Efficiënt wijzigingen doorgeven van bron- naar doeldatabase

Change Data Capture (3)

Toon Loonen

Dit is het laatste deel van de serie over het efficiënt uitvoeren van replicatieprocessen.

Bij het repliceren van grote hoeveelheden gegevens kan de performance/doorlooptijd een probleem worden. Hierboven hebben we al vermeld dat sommige technieken betere prestaties (doorlooptijden) leveren dan andere. Als voor de replicatie de meest optimale methode is gekozen, en de performance is nog onvoldoende, dan zijn er nog diverse mogelijkheden om de snelheid te verbeteren, zoals: allereerst natuurlijk alle standaard technieken voor performanceverbetering in een (relationele) database omgeving zoals het gebruik van indexen, enzovoort [Ref 3 in DB/M 5]; minder tussenbestanden gebruiken, meer acties in het geheugen uitvoeren; minder vaak de replicatieslag uitvoeren; het parallel uitvoeren van diverse taken, enzovoort.
Het is bijvoorbeeld mogelijk om de mutaties van de Klanten en de Artikelen eerst parallel door te geven. Daarna, wanneer deze beide processen klaar zijn, kunnen de mutaties op de Orders worden verwerkt in bijvoorbeeld 10 parallelle processen.
Elk van deze processen krijgt een kengetal van 0 tot 9: getal N. Vervolgens verwerkt het proces N alle Orders waarvoor het laatste cijfer van het Ordernummer (ofwel het Ordernummer modulo 10) gelijk is aan N. Bijvoorbeeld: Proces 5 verwerkt de Orders met Ordernummer 1005, 1015, 1025, enzovoort. Parallelle verwerking heeft meestal alleen zin als er gewerkt wordt op een server met meerdere processoren. Verder kan de snelheid van het netwerk ook een beperking vormen.

Commit grootte en herstel na een fout

Een replicatieproces kan halverwege een batch vastlopen, bijvoorbeeld wegens gebrek aan schijfruimte, een netwerkprobleem enzovoort. Dan moet het mogelijk zijn om de batch opnieuw op te starten nadat de oorzaak van de storing is opgelost [Ref 5 in DB/M 5]. Mogelijk is een deel van de batch al gecommit. Er zijn nu verschillende scenario's (uitgaande van een tussenbestand met de mutaties): het hele proces is één transactie. Bij het vastlopen wordt de hele batch op de doeldatabase gerollbackt. Na herstel van de oorzaak van het probleem kan de hele batch opnieuw opgestart worden en gaat deze automatisch opnieuw de gewenste mutaties ophalen en verwerken; wanneer het replicatieproces is opgesplitst in verschillende delen/transacties, bijvoorbeeld één transactie voor de Klanten, één transactie voor de Artikelen en een transactie voor elk van de 10 parallelle processen voor de Orders, zoals hierboven beschreven, dan moet men in een hulptabel registreren welke onderdelen zijn afgerond en welke onderdelen nog moeten worden overgedaan. Bij het herstarten van het proces moet het systeem dit herkennen en alleen die processen uitvoeren die in de vorige run niet gelukt zijn; wanneer een grote transactie, bijvoorbeeld een replicatie van vele duizenden Orders met Orderregels, uitgevoerd wordt, dan is het raadzaam om voor elke één, 100 of 1000 Orders een commit te doen en daarbij het laatste verwerkte Ordernummer vast te leggen in een hulptabel. Als daarna de batch vastloopt, dan kan het proces op dit punt worden doorgestart. De verwerking moet dan wel op volgorde van Ordernummer worden uitgevoerd; een alternatief hiervoor is het opnemen van een batchnummer in elk gerepliceerd record en, na een storing, alle records van de mislukte batch te verwijderen en het replicatieproces opnieuw op te starten. Dit is echter lastiger als er behalve inserts ook updates en deletes gerepliceerd worden.
In feite moet het replicatieproces altijd zo ontworpen worden dat het ervan uit gaat dat de vorige batch mislukt is.
Een van de hiervoor genoemde opties om een zeer grote hoeveelheid gegevens te repliceren is om dagelijkse partities te definiëren en een dagelijkse batch te gebruiken om de hele partitie naar de doeldatabase te kopiëren. Bij deze grote aantallen zal een tussentijdse commit ook gewenst zijn. Wanneer deze batch vastloopt kan men op de doeldatabase deze partitie geheel leeg maken (delete/truncate partition) en de batch opnieuw opstarten.

Push/Pull

Het is, bij de opzet (architectuur) van de replicatie, goed om vast te leggen waar het initiatief van de replicatie ligt: *Push* – bij de brondatabase: een trigger maakt een mutatiebericht aan;
Pull – bij de doeldatabase: periodiek draait er een batch met SQL scripts die via een databaseconnectie de mutaties op de brondatabase leest en lokaal verwerkt; *Pull/Push* – een ETL-tool is vaak een systeem dat onafhankelijk van de bron of doeldatabase op een eigen server draait. Dit tool kan op de bron de mutaties oppakken (pull) en via en tussenbestand of direct, verwerken in de doeldatabase (push).
Voor het direct bijwerken van een doeldatabase is het Pushmechanisme nodig. Pull kan alleen periodiek in batches.
Bij Push is het ook mogelijk om een trigger of ander proces in de

brondatabase zo te bouwen dat deze de mutatie ook direct verwerkt in de doeldatabase: synchrone verwerking binnen 1 fysieke transactie met behulp van een 'two fase commit'. Dit creëert echter een grote afhankelijkheid: als de doeldatabase down is dan kunnen er ook geen mutaties meer verwerkt worden op de bron. Ook bij netwerkproblemen richting de doeldatabase zijn er geen mutaties op de bron meer mogelijk. Daarom is het beter om deze transacties los te koppelen en dus asynchroon te verwerken, met natuurlijk de consequentie dat soms de doeldatabase wat (seconden, minuten of soms meer) achterloopt op de bron.
Bij het push-mechanisme moet het systeem dat de berichten oppakt en doorgeeft (messagebus of Enterprise Service Bus) ook deel uitmaken van het transactiemechanisme: de update op de brondatabase en het plaatsen van een bericht op de messagebus is 1 transactie. Het verwerken van dit bericht op de doeldatabase en het verwijderen van het bericht van de bus (ofwel: de status wordt op 'verwerkt' gezet) moet eveneens 1 transactie zijn.

Wijzigingen op het datamodel

Als het datamodel van de brondatabase of de doeldatabase wordt gewijzigd, dan moet de impact van deze wijziging op de andere database(s) en op het replicatieproces zorgvuldig worden geanalyseerd. Bij deze wijzigingen kan men denken aan: nieuwe gegevens (nieuwe tabellen of kolommen) in de brondatabase die niet naar de doeldatabase worden doorgegeven: er is geen impact op de doeldatabase en het ETL-proces; nieuwe gegevens (nieuwe tabellen of kolommen) in de brondatabase die wel naar de doeldatabase worden doorgegeven: het model van de doeldatabase en het ETL-proces moet hierop worden aangepast;
Is het doelsysteem ook geïnteresseerd in reeds bestaande gegevens van het bronsysteem die voorheen niet werden doorgegeven, dan moeten het model van de doeldatabase en het ETL-proces hierop worden aangepast en tevens moet mogelijk ook de historie worden doorgegeven, dus de gegevens in deze tabellen/kolommen die al langer in de brondatabase staan.
Daarnaast kan het zijn dat er een derde of vierde brondatabase op de doeldatabase wordt aangesloten. Zeker bij een datawarehouse als doelsysteem zal dit vaak voorkomen. In dat geval zal het datamodel van het doelsysteem worden aangepast zonder dat dit consequenties hoeft te hebben voor één van de bronsystemen. Wel moet het ETL-proces worden uitgebreid en ook goed worden gekeken naar de volgorde van batches in dit ETL-proces.

Database herstellen

Wat gebeurt er met deze replicatie-interface als één van de databases moet worden hersteld vanaf een backup? Meestal is het mogelijk om met een logfile de database bij te werken tot kort voor het onheil. Als dat niet mogelijk is of niet voldoende ver in de tijd, dan moeten we de twee databases (of meer brondatabases met één doeldatabase) weer in overeenstemming brengen. Hierboven hebben we in een procedure beschreven hoe met behulp van timestamps (tijd X en Y) alle veranderingen tussen deze twee tijdstippen te selecteren zijn, in een tussenbestand te plaatsen en het tussenbestand op de doeldatabase te verwerken is. Hiermee zijn de hierna genoemde oplossingen mogelijk. Met andere opties (directe interface tussen de twee databases zonder tussenbestand, triggers, log-bestanden) is een vergelijkbare oplossing mogelijk. Uitgangspunt is dus een proces op de brondatabase die het tussenbestand aanmaakt en een hulptabel bijwerkt waarin de laatste tijd Y is opgenomen van de laatste batch.
Als de doeldatabase is teruggezet vanaf de backup, dan is daar mogelijk een groot aantal transacties verloren gegaan. Als de tussenbestanden bewaard zijn, dan kunnen deze mogelijk opnieuw aan het doelsysteem worden aangeboden. Anders zullen de mutaties opnieuw verstuurd moeten worden door de brondatabase. Dit kan door de tijd van de laatst verstuurde mutatie in de hulptabel (in de brondatabase) terug te zetten naar een tijd voor of op de tijd waarop nog mutaties zijn verstuurd en verwerkt. Als daarna het replicatieproces weer wordt opgestart, dan wordt de doeldatabase weer netjes bijgewerkt. Hiervoor is het wel nodig dat het proces toestaat dat dezelfde wijziging een tweede keer naar de doeldatabase wordt afgeleverd.
Als de hulptabel in de doeldatabase staat, dan zal deze, na het terugzetten, automatisch al de goede tijd hebben en kan de replicatie meteen weer worden gestart. Als de brondatabase wordt teruggezet vanaf de backup zijn er enkele alternatieven. Voorkeur: de doeldatabase wordt ook teruggezet naar een tijdstip vóór de tijd dat de brondatabase is teruggezet. Daarna kan bovenstaande procedure weer gebruikt worden. Als dat niet acceptabel is kan de doeldatabase veel transacties verwachten die al eerder zijn verwerkt. Door daarop te controleren en duidelijk dubbele mutaties te negeren of te signaleren kan men ook de doeldatabase weer synchroon met de bron brengen.

Conclusie en aanbeveling

In dit artikel zijn de mogelijkheden bekeken om op een efficiënte manier wijzigingen op gegevens in een brondatabase op te pakken en door te geven naar een doeldatabase. Afhankelijk van de eisen en mogelijkheden zal een van de hiervoor beschreven opties (of een mix hiervan) gekozen kunnen worden: voor realtime replicatie is het gebruik van database triggers het meest geschikt; voor near real-time- en voor gewone batch interfaces zal het lezen van de mutaties uit een logbestand van het DBMS, indien mogelijk, de voorkeur verdienen; als dat niet mogelijk is dan zal een selectie op basis van een kolom met de laatste mutatietijd de beste optie zijn; bij gebruik van (ERP, CRM of HRM) pakketten als bron zal een tool van de pakketleverancier of een ETL-tool nodig zijn om de mutaties te verzamelen; voor grote tabellen met transactiegegevens is het repliceren via (dagelijkse) partities aan te bevelen; voor kleine tabellen met referentiegegevens is het volledig overhalen van deze gegevens (al of niet gewijzigd) in plaats van alleen de wijzigingen soms ook een optie: het is eenvoudiger te bouwen en de kleine overhead compenseert een ingewikkelder selectieproces.

Toon Loonen (toon.loonen@capgemini.com) werkt bij Capgemini.