Samenvatting

Wijzigingen aanbrengen in gecomprimeerde kolom-opslag

De hedendaagse maatschappij vertrouwt steeds vaker op informatietechnologie om zowel gegevens als dagelijkse activiteiten te beheren. Door de jaren heen heeft dit geleid tot een exponentiële groei in de hoeveelheid informatie die wordt verwerkt en opgeslagen door computers. Sinds 1970 zijn *database management systemen* (DBMS) gebruikt om digitale informatie op gestructureerde wijze te organiseren, uit te vragen en te manipuleren. Vroege databasesystemen richtten zich hoofdzakelijk op *transactie verwerking*, dat zich bezigt met het ophalen en manipuleren van data items, zoals, bijvoorbeeld, het saldo op een bankrekening of adresgegevens van een klant. Met de tijd heeft het snel groeiende volume van door organisaties verzamelde gegevens een behoefte aangewakkerd om een meer globaal inzicht te krijgen in dergelijke gegevens, met als doel het verstrekken van, bijvoorbeeld, historische of voorspellende inzichten aangaande de bedrijfsvoering. Deze nieuwe, *analytische* manier van data verwerken wordt vaak *business intelligence* genoemd, en wordt gekarakteriseerd door een "ad-hoc" query (vraag) patroon, vaak over grote hoeveelheden data, in combinatie met een competitief of economisch voordeel indien deze vragen snel beantwoord worden.

De groei in datavolume is gepaard gegaan met een exponentiële vooruitgang binnen bepaalde eigenschappen van computerhardware, met name verwerkingskracht van processoren (CPUs) en opslagcapaciteit van zowel werkgeheugen als schijfopslag. Echter, zowel de *responstijd* ten aanzien van data verzoeken als de *bandbreedte* voor het transport van de data zijn bij deze exponentiële trend achtergebleven. Het gevolg is dat softwareontwikkelaars geconfronteerd worden met een groeiende discrepantie tussen verwerkingskracht aan de ene kant, en trage toegang tot data aan de andere. Voorts bevoordelen de blok-geörienteerde toegangsmethoden, die opslag-electonica verstrekken, in toenemende mate sequentiële boven lukrake toegangspatronen, aangezien ontwikkelingen in responstijd achterblijven bij ontwikkelingen rond bandbreedte.

Voor analytische database toepassingen, die over het algemeen grote hoeveelheden data verwerken, hebben deze trends geleid tot brede acceptatie van zogenaamde *kolom-georiënteerde* opslagtechnieken. De meeste databasesystemen hanteren een relationeel model, waarbinnen gegevens worden opgeslagen in tweedimensionale *tabellen*, bestaande uit *rijen* en *kolommen*. Elke rij, of *tuple*, vertegenwoordigt een zekere entiteit, waarbij elke kolom een *attribuut*, of eigenschap, van die entiteit voorstelt. In een kolom-geörienteerd opslagmodel, het zogeheten *decomposed storage model* (DSM), worden tabellen verticaal gepartitioneerd opgeslagen, wat inhoudt dat attribuutwaarden uit een enkele kolom aaneengesloten worden opgeslagen binnen een *pagina* – de transporteenheid tussen schijf en geheugen – van vaste grootte. Het voordeel is dat voor vragen die slechts toegang tot een deelverzameling van kolommen in een tabel nodig hebben, het volstaat om alleen de pagina's van relevante kolommen van schijf te lezen, wat kan leiden tot een substantiële besparing van schijfbandbreedte ten opzichte van een rij-georiënteerd opslagmodel, het zogeheten *N-ary storage model* (NSM), waar een pagina altijd een reeks volledige tuples bevat.

Maar kolom-georiënteerde opslag biedt ook voordelen op het gebied van dataverwerkingsefficiëntie binnen de processor. Door kleine fragmenten, of *vectoren*, met attribuutwaarden uit een pagina binnen het geheugen direct naar de CPU-cache te lezen, wordt optimaal gebruik gemaakt van cachelijnen (de transporteenheid tussen geheugen en CPU), waarmee vergelijkbare bandbreedtebesparingen kunnen worden behaald als bij het lezen van kolom-georiënteerde pagina's van disk. Tevens kunnen database *primitieven*, verantwoordelijk voor het uitvoeren van de uiteindelijke berekeningen over vectoren met data, worden geïmplementeerd als simpele en efficiënte lussen (loops) over reeksen (arrays), die vrij zijn van functieaanroepen en vertakkingen, wat compilers in staat stelt om CPU-efficiënte code te genereren. Onderzoek omtrent dit *gevectoriseerde uitvoerings model* heeft geresulteerd in het kolom-georiënteerde *MonetDB/X100* DBMS, dat later is gecommercialiseerd onder de noemer *Vectorwise*.

Het werk in deze dissertatie biedt oplossingen voor twee problemen die kunnen voorkomen bij kolom-georiënteerde en gevectoriseerde databasesystemen zoals Vectorwise: *(i)* een gevaar voor knelpunten aangaande input/output (I/O) bandbreedte, die zelfs bij gebruik van DSM nog kunnen voorkomen, en *(ii)* hoe te voorzien in functionaliteit die het mogelijk maakt om op efficiënte wijze transactionele wijzigingen aan te brengen binnen een opslagstructuur die uiterst ongeschikt is voor het aanbrengen van lokale wijzigingen.

Gegeven de groeiende discrepantie tussen processorkracht enerzijds en schijfbandbreedte anderzijds, is het soms onmogelijk om ruwe data vanaf schijf aan te leveren met een snelheid die overeenkomt met de verwerkingskracht van een gevectoriseerde databasekern. We tonen aan dat datacompressie gebruikt kan worden om dergelijke knelpunten significant te verlichten, zowel bij analytische als IR (information retrieval) toepassingen. Het idee is om gecomprimeerde pagina's van schijf te lezen, en onder minimale CPU-belasting te decomprimeren, zodat de waargenomen I/O-bandbreedte verbeterd kan worden, met maximaal een factor gelijk aan de compressie ratio. We verruilen hier CPU-rekenkracht, hetgeen zich gunstig ontwikkelt, tegen waargenomen schijfbandbreedte, hetgeen schaars is.

Om dergelijke technieken ook op snelle schijfsystemen, zoals RAID (redundant array of independent disks) of SSD (solid-state drive), te laten werken, is het van belang dat (de)compressiemethoden *lichtgewicht* zijn. We dragen drie nieuwe compressie schema's bij, PFOR, PFOR-DELTA en PDICT, die specifiek zijn ontworpen voor de superscalaire infrastructuur van moderne CPUs. Deze algoritmen maken gebruik van de observatie dat waarden binnen een kolom allen uit hetzelfde domein afkomstig zijn en hetzelfde datatype hebben, wat ons in staat stelt om (de)compressie simpel en snel – gigabytes per seconde – te houden. Tevens tonen we aan dat door gecomprimeerde pagina's binnen het werkgeheugen slechts met lage granulariteit, naar behoefte, te decomprimeren, direct *naar de CPU-cache*, we kunnen voorkomen dat geheugenbandbreedte een knelpunt wordt.

Om transactionele wijzigingen aan te brengen binnen een kolom-opslag DBMS, beschouwen we het lokaal aanbrengen van wijzigingen als niet afdoende, vanwege het gegeven dat de I/O kosten proportioneel stijgen met het aantal attributen van een tabel. Liever richten wij ons tot differentiële wijzigingstechnieken, waar wijzigingen ten opzichte van anderszins onveranderlijke tabellen in een voor scans geoptimaliseerde "lees-opslag", worden bijgehouden in een makkelijk te wijzigen "schrijf-opslag" in het werkgeheugen. Tijdens een scan worden wijzig-

ingen uit de schrijf-opslag vervlochten met de tabel uit lees-opslag om de huidige toestand van de tabel te reproduceren. Het idee is om werkgeheugencapaciteit, hetgeen zich gunstig ontwikkelt, te verruilen voor een reductie in het aantal I/O operaties, die kostbaar zijn, zowel in termen van responstijd als bandbreedte.

We stellen voor om differentiële wijzigingen ten opzichte van een tabel in lees-opslag bij te houden op basis van tuple *positie*, in een nieuwe boomvormige indexstructuur genaamd *positional delta tree* (PDT). Alternatief zou men wijzigingen kunnen bijhouden en rangschikken op basis van de *waarde* van (sorteer) sleutelattributen. We tonen aan dat het positioneel bijhouden van wijzigingen voordelen biedt ten opzichte van een op waarden gebaseerde aanpak, met name bij het vervlechten van wijzigingen tijdens een scan. Tevens maken PDTs een efficiënte encodering van modificatie (SQL UPDATE) mogelijk, en kunnen zij boven op elkaar gestapeld worden, wat een overzichtelijke implementatie van momentopname isolatie (snapshot isolation) levert om te voorzien in transacties.