Willkommen zum „IBM Informix Newsletter“

Inhaltsverzeichnis

Aktuelles

Liebe Leserinnen und Leser,

wie angekündigt beschäftigt sich diese Ausgabe des Newsletters fast ausschliesslich mit dem neuen Feature der Sharded Tables. Die Idee des Sharding ist eine konsequente Fortsetzung der „Scalable Architecture“. Informix kann auf fast allen Grössenklassen von Rechnern genutzt werden. Die beginnt beim Raspberry Pi und findet nach oben nur Grenzen in der Rechnerarchitektur. Diese obere Grenze wird nur mittels Sharding über Rechner hinweg durchbrochen. Die wesentlichen Features des Sharding sind bereits mit der aktuellsten Version verfügbar. Weitere Features werden nach und nach hinzukommen und die Nutzungsmöglichkeiten erweitern.

Wie immer haben wir für Sie eine Reihe an Tipps und Tricks zusammengestellt.
Viel Spaß mit den Tipps der aktuellen Ausgabe.

Grüße an unseren neuen Leser in Burundi – Vielen Dank für die Anmeldung !
Ihr TechTeam

TechTipp: Sharded Tables

Das Feature „Sharded Tables“ (wörtlich: geteilte Tabellen) ermöglicht es, Tabellen über mehrere Server, die in einer Enterprise Replikation verbunden sind, zu verteilen. Hierbei besteht die Option, die Tabelle nach einem Key bewusst aufzuteilen, analog des Features der Fragmentierung/Partitionierung (z.B. nach Monaten, Postleitzahl, …), oder eine Round-Robin Verteilung durchzuführen.
Das Ziel des Sharding ist die Verteilung von OLTP-Abfragen grosser Tabellen auf mehrere Server zur Verbesserung der Performance, bzw. zur Verbesserung der Skalierbarkeit. Stammdaten und kleinere Tabellen, die mit diesen Tabellen gejoint werden, sollten nach Möglichkeit komplett auf alle beteiligten Server repliziert werden, um die lokale Verarbeitung der Abfragen zu ermöglichen.

TechTipp: ONCONFIG - SHARD_MEM

Der Parameter SHARD_MEM bestimmt, wie der Speicher für „sharded queries“ auf einem „shard server“ genutzt wird.

Mögliche Werte sind:

- 0: Der Speicher für „sharded queries“ wird aus einem einzigen Memorypool geholt. (dies ist der Default)
- 1: Der Speicher wird anhand der genutzten virtuellen Prozessoren vom Typ „cpu“ geholt, auf denen die „Sharded Query“ verarbeitet wird. Die Zuteilung erfolgt somit parallel im Gegensatz zur Option 0.
- 2: Der Speicher wird aus einem freien Bereich mit einer festen Blockgrösse geholt, so dass sofort der erste verfügbare Block genutzt werden kann. Da kleinere Blöcke nicht genutzt werden, erhöht dies den Speicherbedarf, resultiert jedoch in einer schnelleren Bereitstellung und Freigabe der genutzten Blöcke.

Der Parameter kann in der Konfigurationsdatei „$ONCONFIG“ geändert werden und ist dann nach einem Neustart wirksam. Eine dynamische Anpassung mittel „onmode -wf/-wm“ ist möglich.

TechTipp: ONCONFIG - SHARD_ID

Der Parameter SHARD_ID identifiziert einen Server, der in der Replikation am Sharding beteiligt ist. Der Wert kann zwischen 0 (default) und 65535 gewählt werden. Wichtig ist, dass der Wert auf allen Servern des Sharding eindeutig ist (also keine doppelte Vergabe der Shard_ID erfolgt ist).

Der Wert kann in der Konfigurationsdatei $ONCOFIG gesetzt werden. Stand der Wert beim Start des Servers auf 0 oder war nicht gesetzt, dann ist eine dynamische Änderung mittels „onmode -wf/-wm“ möglich.

TechTipp: Sharded Tables - Beispiel

Die Nutzung von Sharded Tables wollen wir an einem einfachen Beispiel zeigen.

Im ersten Schritt erstellen wir eine Instanz (shard1), die für die Nutzung in der Enterprise Replikation vorbereitet ist (in der also zumindest in der Konfiguration der Parameter CDR_QDATA_SBSPACE gesetzt wurde).

Von dieser Instanz erstellen wir zwei Kopien (shard2, shard3) mittels „renamed restore“ und ändern jeweils die SHARD_ID auf einen eindeutigen Wert. Die Datei SQLHOSTS erweitern wir um die Einträge für die Replikationsgruppen (shard1_rep, shard2_rep, shard3_rep) wie bei der üblichen Einrichtung der Enterprise Replikation.

Nachdem alle drei Instanzen aktiv sind und die Vorbereitungen abgeschlossen wurden, nehmen wir die drei Instanzen in die Replikation auf:

```
cdr define server shard1_rep -I
cdr define server -c shard2_rep shard2_rep -I -S shard1_rep
cdr define server -c shard3_rep shard3_rep -I -S shard1_rep
```

und überprüfen das Resultat:

```
cdr list server
SERVER         ID STATE    STATUS      QUEUE  CONNECTION CHANGED
-------------------------------------------------------------------
shard1_rep    161 Active   Local           0
shard2_rep    162 Active   Connected       0 Feb 16 21:32:56
shard3_rep    163 Active   Connected       0 Feb 16 21:33:10
```

Nun erstellen wir auf allen Servern jeweils die identische Datenbank „shard_nl“ und darin die Tabelle „shard_table_tab“:

```
CREATE DATABASE if not exists shard_nl in datadbs WITH LOG;
CREATE TABLE if not exists shard_table_tab(
    id          INT,
    tabname     VARCHAR(128),
    type        CHAR(1) ,
    primary key (id)
) WITH CRCOLS;
```

Sind diese Vorbereitungen abgeschlossen, können wir unsere (noch leere) Tabelle als „Sharded Table“ definieren. Dies erfolgt mit dem Aufruf von „define shardCollection“.

Die Syntax hierfür ist:

```
cdr define shardCollection <shardColName>    \
     <datenbank>:<owner>.<table> \
     --type=<type> --key=<key_cols> --strategy=<strategy> \
     --versionCol=<column> \
     <repserver1> <repserver2> ... bzw. Fragmentierungsanweisungen
```

Als <**type>** sind die Optionen

- **delete** - auf andere Server replizierte Sätze werden lokal gelöscht
- **keep** – auf andere Server replizierte Sätze werden lokal als Kopie gehalten
- **informational** – Datensätze werden nicht repliziert/verteilt
  Die Verteilung erfolgt über lokale Inserts

möglich.

Der **<key>** gibt an, welche Spalten der Tabelle der Verteilungsschlüssel sind.

Die **<strategy>** kann entweder „hash“ oder „expression“ sein.
„hash“ ist eine annähernd gleichmässige Verteilung auf die Server
„expression“ ermöglicht eine bewusste Verteilung z.B. nach Jahr, Postleitzahl, ….

Die versionCol **<version>** ist beim Typ „delete“ wichtig, da hierüber bei Updates entschieden wird, welcher Datensatz neuer ist (bei Änderungen auf mehreren Servern). Hier kann z.B. die versteckte Spalte „cdrtime“ eingetragen werden, die mit den CRCOLS erstellt wird.

Abschliessend wird die Sharding Strategie bestimmet. Beim Typ „hash“ wird die Liste der teilnehmenden Server angegeben, bei „expression“ ein Ausdruck ähnlich dem bei der Partitionierung/Fragmentierung (siehe Beispiel):

```
cdr define shardCollection shard_coll_tab    \
     shard_nl:informix.shard_table_tab \
     --type=delete --key=id --strategy=expression \
     --versionCol=cdrtime \
     shard1_rep " BETWEEN 0 and 30 " \
     shard2_rep " BETWEEN 31 and 60 " \
     shard3_rep " > 60 "
```

Durch den Aufruf werden weitere Tabellen und Strukturen erstellt, wozu als Rückgabe nach dem Aufruf ein Protokoll ausgegeben wird:

```
create table informix.shard_coll_tab_errors (
     shard_target       integer,
     shard_key          integer,
     shard_txcode       integer,
     shard_dscode       integer,
     shard_sqlerror     integer,
     shard_isamerror    integer,
     id   integer,  tabname   varchar(128),
     type char(1),
     primary key (shard_target, shard_key)
) lock mode row


Verification of shard_nl@shard1_rep:informix.shard_table_tab
started
Verification of shard_nl@shard1_rep:informix.shard_table_tab is
successful
Verification of shard_nl@shard2_rep:informix.shard_table_tab
started
Verification of shard_nl@shard2_rep:informix.shard_table_tab is
successful
Verification of shard_nl@shard3_rep:informix.shard_table_tab
started
Verification of shard_nl@shard3_rep:informix.shard_table_tab is
successful

create table informix.shard_coll_tab_check (
     id   integer not null,
     tabname   varchar(128),
     type char(1),
```

```
     primary key (id))
 FRAGMENT BY EXPRESSION
PARTITION PART_1 (id  BETWEEN 0 and 30 ) in rootdbs,
PARTITION PART_2 (id  BETWEEN 31 and 60 ) in rootdbs,
PARTITION PART_3 (id  > 60 ) in rootdbs
```

Nun ist die Sharded Tabelle table_shard_tab angelegt und bereit für Daten. Wir füllen den Inhalt der ersten drei Spalten der Tabelle „systables“ in die neue Tabelle:

```
insert into shard_table_tab (id, tabname, type)
     select tabid, tabname, tabtype from systables;

78 row(s) inserted.
```

Die Datensätze wurden anhand der Regeln für die Sharding Collection auf die Instanzen verteilt. Jeder Shard erhält damit seinen Anteil an Daten. Durch die Angabe des Parameters „delete“ wurden die Daten auf Ursprungsserver gelöscht.

TechTipp: Environment - USE_SHARDING

Wer Sharded Tables nutzt, der sollte sich zuerst im Klaren sein, dass es zwei Sichten auf diese Tabellen gibt:

- Lokal: Es werden nur die lokal abgelegten Daten gesehen und geändert (default)
- Global: Es werden alle Daten der Tabelle berücksichtigt (USE_SHARDING 'on')

Die üblichen SQL-Befehle wie SELECT, UPDATE oder DELETE arbeiten jeweils lokal auf ihren Shards. Nur der INSERT berücksichtigt das Sharding automatisch und verteilt (je nach Einstellung) die Daten.

Die erste Überraschung erlebt man somit bereits, wenn man nach dem Insert die Daten abfragen will. (Annahme, wir befinden uns auf dem Server „shard1“).

Hatte der Insert gerade erst noch
„76 rows(s) inserted.“
gemeldet, bringt die Abfrage mit „select * ...“ nach der Ausgabe
„30 row(s) retrieved“
zurück.

Dieses Ergebnis lässt sich nicht einmal mehr mit dem bayerischen „a bisserl Schwund is immer“ erklären. Mit Sharding wird das Ergebnis allerdings verständlich.

Die lokalen Abfragen auf den anderen Server des Sharding zeigen den Verbleib der restlichen Datensätze:

```
Shard1: „30 row(s) retrieved. “
Shard2: „30 row(s) retrieved. “
Shard3: „16 row(s) retrieved.“
```

Die Gegenprobe mittels „delete“ auf einem der Server löscht nur die lokalen Einträge, lässt die anderen Teile der Sharded Table unverändert.

Will man mit allen Daten der Tabelle arbeiten, so muss zuvor im SQL die Umgebungsvariable „**USE_SHARDING**“ gesetzt werden. Die Abfrage mit gesetzter Variable ergibt dann:

```
set environment USE_SHARDING "on";
select * from shard_table_tab;

76 row(s) retrieved.
```

Ebenso würde der Aufruf „delete“ die Daten mit gesetzter Umgebung aus allen Shards der Tabelle löschen.

Wird erstmalig eine Sharded Collection angelegt, so erstellt die Instanz implizit eine weitere Systemdatenbank „sysclix“, was auch im online.log zu sehen ist:

```
02/16/16 23:02:31  CLIX: Building sysclix database
02/16/16 23:02:32  CLIX: Successful building sysclix
```

und startet Transportthreads:

```
02/16/16 23:07:22  smx creates 2 transports to server shard1
02/16/16 23:07:22  smx creates 2 transports to server shard2
02/16/16 23:07:22  smx creates 1 transports to server shard2
02/16/16 23:07:22  smx creates 2 transports to server shard3
02/16/16 23:07:22  smx creates 1 transports to server shard3
02/16/16 23:07:22  CLIX: prepare node shard1_rep
02/16/16 23:07:22  CLIX: prepare node shard2_rep
02/16/16 23:07:22  CLIX: prepare node shard3_rep
02/16/16 23:07:22  CLIX: Transaction  response: shard2_rep, 62
02/16/16 23:07:22  CLIX: Transaction  response: shard3_rep, 62
02/16/16 23:07:22  CLIX: Transaction  response: shard1_rep, 62
02/16/16 23:07:22  CLIX: prepare node shard1_rep
02/16/16 23:07:22  CLIX: prepare node shard2_rep
02/16/16 23:07:22  CLIX: prepare node shard3_rep
02/16/16 23:07:22  CLIX: Transaction  response: shard2_rep, 62
02/16/16 23:07:22  CLIX: Transaction  response: shard3_rep, 62
02/16/16 23:07:22  CLIX: Transaction  response: shard1_rep, 0
02/16/16 23:07:22  CLIX: com/roll 7, node shard1_rep
02/16/16 23:07:22  CLIX: Transaction  response: shard1_rep, 0
```

Für den Datenaustausch werden etliche Replikate erstellt, die mit „cdr list repl brief“ zu sehen sind. Die Threads können in der Ausgabe des „onstat -g ath“ aufgelistet werden.

Hinweis:
In der aktuellen Implementierung kann kein Join zwischen Sharded Tables erfolgen, sondern nur zwischen diesen und normalen, nicht verteilten Tabellen.
Wird versucht zwei Sharded Tables miteinander zu kombinieren wird eine Fehlermeldung zurückgegeben:

```
26991: The query failed because the FROM clause includes more than
       one sharded table.
```

TechTipp: Optionen des ONSTAT (onstat -g shard)

Das Feature Sharding wurde in den ONSTAT integriert, so dass die ShardCollections, die erstellt wurden, mittels „onstat -g shard“ angezeigt werden können:

```
onstat -g shard

IBM Informix Dynamic Server Version 12.10.FC6AEE -- On-Line -- Up 00:15:45 --
538328 Kbytes

shard_coll_tab shard_nl:informix.shard_table_tab  key:id EXPRESSION:DELETE SHARD
OPTIMIZATION:ENABLED
Matching for delete:cdrtime
   shard1_rep (10551323) id  BETWEEN 0 and 30
   shard2_rep (10551324) id  BETWEEN 31 and 60
   shard3_rep (10551325) id  > 60
```

Zu sehen ist neben der Definition auch die Option „delete“ mit der VersionColumn „cdrtime“. Zudem ist zu sehen, dass die Optimierung der Abfragen mittels Sharding aktiviert ist.

TechTipp: Sharding im SQEXPLAIN

Die Vorteile des Sharding zeigen sich direkt in der Ausgabe des sqexplain.
Hier wird ausgegeben welche Shards für die Verarbeitung genutzt wurden.

Beispiel:

```
QUERY: (OPTIMIZATION TIMESTAMP: 02-21-2016 14:42:42)
------
select t.id, c.colname
from shard_table_tab t,
      syscolumns c
where t.id = c.tabid
and t.id >= 35 and t.id <= 55

Estimated Cost: 4
Estimated # of Rows Returned: 1

  1) informix.t: REMOTE PATH
    REMOTE SESSION ID FOR 'CLIX_2' is UNKNOWN (shard2_rep)
    Remote SQL Request:
    select x0.id ::integer from shard_nl:"informix".shard_table_tab x0 where
      ((x0.id <= 55 ) AND (x0.id >= 35 ) )



QUERY: (OPTIMIZATION TIMESTAMP: 02-21-2016 14:45:20)
------
select t.id, c.colname
from shard_table_tab t,
      syscolumns c
where t.id = c.tabid
--and t.id >= 35 and t.id <= 55;
and t.id > 55

Estimated Cost: 4
Estimated # of Rows Returned: 3
  1) informix.t: REMOTE PATH
    REMOTE SESSION ID FOR 'CLIX_1' is UNKNOWN (shard2_rep, shard3_rep)

    Remote SQL Request:
    select x0.id ::integer from shard_nl:"informix".shard_table_tab x0 where
      (x0.id > 55 )
```

TechTipp: Änderung der Einstellungen für Tenants (Task: „tenant update“)

Tenants können mittels des Tasks „tenant update“ verändert werden. Sowohl die Limits des Tenant, als auch die Session Limits können hierbei verändert werden.

Wird die Angabe der vpclass verändert, so ersetzt diese die bisherigen zugeordneten VPs. Werden neue DBSpaces, TempSpaces, Blobspaces oder SBSpaces angegeben, so werden diese zusätzlich zu den bisher zugeordneten Spaces vom Tenant genutzt.

Im folgenden Beispiel wird dem Tenant “carmen“ zusätzlich der Platz in den DBSpaces „datadbs42“ und „datadbs52“, sowie dem Smartblobspace „sbspace23“ gegeben. Statt dem bisher genutzten VP sollen nun die VPs mit dem Namen „carmen_vp“ genutzt werden und die maximale Transaktionsdauer wird auf 120 Sekunden begrenzt:

```
EXECUTE FUNCTION task('tenant update', 'carmen',
                      '{dbspace:"datadbs42,datadbs52",
                        sbspace:"sbspace23",
                        vpclass:"carmen_vp",
                        session_limit_txn_time:"120"}'
);
```

Mit dieser Option lassen sich bei aktiven Tenants Anpassungen vornehmen, so dass dynamisch auf veränderte Anforderungen an Platz und Resourcen reagiert werden kann. Dies ermögliche die optimale Anpassung von Subkapazitäten an die Anforderungen der Kunden des Systems.

Anmeldung / Abmeldung / Anmerkung

Der Newsletter wird ausschließlich an angemeldete Adressen verschickt. Die Anmeldung erfolgt, indem Sie eine Email mit dem Betreff „**ANMELDUNG**“ an **ifmxnews@de.ibm.com** senden.
Im Falle einer Abmeldung senden Sie „ABMELDUNG“ an diese Adresse.

Das Archiv der bisherigen Ausgaben finden Sie zum Beispiel unter:

http://www.iiug.org/intl/deu
http://www.iug.de/index.php?option=com_content&task=view&id=95&Itemid=149
http://www.informix-zone.com/informix-german-newsletter
http://www.drap.de/link/informix
http://www.nsi.de/informix/newsletter
http://www.cursor-distribution.de/index.php/aktuelles/informix-newsletter
http://www.listec.de/Newsletter/IBM-Informix-Newsletter/View-category.html
http://www.bereos.eu/software/informix/newsletter/

Die hier veröffentlichten Tipps&Tricks erheben keinen Anspruch auf Vollständigkeit. Da uns weder Tippfehler noch Irrtümer fremd sind, bitten wir hier um Nachsicht falls sich bei der Recherche einmal etwas eingeschlichen hat, was nicht wie beschrieben funktioniert.

Die Autoren dieser Ausgabe

Gerd Kaluzinski IT-Specialist für Informix, DB2 LUW, InfoSphere Replikation
IBM Software Group, Information Management
gerd.kaluzinski@de.ibm.com +49-175-228-1983

Martin Fuerderer IBM Informix Entwicklung, München
IBM Software Group, Information Management
martinfu@de.ibm.com

Markus Holzbauer IBM Informix Advanced Support
IBM Software Group, Information Management Support
holzbauer@de.ibm.com

Sowie unterstützende Teams im Hintergrund.

Fotonachweis: Carmen Kaluzinski (Lindau in der NarrenNacht2016)