DHCP, DNS szolgáltatásokkal
Külső hálózat: 512 Kbps ADSL, a szolgáltató által biztosított DHCP és DNS szolgáltatásokkal

A programfejlesztés a Microsoft Visual Studio .NET 2003 részét képező C# nyelven történt. Az adatbázis egy Microsoft SQL Server-en található.

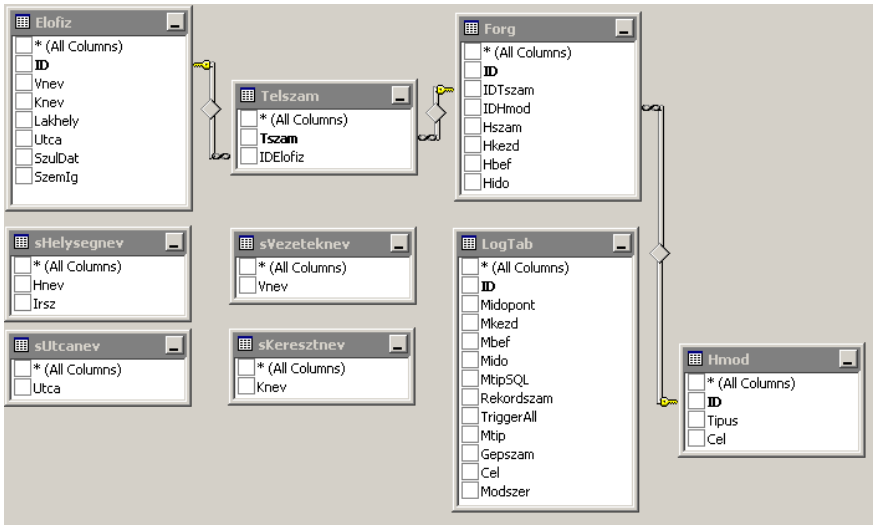
Az adatbázis

Segédtáblák

Az előfizetők adatait tartalmazó tábla véletlenszerű feltöltéséhez használt egyszerű táblák melyek alapadatokat tartalmaznak: (sHelysegnev, sKeresztnev, sKeresztnev, sUtcanev). Ezek nem játszanak jelentős szerepet a mérésben, mindössze a környezet megteremtésében van szerepük.

Táblák

A tesztekhez közvetve, illetve közvetlenül szükséges adattáblák.
Elofiz: a telefontársaság előfizetőinek adatait tárolja.
Telszam: az ügyfelekhez tartozó telefonszámok
Hmod: Hívás módja (vezetékes, mobil, belföld, …)
Forg: A mérések alapjául szolgáló forgalom tábla, mely a hívások adatait tárolja.



1. ábra: Triggerek

A forgalom táblához két trigger tartozik:

`forg_hmod`: Rekordbeillesztés után beállítja a hívás időpontját és a hívásmódot, az automatizálható adat-meghatározást érdemes kihasználni, hiszen ezzel a hálózati adatforgalmat tudjuk csökkenteni.
(Az 1. függelék ennek programját mutatja.)

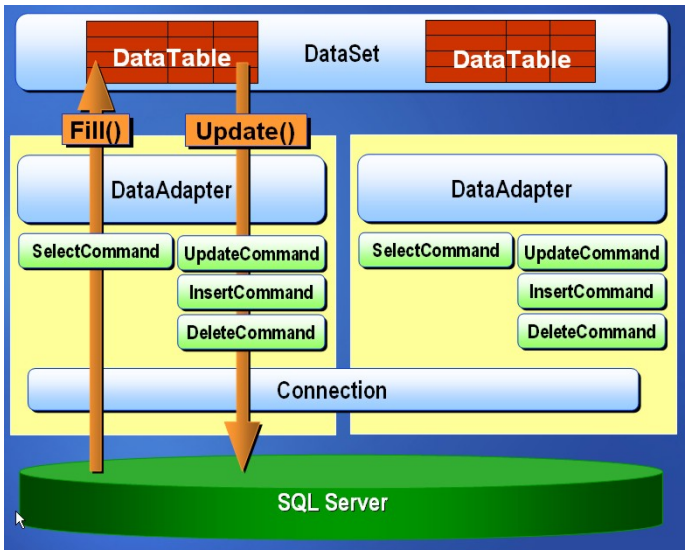
`forg_hbef`: a 'Hívás befejezése' mező kitöltésekor kiszámítja a hívás időtartamát, majd bejegyzi a rekord megfelelő oszlopába.
(A 2. függelék e feladat megoldását adó programot tartalmazza.)

A program

A kliensprogram technológiája a legújabb Microsoft fejlesztést, a Visual .Net rendszert használja. A szoftver C# nyelven íródott, mely rugalmas eszközt biztosít a megfelelő vizsgálatok elvégzéséhez.

Adatbázis-elérési módszerek

Mivel a vizsgálatunk a Microsoft MSSQL szerverének adat-insert partíciójára terjedt ki, ezért a lehetséges DataReader élő kapcsolattal rendelkező, de csak olvasásra alkalmas és a DataSet kapcsolat nélküli lehetőségek közül a DataSet megoldást választottuk. A DataSet osztály kommunikációját az SQL szerverrel jól szemlélteti a *2 ábra*.



2. ábra: Kommunikáció az SQL szerverrel

A program jelenlegi állapotában – a tesztek szempontjából – két különböző adatkezelési módszert használ, az egyik az ADO.NET keret által biztosított SqlDataSet osztály DataTable osztályának a Rows collection-jét bővíti az új rekordokkal, majd a bővítés befejezésekor az adott SqlDataAdapter osztály Update metódusával aktualizálja az adatbázis tartalmát.

A másik módszer pedig tárolt eljárások használatával teszi ugyanezt. Gyakorlatilag az adatbázissal történő kapcsolattartás mindkét esetben ADO.NET alapokon nyugszik, csak utóbbi esetben az adatfelvitelért az adatbázis szerveren lévő tárolt eljárások felelősek, melyeket paraméteresen az SqlCommand osztály segítségével hívunk meg.

Amikor tárolt eljárást használunk az adatok feltöltésére, akkor mindössze a SQLCommand osztály megfelelő paraméterezésére van szükség, és a parancs futtatására.

A két módszer (ADO; SP) vizsgálata volt a célunk, és ennek eredményeit jelenítjük most meg.

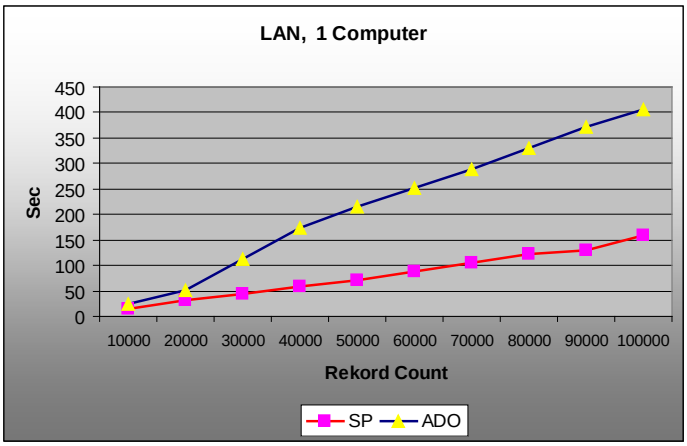
A mérések és az eredmények

A méréseknél szem előtt tartottuk, hogy a rendszer összetett felépítése miatt sok tényező befolyásolhatja az eredményeket. Ezért minden itt közölt mért eredmény minimálisan három, nagyobbrészt néhány tíz mérésnek az átlagából adódik. A különböző rekordszámok vizsgálatánál így összesen mintegy 800 mérést végeztünk. A mérési eredmények átlagolása előtt elemeztük az értékeket, és az egy-két alkalommal előforduló szélsőségesen kiugró eltéréseket mutató értékek nem kerültek be az átlagosba sem. Ezen eltéréseknek mindig valamilyen, a méréstől független oka volt (hardverhiba, a szerver nem tervezett terhelése). A szerveren a mérés idejére leállítottuk az egyébként erőforrás-igényes folyamatokat. Így nem futottak az egyéb SQL szerverek (Oracle, MySQL). Ezzel próbáltuk a legzavarmentesebb körülményeket biztosítani.

a. Helyi hálózaton belülről, egy kliensgép esetében az adatok:

Rcount	ADO	SP
10000	14,26565	24,2969
20000	30,9583	52,224
30000	44,401	113,5
40000	59,4896	174,3
50000	71,32825	216,016
60000	89,25	289,2373
70000	106,37	330,556914
80000	121,474	371,876529
90000	129,271	406,723

100000	159,161	289,2373



3. ábra: A két adatkezelési módszer összehasonlítása

A *3. ábrán* a kialakult görbék láthatóan jól leírhatók lineáris egyenlettel, ahol az y = mx + b alakú egyenletből, az m paraméter értékét vizsgáljuk meg egymáshoz viszonyítva. Az egyenlet meghatározását a legkisebb négyzetek módszerével végeztük, így illesztettük az egyenest a mért értékpárokra. Ebből a számításból kapott eredmények:

m_{SP} = 0,00150933

m_{ADO} = 0,00411515

Ahogy a grafikon is mutatja, a tárolt eljárás használata egyenletesebb, és sokkal kedvezőbb hatékonysági mutatóval rendelkezik:

$M = m_{ADO} / m_{SP}$ = 2,7265

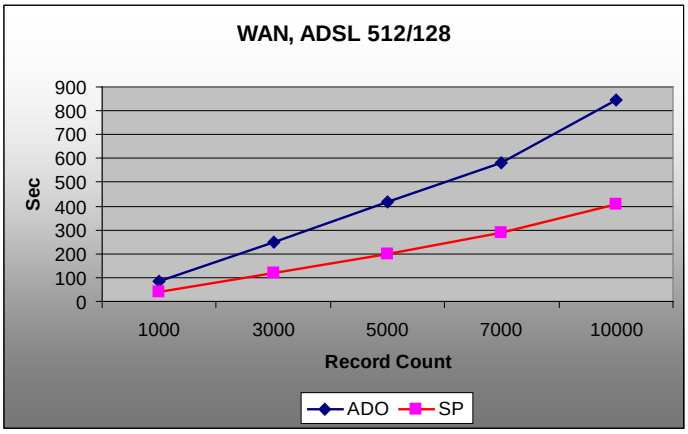
Ez mutatja, hogy közel háromszoros sebességet biztosít a tárolt eljárás használata ebben az esetben.

Egy fontos megjegyzés: ha a rekordok felvitelénél az Update metódust nem a teljes rekordcsoport memóriabeli létrehozása után alkalmazzuk, hanem minden egyes rekord után, ez a szám akár 100 szorosára is nőhet. Így ha több ezer rekordot viszünk fel, és nem feltétel a pillanatnyi aktualizálás, akkor a rekordok rögzítése után tegyük ezt meg, de mindenképpen nagyobb csoportonként.

WAN hálózatból, ADSL kapcsolat felhasználásával az adatok a következők:

Rcount	ADO	SP
1000	82,8625	39,9775

3000	248,073	120,266
5000	415,618	200,004
7000	583,255	286,318
10000	847,462	407,74



4. ábra: A két módszer összehasonlítása WAN hálózat esetén

A *4. ábrán* kialakult görbék ebben az esetben is jól leírhatók lineáris egyenlettel. A számítási műveletek elvégzése után a következő együtthatók adódnak:

$m_{SP} = 0{,}040665$

$m_{ADO} = 0{,}084036$

Ahogy a grafikon is mutatja, a tárolt eljárás használata egyenletesebb, és sokkal kedvezőbb hatékonysági mutatóval rendelkezik:

$M = m_{ADO} / m_{SP} = 2{,}06652$

A hatékonysági mutató csökkenését befolyásolja a hálózat eltérő sebessége, és stabilitása is.

Következtetések, továbblépési irányok

Az adatbázisok programozása, elérése felhasználói programokból napjainkban igen elerjedt, az élet különböző területein megjelenő, sok helyen vezető szerepet betöltő problémakör. Az adatok kezelésének első lépése, azok tárolása. Ez a művelet minden rendszerben megjelenik, helyenként jelentős erőforrásokat felemésztve a rendelkezésre álló keretekből. A célunk ezzel a vizsgálattal az volt, hogy egy, napjainkban széles körben használt rendszer esetén vizsgáljuk meg ezen terhelés csökkentésének lehetőségét.

A mérési eredmények egyértelműen alátámasztják, hogy a rendszer adatfelviteli hatékonysága nagy mértéken növelhető, ha kihasználjuk az SQL szerverek biztosí-

totta lehetőségeket, *a tárolt eljárások* használatát még látszólag más módszerrel is könnyedén megoldható feladatok esetén is.

Az insert művelet vizsgálatát ki fogjuk terjeszteni az Oracle, az IBM DB2, az Interbase SQL szerverek vizsgálatára is. E vizsgálatokat nem kizárólag a módszerek összevetésével fogjuk megtenni, hanem az így kapott eredményeket egymás mellé állítva keressük a fenti szerverek adat-insert hatékonyságának keresztmetszetét.

Ennek rugalmasabb, és könnyebb kezeléséhez a C# nyelven írt kliens program továbbfejlesztése is szükséges. Feladat lesz különböző osztályok létrehozása a különböző adatbázis-kezelőkhöz, továbbá módszerekhez. Minden osztálynak ugyanazon függvényeket kell tartalmaznia, így a főprogramban az egyre bonyolultabbá váló feltételek helyett egyszerűen a megfelelő osztály példányát kell használni.

További feladat az időzítési rendszer átalakítása olyan formára, hogy az egyes időzítéseket ne kelljen gépenként újra és újra beállítani, hanem csak időzítő módba kelljen kapcsolni azokat. Az aktuális időzítések az adatbázisban központilag jelennek meg, és az időzített programok folyamatosan vizsgálnák, hogy van-e végrehajtandó feladat kijelölve számukra. Ez jelentősen megkönnyíti majd a tesztelést, már kis számú gép esetén is, de a nagyszámú kliensek használatához nélkülözhetetlen.

1. függelék

```
CREATE TABLE [dbo].[Elofiz] (
    [ID] [int] IDENTITY (1, 1) NOT NULL ,
    [Vnev] [varchar] (25) COLLATE Hungarian_CI_AS NOT NULL ,
    [Knev] [varchar] (20) COLLATE Hungarian_CI_AS NOT NULL ,
    [Lakhely] [varchar] (25) COLLATE Hungarian_CI_AS NOT NULL ,
    [Utca] [varchar] (25) COLLATE Hungarian_CI_AS NOT NULL ,
    [SzulDat] [datetime] NOT NULL ,
    [SzemIg] [char] (8) COLLATE Hungarian_CI_AS NOT NULL
) ON [PRIMARY]

CREATE TABLE [dbo].[Forg] (
    [ID] [bigint] IDENTITY (1, 1) NOT NULL ,
    [IDTszam] [char] (12) COLLATE Hungarian_CI_AS NOT NULL ,
    [IDHmod] [int] NULL ,
    [Hszam] [char] (12) COLLATE Hungarian_CI_AS NOT NULL ,
    [Hkezd] [datetime] NULL ,
    [Hbef] [datetime] NULL ,
    [Hido] [int] NULL
) ON [PRIMARY]


CREATE TABLE [dbo].[Hmod] (
    [ID] [int] IDENTITY (1, 1) NOT NULL ,
    [Tipus] [varchar] (20) COLLATE Hungarian_CI_AS NOT NULL ,
    [Cel] [varchar] (20) COLLATE Hungarian_CI_AS NOT NULL
) ON [PRIMARY]


CREATE TABLE [dbo].[IdozitLogin] (
    [IDIdozites] [int] NOT NULL ,
    [Gepszam] [smallint] NOT NULL ,
    [Befejezte] [smallint] NOT NULL
) ON [PRIMARY]


CREATE TABLE [dbo].[Idozites] (
    [ID] [int] IDENTITY (1, 1) NOT NULL ,
    [Tipus] [int] NOT NULL ,
    [Gepszam] [smallint] NOT NULL ,
    [Kezdet] [datetime] NOT NULL ,
    [Befejezve] [datetime] NULL
) ON [PRIMARY]


CREATE TABLE [dbo].[IdozitesTipus] (
    [ID] [smallint] NOT NULL ,
    [Gepszam] [smallint] NOT NULL
) ON [PRIMARY]


CREATE TABLE [dbo].[LogFej_nincskesz] (
    [ID] [char] (10) COLLATE Hungarian_CI_AS NULL ,
    [Server] [char] (10) COLLATE Hungarian_CI_AS NULL ,
```

```
      [Oprend] [char] (10) COLLATE Hungarian_CI_AS NULL ,
      [NetSpeed] [char] (10) COLLATE Hungarian_CI_AS NULL ,
      [ServerProcNum] [char] (10) COLLATE Hungarian_CI_AS NULL ,
      [ProgNev] [char] (10) COLLATE Hungarian_CI_AS NULL
  ) ON [PRIMARY]


  CREATE TABLE [dbo].[LogTab] (
      [ID] [int] IDENTITY (1, 1) NOT NULL ,
      [Midopont] [datetime] NULL ,
      [Mkezd] [datetime] NULL ,
      [Mbef] [datetime] NULL ,
      [Mido] [float] NULL ,
      [MtipSQL] [char] (10) COLLATE Hungarian_CI_AS NULL ,
      [Rekordszam] [bigint] NULL ,
      [TriggerAll] [bit] NULL ,
      [Mtip] [char] (10) COLLATE Hungarian_CI_AS NULL ,
      [Gepszam] [smallint] NULL ,
      [Cel] [char] (10) COLLATE Hungarian_CI_AS NULL ,
      [Modszer] [char] (10) COLLATE Hungarian_CI_AS NULL
  ) ON [PRIMARY]


  CREATE TABLE [dbo].[Telszam] (
      [Tszam] [char] (12) COLLATE Hungarian_CI_AS NOT NULL ,
      [IDElofiz] [int] NOT NULL
  ) ON [PRIMARY]


  CREATE TABLE [dbo].[h] (
      [Col001] [varchar] (255) COLLATE Hungarian_CI_AS NULL
  ) ON [PRIMARY]


  CREATE TABLE [dbo].[sHelysegnev] (
      [Hnev] [varchar] (25) COLLATE Hungarian_CI_AS NOT NULL ,
      [Irsz] [char] (4) COLLATE Hungarian_CI_AS NULL
  ) ON [PRIMARY]


  CREATE TABLE [dbo].[sKeresztnev] (
      [Knev] [varchar] (20) COLLATE Hungarian_CI_AS NOT NULL
  ) ON [PRIMARY]


  CREATE TABLE [dbo].[sUtcanev] (
      [Utca] [varchar] (25) COLLATE Hungarian_CI_AS NOT NULL
  ) ON [PRIMARY]


  CREATE TABLE [dbo].[sVezeteknev] (
      [Vnev] [varchar] (25) COLLATE Hungarian_CI_AS NOT NULL
  ) ON [PRIMARY]
```

2. függelék

```
CREATE TRIGGER forg_hmod
ON dbo.Forg
FOR INSERT
AS
Declare @i int, @ins bigint, @htel char(12), @tipus char(20), @cel char(20), @s char(2)

select @ins="ID" from inserted                    --beilleszett forgalom-rekord azonosí-
tója
select @htel="hszam" from inserted          --hívott szám

select @s=(select substring(@htel,3,2))           -- körzetszám kinyerése

select @tipus=case                                -- hívástípus megállapítása (mobil,
vez., stb..)
 when @s='30' then 'm30'
 when @s='20' then 'm20'
 when @s='70' then 'm70'
 else 'v'
end

select @s=(select substring(@htel,1,2))           -- országhívószám kinyerése

select @cel=case                                  -- cél megállapítása (bel-, külföld)
 when @s='06' then 'belfold'
 else 'kulfold'
end


--if (select substring(@htel,3,2))='30' select @i="ID" from hmod where cel='A' and tipus='b'
--else if (select substring(@htel,3,2))='20' select @i="ID" from hmod where cel='A' and tipus='c'
select @i="ID" from hmod where cel=@cel and tipus=@tipus

BEGIN
--@i nulla lesz, ha nincs megfelelő rekord!
        update forg set IDhmod=@i where forg.ID=@ins -- in (SELECT "ID" from inserted)
        update forg set Hkezd=getdate() where forg.ID=@ins
END
CREATE TRIGGER Forg_hido
ON dbo.Forg
FOR UPDATE
AS
BEGIN
 if update(hbef)
 UPDATE forg SET hido=DATEDIFF(s,forg.hkezd,forg.hbef) WHERE forg.id in (SELECT "ID"
from inserted)
END
```

Hivatkozások

[1] A. Ailamaki, M. Shao: „DBMbench: Microbenchmarking Database Systems in a Small, Yet Real World” in Confidential – Submitted to ICDE 2004

[2] Microsoft: Improving .NET Application Performance and Scalability, (2004), 639–682

[3] Mike Ruthruff (Microsoft Co.): Microsoft SQL server 2000 Index Defragmentation Best Practices, (2003)
[4] J. Gray. „The Benchmark Handbook for Database and Transaction Processing Systems", Morgan Kaufman Publishers, Inc. 2nd edition, 1993
[5] Jim Gray: http://research.microsoft.com/~gray