Vergleichende Untersuchung von Datenbanksystemen mit In-Memory-Technologien

Marcel Kunz, Robin Arnoldt, Clemens Köhler,
Phillipp Winkler, Moritz Buchwälder, Robert Pietzschmann

26.02.2018

Inhaltsverzeichnis

1 Einleitung

Während unseres Projektseminars war es unsere Aufgabe Datenbanksysteme mit In-Memory Technologie vergleichend zu untersuchen. Betreuer bzw. Auftraggeber dessen war Prof. Dr. oec. Gunter Gräfe.
Wir hatten dabei mehrere Schwerpunkte. Es musste untersucht werden, wie Hochverfügbarkeit und Performancesicherung bei In-Memory-Technologien umgesetzt wurden. Es mussten Strategien erarbeitet werden, die zeigen wie Anforderungen an Verfügbarkeit, Ausfalltoleranz, Performance und Konsistenz in den unterschiedlichen Systemen, um transaktionale Daten zu verwalten und auf der anderen Seite die Analyse von großen Daten umgesetzt wurden. Dazu mussten wir uns in die Technologie der In-Memory Datenbank SAP Hana Express und dem MS SQL Server 2016 einarbeiten. Neben diesen beiden Hauptsystemen war noch gefordert selbiges bei NO-SQL Systemen zu überprüfen. Dort wählten wir Cassandra, sowie eine Maria DB mit vorgestelltem Memcache aus. Weiterhin sollten wir auf mehreren Systemen die verschiedenen Technologien umsetzen und in einem selbstgewählten Beispielszenario mit Daten füllen. Diese wurden für die Vergleiche zwischen den Systemen benötigt. Ein möglicher Zusatz dazu sollte der Entwurf einer Übungsaufgabe für das Modul „Erweiterte Datenbanksysteme“.

2 Gruppenmitglieder

Unsere Gruppe bestand während des Projektes aus sechs Mitgliedern. Marcel Kunz, Robin Arnoldt, Clemens Köhler, Phillipp Winkler, Moritz Buchwälder und Robert Pietzschmann.

3 Aufgabenstellung

Schwerpunkte des Projektseminars

1. Vorstellung und Diskussion von In-Memory-Technologien unter dem Aspekt der Hochverfügbarkeit und Performancesicherung
2. Erarbeitung von Strategien für die Umsetzung von Anforderungen an Konsistenz, Verfügbarkeit, Performance und Ausfalltoleranz (CAP)
   bei verschiedenen Systemen für die Verwaltung transaktionaler Daten einerseits und die Analyse großer Datenmengen andererseits
3. Einarbeitung in die In-Memory-Funktionalitäten von SAP HANA Express und MS SQL Server 2016 hinsichtlich (alter und)
   neuer Features zur Sicherung von Hochverfügbarkeit und/oder Performance
4. Untersuchung der Möglichkeiten von Cache-/In-Memory-Technologien
   bei NoSQL-Datenbanken (z.B. Memcached)
5. Erarbeitung eines konzeptionellen Entwurfs für ein mögliches Beispielszenario
6. Prototypische Umsetzung von In-Memory-Technologien an mehreren Beispielsystemen (vergleichende Analyse)
7. Aufbereitung und Auswertung der Ergebnisse

4 Theorie

4.1 Grundlagen In-Memory Technologie

Der wichtigste Unterschied von einem In-Memory Datenbanksystem gegenüber einem herkömmlichen ist, dass die Datenhaltung im Hauptspeicher des Rechners erfolgt. Das führt zu erheblichen Vorteilen bei der Performance, da die Zugriffszeit so enorm verkürzt wird und Zugriffsalgorithmen sind einfacher. Nachteil ist vor allen der wesentlich höhere Preis für Arbeitsspeicher als für Festplatten. Da aber auch die Preise für jene mit ausreichender Speicherkapazität inzwischen nicht mehr unbezahlbar sind, werden In-Memory Datenbanken immer beliebter. Erst mit der Entwicklung der Kapazität der Hauptspeicher im letzten Jahrzehnt wurde es möglich In-Memory basierte Systeme zu schaffen. Wichtigste Vorraussetzung dafür war die Entwicklung von 64 Bit Systemen. Durch diese wurde es möglich ausreichend großen Arbeitsspeicher zu entwickeln. Im allgemeinen gilt für Speicherarchitekturen, dass je langsamer sie sind, desto billiger werden sie. Zuletzt veränderte sich zwar auch der Rest der Speicher grundlegend durch den Einzug von SSD Festplatten (Flash-Speichern), die wesentlich schneller sind als die Hard Disk. Allerdings sind diese aus Sicht der Software immer noch eine Platte wegen der Persistenz und der Nutzungseigenschaften. Dies hat zur Folge, dass für den Zugriff auf diese immer noch die gleichen Methoden verwendet werden wie für die Hard Disks. Zur vollen Ausnutzung der Geschwindigkeit des Flash Speichermediums müssen diese Algorithmen ständig erneuert werden. Der Hauptspeicher hat dagegen den Vorteil, dass ein direkter Zugriff möglich ist. So dauert das sequenziellen Lesen von einem MB hier nur 250000 ns gegenüber 1851851,9 ns bei einer aktuell schnellen SSD und 30000000 ns bei einer Hard Disk. Alle normalen Operationen der Datenbank werden deshalb im Hauptspeicher augeführt und die Festplatte wird nur für Backups bzw. als Archi genutzt. Als Problem ergibt sich, trotz der wesentlich besseren Performance ein Bottleneck beim Lader der Daten vom RAM in den Cache.
Einhergehend mit der Datenhaltung im Hauptspeicher gibt es einer weitere wesentliche Änderung gegenüber konventionellen Datenbanksystemen. So erfolgt die Datenhaltung spaltenorientiert. Dabei werden alle Reihen (Tupel) in angrenzenden Blöcken gespeichert. Dadurch eignen sie sich perfekt für einen schnellen lesenden Zugriff, was beispielsweise für die Aggregation sinnvoll ist. Zur Reduzierung der Datenmenge werden bei In-Memory Datenbanksystemen verschiedenen Kompressionsverfahren genutzt.
Aufgrund der Datenhaltung im Hauptspeicher ergibt sich noch ein Nachteil aus ökologischer und ökonomischer Sicht. So ist der Energieverbrauch einer von Main-Memory

Datenbankmanagementsystemen erheblich viel höher als der von konventionellen. So verbraucht ein „System“ ganze 1,5 Megawatt.

4.2 Verwendete Datenbanksysteme

4.2.1 MS SQL Server

Micorsoft SQl Server ist Datenbankmanagementsystem von Microsoft und auch eines der bekanntesten mit einem sehr weiten Verbeitungsgrad. Die erste Version wurde bereits in Kooperation mit der Firma Sybase 1989 veröffentlicht. Seit 1995 ist er in eigenständiger Entwicklung bei Microsoft. Eigentlich handelt sich es hier um ein klassisches relationales Datenbanksystem. Seit der Version 2014 stehen allerdings eine Reihe an IN-Memory Funktionen unter dem Namen „In-Memory OLTP“ zur Verfügung. So ist es jetzt mögliche einzelne/ mehrere Tabellen speicheroptimiert („memory-optimized“) anzulegen.
Speicheroptimierte Tabellen werden als Objekte der Programmiersprache C gehalten. Diese sind so implementiert, dass eine Anwendung genauso auf sie zugreifen kann, wie auf eine althergebrachte Tabelle im Dateisystem.
Klare Vorteile dieser Tabellen sind die Möglichkeit wesentlich schneller arbeiten zu können, wie auch eine Multiversionsverwaltung. Dies bedeutet, dass wenn mehrere Transaktionen auf dieselben Datensätze zugreifen, eine eigenständige Version dieser verwendet. Dadurch fallen Sperren weg. Negativ ist, dass unter Umständen Datenverluste durch die Datenhaltung im Hauptspeicher entstehen können, da es verschiedene Bestädigkeiten dieser Tabellen gibt. So kann man Tabellen mit beständigen und mit nicht beständigen Inhalten erstellen. Bei letzteren erfolgt keine zusätzliche Sicherung auf der Festplatte, weshalb dier Daten verloren sind, wenn es zu einem Ausfall, Neustart oder ähnlichen des Servers kommt.
Wird nun eine neue Tabelle mit beständigen Inhalt angelegt, erstellt MS SQL Server eine speicheroptimierte Dateigruppe mit einem Container. Dieser enthält Datendateien und Änderungsdateien. In der Dateigruppe erfolgt die zwischen Speicherung der Daten aus dem Arbeitsspeicher als Backup-Lösung. Eine speicheroptimierte Dateigruppe ist erforderlich, damit die Behandlung speicheroptimierter SCHEMA-ONLY-Tabellen für Datenbanken mit speicheroptimierten Tabellen konsistent ist.
Anfangs gab es noch zahlreiche Einschränkungen bei speicheroptimierten Tabellen. So waren SQL Befehle wie ALTER, LEFT/ RIGHT OUTER JOIN,1 OR, NOT, SELECT DESTINCT und einige andere nicht nutzbar, der maximal nutzbare Speicher auf 256 GB limitiert, Lagre Objects (LOBs) und einige weitere Dinge nicht unterstützt. Dies wurde mit der Version 2016 behoben. Der nutzbare Speicher wurde auf 2 TB erhöht.

4.2.2 SAP HANA

Bei SAP Hana handelt es sich um eine komplette In-Memory Datenbanksystem aus dem Hause SAP. Es wurde erstmalig 2010 vorgestellt und setzt sich seit dem immer weiter durch. Entwickelt wurde es von dem Hasso-Plattner-Institut in Kooperation mit der Stanford University aus den USA. In einigen Veröffentlichungen, wie auch einem hier stark als Quelle genutzten Buches, wurde es auch als „NewDB“ oder „SanssouciDB“ bezeichnet. In unserem Projektseminar nutzten wir die Express Version die im Jahre 2016 veröffentlicht wurde. Sie ist bis zu einer Arbeitsspeicherkapazität von 32 GB. SAP plante sie um sie in Umgebungen einzusetzen wo nur eingeschränkte Ressourcen zur Verfügung stehen. Im Gegensatz zum Microsoft SQL Server wird bei SAP Hana die komplette Datenbank im Hauptspeicher gehalten und durch spezielle Sicherungsverfahren werden laufend Änderungen auf der Festplatte repliziert, sodass kein Datenverlust entsteht (Recoverystrategien). Es handelt sich hier um eine spaltenorientierte Datenbank. Betrieben werden kann SAP Hana Express entweder fest auf einem Server, auf einer VM oder seit neustem auch in der Cloud (z.B. Amazon Webservices oder Microsoft Azure). Unterstützt werden Server mit SUSE oder Red Hat Linux. Will man die 32 GB Begrenzung umgehen, so ist dies gegen eine Gebühr gestaffelt möglich. Sie lässt sich mit dem Hana Studio oder über eine Weboberfläche bedienen, wobei wir das Hana Studio genutzt haben, was auf der Eclipse Plattform basiert. Für das Datenbankmanagment stehen viele Möglichkeiten zur Verfügung. Neben der Spaltenorientierung „Multi-Core“ und Parallelisierung, erweiterte Komprimmierungsverfahren, „Multi-Tenancy“, Möglichkeiten zur Datenmodellierung, Sicherheitskonzepte, sowie eine Offenheit. Daneben lässt sich noch Anwendungsentwicklung, „Advanced Analytical Processing“, sowie Datenintegration und Qualitätssicherung betreiben. Bei letzteren Daten Virtualisierung, ETL, Replikationen sowie eine Hadoop und Spark Integration. In der Express Version fehlen allerdings Funktionen wie „Multi-Tier-Storage“, Hochverfügbarkeit und Kotastrophenmanagement ebenso wie die Funktion zur Qualitäötssicherung und zur Datensynchronisation (remote). Positiv ist ganz klar, dass sie sowohl für OLTP wie auch OLAP geeignet ist. Verfügt der Server mehr als 32 GB Arbeitsspeicher, lässt sich SAP Hana Express natülich weiterhin anwenden, die Datenbankgröße ist lediglich auf 32 GB begrenzt.

4.2.3 Cassandra

Cassandra ist ein sehr einfaches verteiltes Datenbanksystem. Im Gegensatz zu konventionellen Datenbanksystemen, wird hier auf No SQL gesetzt. Dies steht für „Not only

SQL“ und ist ein alternatives, nicht-relationales Datenbankmodell. Verwendet wird es überwiegend in Key-Value Datenbanken, Graphendatenbanken, dokumentenorientierte Datenbanken oder wie hier spaltenorientierten Datenbanken. NoSQL ist gekennzeichnet durch eine horizontale Skalierbarkeit, das Vermeiden unnötiger Komplexität, eine hohe Performance sowie ein hoher Datendurchsatz. Weiterhin werden relationale Ansätze des Datenmappings vermieden und es erfolgt eine einfache Replikation der Datenbanken. Schwächen liegen im einen Mangel einer umfangreichen Dokumentation. Es ist außerdem keine universelle Sprache wie SQL, es gibt immer wieder unerwartete Verhaltensweisen und fehlenden Support.
Heute ist Cassandra die populärste spaltenorientierte NOSQL-Datenbank. Große Firmen wie Apple, Netflix und Twitter nutzen sie und setzen dabei auf die Stärke, wie eine einfache horizontale Skalierbarkeit, eine hohe Ausfallsicherheit, die Unterstützung mehrere Datacenter und die Speicherung großer Datenmengen.
Sie besitze eine IN-Memory Funktion. Dabei ist eine parallele Nutzung von Standardtabellen und IN-Memory Tabellen, sowie ein blitzschneller Lesezugriff auf die Daten möglich. Cassandra eignet sich dadurch optimal bei read-only Zugriffen und konstanten Daten.

4.2.4 Maria DB in Kombination mit Memcached

Maria DB ist ein kostenloses Datanbankmanagementsystem. Entstanden ist es aus einer Abspaltung von MySQL und wurde 2009 veröffentlicht. Im Gegensatz zu den anderen verwendeten Systemen handelt es sich hier um ein normales relationales Datenbanksystem. Es bietet also keine In-Momory Funktionalität und ebensowenig spaltenorientierung. Sie dient hier dem Vergleich in Kombination mit Memcached. Memcached wurde entwickelt von Brad Fitzpatrick (Danga Interactive / LiveJournal)und ist seit 15. Juni 2003 unter BSD-Lizenz verfügbar. Viele große Unternehmen setzen auch hier wieder auf die Software wie zum Beispiel Facebook, Youtube oder Reddit. Hierbei handelt es sich um eine Server Applikation die Speicher im Arbeitsspeicher zur Verfügung stellt. Sie lässt sich über das Netzwerk oder das Internet ansprechen und kann Daten aufnehmen und auch zurückgeben.
Der größte Anwendungsbereich des Systems sind Webanwendungen. Es wird vor allem genutzt um einen schnellen Zugriff auf im Cache abgelegte Daten zu erhalten. Hier erfolgt eine Lastverteilung von Festplattenzugriffen und Datenbankanfragen auf den RAM. Es funktioniert dabei so, dass häufig abgefragte Daten (wie die Ergebnisse einer SELECT Abfrage) direkt im Arbeitsspeicher festgehalten werden, was wiederum erheblich kür-

zerer Antwortzeiten ermöglicht, da keine unnötigen Festplatten- und DBMS Zugriffe durchgeführt werden müssen. Ziel der Software ist die Optimierung der Antwortzeiten von Webanwendungen.
Memcached stellt dabei mehrere Hashtabellen zur Verfügung (Key-Tabelle und Hashtabelle). Es ist von der Funktionsweise vergleichbar mit assoziativen Arrays in verschiedenen Programmiersprachen. Memcached ist eine Hash-Tabelle und speichert die Daten in einer Key / Value Abhängigkeit ab. Dabei werden die Schlüssel und die Werte als Zeichenketten abgespeichert. Da von verschiedenen Programmen auf Memcached zugegriffen werden kann, ist es diesen möglich die Daten vor dem absenden an Memcache zu serialisieren. Somit können auch andere Datentypen wie Ganzzahlen, Fließkommazahlen und Objekte abgespeichert werden. Die Speicherverwaltung erfolgt über eine Slab-Allocator. Dies bedeutet, dass viele kleine Speicherbereiche häufig reserviert und wieder freigegeben werden. Diese sind dabei maximal ein Megabyte groß. Definierte Clients existieren hier nicht, wobei die Programm Bibliotheken sehr großen Einfluss nehmen. Am weitesten verbreitet ist die für PHP (PECL Memchached).
Memcached-Server lassen sehr gut dezentralisieren. Dabei müssen diese als Cluster zusammengeschlossen werden, um so den Arbeitsspeicher mehrerer Web-Server effizienter nutzen zu können. Wobei zu beachten ist, das die einzelnen Knoten keinen gemeinsamen Master-Knoten über sich haben. Dies muss von den Client-Bibliotheken übernommen werden, die entsprechend die Verteilung von Daten an die Memcached-Server überwachen muss.

4.3 In-Memory Technologie im Vergleich zur konventionellen Datenhaltung

Die Speicherung in IN-Memory Datenbanken erfolgt spaltenorientiert was einer der größten Unterschiede im Vergleich zu konventionellen Datenbanken ist. Die Notwendigkeit zu diesem neuen Speicherverfahren resultiert aus dem extrem schnell wachsenden Berg an unstrukturierten Daten. Dies lässt sich vor allen im Internet beobachten wo es zu einem enormen Wachstum solcher Daten gekommen ist. In Spaltenorientierte Datenbanken (mitunter auch als Wide Column Stores bezeichnet)erfolgt die Speicherung der Daten in Spalten und nicht in Zeilen. Die Einträge in einer Spalte bestehen dabei jeweils aus dem Namen, den Daten und dem Zeitstempel. Untergliedert wird dabei in „Column Families“. Das sind die Spalten mit gleichartigen, sich ähnelnden Inhalt die eine Tabelle bilden. Ein weiterer großer Unterschied ist, dass es innerhalb einer „Column Family“ keinerlei logische Struktur existiert.

4.4 Kompressionsverfahren

Aus dem neuen Performance Bottleneck zwischen Hauptspeicher und Cache ergibt sich die Notwendigkeit Komprimierungsverfahren zu nutzen um eine schnellere Arbeit zu gewährleisten. Weiterhin sind diese notwendig, da trotz der aktuell relativ großen Arbeitsspeicherkapazitäten die Datenbanken oft noch größere Mengen an Daten enthalten, weshalb ohne Komprimierung eine Datenhaltung im Hauptspeicher nicht möglich wäre.

4.4.1 Kompressionsverfahren bei SAP HANA

Basis für die sehr effiziente Kompression stellt hier die spalten-orientierte Speicherung dar. Durch die Verringerung von Bits die zur Darstellung der Daten genutzt werden kann sowohl die Zugriffszeit, als auch der Speicherverbrauch verringert werden.
Grundlage stellt in der HANA das „Dictionary Encoding“ dar. Dabei werden Werte mit einer großen Länge (wie Texte) als Integer Wert gespeichert. Dabei ist es einfach zu verstehen und nicht schwer zu implementieren, was zu höheren Vorteilen in der Performance führt. Es arbeitet dabei spaltenweise. Hat die Tabelle zum Beispiel eine Spalte fName, so wird hier jedem Vornamen eine Positionsnummer zugeordnet. In dem sogenannten „Dictionary“ werden nun die Namen jeweils einzeln eingetragen und ebenso mit IDs versehen. Die beiden IDs werden dann einfach ein einem „Attribut Vector“ verknüpft. Je mehr Dopplungen von Namen in der Spalte fname sind, desto höher ist die Komprimierung. Angenommen jeder Vorname besteht aus 49 Byte und es gibt acht Milliarden Menschen auf der Erde, dann werden rund 365,1GB benötigt um alle zu speichern. Dahingegen braucht der „Attribut Vector“ 23 Bit pro Eintrag multipliziert mit den 8 Millionen Einträgen also 184 Milliarden Bit, was 21,4GByte entspricht. Das „Dictionary“ verbraucht bei 49 Byte pro Eintrag und 5 Millionen Vornamen nur 0,23GByte. Dividiert man nun die Menge ohne Kompression von 365,1GByte mit den 21,63GByte nach Kompression, sieht man, dass man der Speicherverbrauch um den Faktor 17 gesunken ist, was 6 Prozent des ursprünglichen Speicherbedarfs darstellt. Nutzt man dieses Prinzip bei den Geschlechtern,über ebenfalls 8 Milliarden Menschen, werden nur noch 0,93GB anstelle von 7,45 GB benötigt (Kompressionsfaktor 8). Der Kompressionsfaktor hängt dabei von der größe der Daten sowie der Tabelle ab, also der Spaltenkardinalität und der Tabellenkardinalität (Entropie - Maß für Informationsgehalt). Ein weiterer Geschwindigkeitsvorteil lässt sich durch sortierte „Dictionarys“ errreichen, da dann Binärsuche angewendet werden kann. Nachteil dieser Sortierung ist, das sie immer wieder neu durchgeführt werden muss, sobald ein Wert in dem „Dictionary“ angefügt wird. Dies

verursacht bei großen, sich ständig ändernden Datenmengen, aber wieder erhebliche Kosten.
Ein sehr großer Vorteil dieser Methode ist außerdem, dass alle Operationen auf den Daten der Tabelle in Attributvektoren mit Integer Werten ausgeführt werden. Diese kann der Prozessor wesentlich schneller verarbeiten als Zeichenketten. Der Prozessor muss so zwar eine Zusatzlast stemmen, aber da der Arbeitsspeicher hier der Bottleneck ist, ist dies vertretbar und fällt nicht sehr ins Gewicht. Da in der Regel die Ergebnismenge kleiner als die komplette Tabelle ist und bei vielen Operationen wie z.B. „Count“ nicht die echten Werte benötigt werden, ergibt sich auch kein Nachteil aus dieser Methode.
Da die Datenbanken großer Unternehmen oft mehrere Terabyte groß sind und die Kapazität der Arbeitsspeicher noch weit darunter ist, werden zusätzlich noch Kompressionsverfahren genutzt um ganze Datenbank im Hauptspeicher halten zu können. Dies hat auch den Vorteil, dass weniger Daten zwischen dem Prozessor und dem Speicher fließen müssen, was wiederum die Performance verbessert. Wichtig ist hier das Verhältnis von Kompression und den zusätzlich benötigten Operationen des Prozessors.
Heutigte Datenbanken enthalten oft „tonangebende“ Werte, wodurch derselbe Wert ohne Komprimierung sehr oft gespeichert werden muss. Um dieses Problem zu beheben und Speicher einsparen zu können wird das sogenannte „Prefix Encoding“ genutzt. Dafür müssen die Daten nach besagtem Wert sortiert werden und der Attributvektor mit diesem starten. Bei dem Vefahren wird nun nur eine Instanz des Wertes und die Häufigkeit dessen im „Attribut Vector“ gespeichert, anstelle des Wertes in doppelter und dreifacher Ausführung. Der Vektor besteht dann aus der Häufigkeit des Wertes, der ID aus dem „Dictionary“ und der ID's der folgenden Werte. Nimmt man nun zum Beipsiel die Tabelle mit der Bervölkerungsmenge nach Ländern sortiert, so befand sich der Wert für China ungefähr 1,4 Milliarden mal in dem „Attribut Vector“. Daraus ergibt sich ein Speicherbedarf von 1,4 Milliarden mal 8 Bit. Mit „prefix Encoding“ lässt sich dies auf nur 39 Bit verringern (8 Bit für das einmalige Speichern und 31 Bit für die Häufigkeit). Darauf ergibt sich eine Ersparnis von immerhin 17 Prozent (1,3 GByte in diesem Beispiel). Zusätzlich dazu ermöglicht es direkten Zugriff über „row number claculation“ (Dabei determiniert die Datenbank, dass nur die bestimmte Anzahl an Zeilennummern benötigt (im Beispiel 1 - 1,4 Milliarden), was zu schnelleren Ergebnissen führt.
Eine weitere Möglichkeit der Kompression bietet das „Run Length Encoding“. Es funktionier am besten, wenn der „Attribut Vector“ einige sich unterscheidende Werte enthält, die sich sehr oft wiederholen. Zur Erreichung besonders guter Ergebnisse, muss die Spalte (des Vektors) sortiert werden. Die beieinaderliegenden gleichen Werte werden nun wie-

der zu einem zusammengefasst mit entweder der Häufigkeit oder der Startposition als Abstände. Letzteres hat dabei den Vorteil, dass ein schnellerer Zugriff ermöglicht wird, da direkt die Adresse eines Wertes gelesen werden kann, anstatt das diese erst berechnet werden muss. Bezogen auf das obige Beispiel lässt sich bei der Länderspalte wieder sehr viel einsparen. Dafür werden zwei Vektoren benötigt. Einer mit allen Ländern (200 Einträge) und einer mit den Startpositionen der einzelnen Länder. Acht Bit benötigt jeder Eintrag der Länder und 33 Bit jede Startposition (Log_2 8.000.000.000) plus ein Extra Feld (33 Bit) am Ende, welches die Häufigkeit des letzten Wertes beinhaltet (da diesem ja kein neuer Wert folgt). Alles in allem ergibt sich eine Größe des Vektors von nur rund 1 KB gegenüber 7,45 GB [200*(33Bit + 8 Bit)+ 33Bit]. Würde man die Häufigkeit anstelle der Startposition nehmen, könnte man zwar nochmal 33 Bit einsparen, aber es wäre kein direkter Zugriff mehr möglich, was zu einer viel höheren Antwortzeit führen würde und daher keine gute Option für IN Memory Datenbanken ist.
Cluster Encoding ist ein weiteres Interessantes KOmpressionsverfahren. Hier wird der „Attribut Vector“ in Blöcke der gleichen Größe unterteilt (oft 1024 Werte). Befinden sich in einem Block nur gleiche Einträge, so werden diese zu einem zusammengefasst. Ist dies nicht gegeben, bleibt der Block unverändert. Nun wird noch ein zweiter Vektor benötigt in dem steht, welcher Block durch einen einzelnen Wert ersetzt wurde (wenn Block unverändert 0, wenn zusammengefasst dann 1). Um hier auf die einzelnen Werte zuzugreifen, wird der Index mit Hilfe von Integer Division berechnet (Zeilennummer durch Größe des Blockes). Der wohl größte Nachteil ist hier, dass kein direkter Zugriff auf die Werte möglich ist. Es lässt sich aber eine große Menge Speicher einsparen. In Bezug auf das obige Beispiel, benötigt man bei der Spalte mit den Städten nur noch rund 2,4 GB, anstatt 18,6 GB (87 Prozent Kompressionsrate unter der Annahme, dass es 1 Million Städte gibt).
Bei „Indirect Encoding“ wird der Vektor ebenso in eine Vielzahl gleich großer Blöcke unterteilt (Größe meist wieder 1024). Als sehr effektiv erweist sich diese Methode, wenn ein Block einige sich unterscheidende Werte enthält, was oft gegeben ist, wenn eine Abhängigkeit zwischen zwei Spalten existiert und die Tabelle nach einer sortiert wurde (z.B. nach Land sortiert, wenn man die Spalte Vorname betrachtet wird). Hier wird neben dem globalen „Dictionary“ noch ein lokales benötigt. Wenn man nun die Spalte Vorname betrachtet und annimmt, dass es im Schnitt 200 verschiedene Vornamen in einem Block pro Land gibt, wird in dem lokalen Wörterbuch jedem Wert für einem Namen ein Wert von 0 bis 199 zugeordnet, dieser dann in einem Vektor gespeichert. Die Einsparung an Speicherplatz resultiert hier daraus, dass bei nur 200 verschiedenen Werten nur noch acht

statt 23 Bit pro Eintrag belegt werden (Log_2(5 Millionen verschiedenen Namen Global) ergibt 23 Bit pro Eintrag ohne Kompression), unter der Voraussetzung das vorher nach den Ländern sortiert wurde. Hier erreicht man z.B. immerhin eine Kompressionsrate von 44 Prozent (nur rund 11,8 GB und nicht 21,4 GB) und ein direkter Zugriff ist im Gegensatz zum „Cluster Encoding“ weiterhin möglich.
Neben der Verringerung der Größe des „Attribut Vectors“ gibt es noch die Möglichkeit das „Dictionary“ zu optimieren. Das wird mit „Delta Encoding“ erreicht. Ist es nun alphanumerisch sortiert gibt es oft viele Werte mit der derselben Vorsilbe. Diese Tatsache macht man sich hier zu Nutze indem gemeinsame Vorsilben nur ein Mal gespeichert werden. Dabei wird wieder Blockweise gearbeitet (oft mit 16 Strings pro Block). es speichert die Länge der ersten Zeichenkette, gefolgt von dieser am Anfang eines Blocks. Bei den darauffolgenden Werten wird nun die Anzahl an Zeichen die gleich dem Vorgänger sind gespeichtert plus der Anzahl an folgenden, gefolgt von eben diesen Zeichen. Nimmt man zum Beispiel die Städte im „Dictionary“, der erste Wert des Blocks ist Aach gefolgt von Aachen, dann wird im komprimierten Vektor die Länge 4 und die Zeichen von „Aach“ festgehalten. Für „Aachen“ wird die Länge der gleichen Teils „Aach“ mit der 4 ersetzt, dann die Anzahl der folgenden und die Zeichen an sich gespeichert „en“. Dies lässt sich dann immer weiter so aufbauen, was im optimalen Fall immer mehr Einsparung mit sich bringt. Im hier brtrachteten Beispiel mit den Städten wird eine Kompressionsrate von 90 Prozent erreicht (rund 5,4 MB gegenüber 46,7 MB).
Bei al den Vorteilen durch Kompression sind dem Ganzen auch Grenzen gesetzt. So werden für die meisten Verfahrem sortierte Tabellen benötigt um maximale Erfolge zu erzielen, aber es kann in einer Tabelle immer nur nach einer Spalte sortiert werden. Wenn kein direkter Zugriff möglich ist, ergibt sich außerdem ein großer Nachteil in der Performance.

4.4.2 Kompressionsverfahren bei MS SQL Server

4.4.3 Kompressionsverfahren bei Cassandra

Cassandra beitet drei verschiedene Verfahren zur Komprimierung: LZ4, Snappy oder Deflate. Mit diesen ist ein Performance-gewinn von bis zu 10 Prozent möglich.

4.4.4 Kompressionsverfahren bei Maria DB mit Memcached

Da Memcached eine reine Speicherfunktion erfüllt, gibt es im Programm selber keine Methoden zur Komprimierung. Jedoch bieten viele der Bibliotheken eigene Kompressionsverfahren an, die auf die Daten angewendet und danach an den Memcached-Server übermittelt werden. Die PHP-Extension stellt zum Beispiel eine auf dem Deflate-Algorithmus basierende Kompressionsmethode zur Verfügung.

4.5 Parallele Verarbeitung

Unterteilt Arbeitsschritte um parallel daran zu arbeiten Verteilt die Daten auf mehrere Serverblades um Lesezugriff zu ermöglichen Erhöht die Ausfallsicherheit durch Standby Blades

Bild parallele Verarbeitung einfügen

4.6 Hochverfügbarkeit

Storage Replication (Spiegelung der Speicherarchitektur) Host Auto-Failure (Data- und Log-Volumes werden von einem Hot Standby-System übernommen) SAP HANA System Replication (Permanente Replikation der Daten auf Sekundäres System)

Wie schützt sich die InMemory Datenbank vor z.B. Stromausfällen? Data- und Log-Volumes werden auf der Festplatte gespeichert

5 Installation der Systeme

5.1 Aufsetzen des MS SQL Server

5.2 Aufsetzen der SAP Hana Express

5.3 Aufsetzen von Cassandra

5.4 Aufsetzen von Maria DB in Kombination mit Memcached

Beschaffung der nötigen Software

1. Download des XAMPP Paketes:
   https://www.apachefriends.org/de/download_success.html
2. Download PHP-Extension:
   https://github.com/nono303/PHP7-memcache-dll/blob/master/
   vc14/x86/ts/php-7.0.x_memcache.dll
3. Download Memcached: http://downloads.northscale.com/memcached-win32-1.4.4-14.zip

5.4.1 Installationsvorgang

XAMPP installieren

1. XAMPP wird von www.apachefriends.org heruntergeladen und installiert. Ein Installationsprogramm leitet durch den Installationsprozess und weist bereits auf Typische Probleme mit anderen eventuell installierten Programmen hin. Diese müssen vor der Installation beendet werden. Bild 1 aus Doku!
2. Wichtig ist zu beachten das der Apache-Server auf den Standartports 80 (HTTP) und 443 (SSL) eingerichtet ist und somit keine anderen Prozesse auf dem Rechner laufen dürfen die diese Ports blockieren. Bild 2 aus Doku!
3. Die Module, wie Apache-Server oder MariaDB können über das im Installationsordner befindliche „XAMPP Control Panel.exe“ einzeln gestartet und administriert werden. Weitere Vorraussetzung ist das der Standartbrowser des Host-Rechners den Zugriff auf die Lokale Domain / IP zulässt. („localhost / 127.0.0.1 “) Anschließend ist der Apache-Server nach dem starten über einen beliebigen Browser über die "localhost"Domain/IP erreichbar.

Maria DB einrichten

1. Wird in der XAMPP Installation mitgeliefert und ist über den Browser administrierbar (PHPmyAdmin) – http://localhost/phpmyadmin/
2. Datenbankstruktur der Testdatenbank wurde über die GUI implementiert

```
        /* Erstellung der 4 Tabellen mit Fremdschlüsseln */
        /*Tabelle Kunden anlegen*/
CREATE TABLE Kunden
(Kunnr int Primary Key,
Name varchar(50),
Vorname varchar(50),
PLZ int,
Ort varchar(50),
Landkz varchar(50),
Land varchar(50),
Bundesland varchar(50),
Region varchar(50),
Debitornr int,
Kreditorennr int,
BLZ int,
IBAN int,
Kreditinstitut varchar(50),
Beziehungslvl int,
Werber varchar(50)
)


/*Tabelle Bestellungen anlegen*/
CREATE TABLE Bestellung (
Bestellnr int Primary Key,
Datum date,
Pos int,
Artnr int,
Artbez varchar(50),
Preis int,
```

```
Mwst int,
Menge int,
RabattMenge int,
BetragGesamt int,
RabattKunde int,
MwstGesamt int,
Kunnr int,
Eilauftrag int,
Foreign Key (Kunnr) References Kunden(Kunnr)
)
/*Tabelle Lieferdienst anlegen*/
CREATE TABLE Lieferdienst (
Bestellnr int,
Liefernr int Primary Key ,
Fahrzeugtyp varchar(50),
Unternehmen varchar(50),
DauerDurchschnitt int,
Preis int,
Mwst int,
MaxLieferMenge int,
Sitz varchar(50),
MitarbeiterAnzahl int,
Abholstationen int,
Lieferstart int,
Fahrzeuganzahl int,
FOREIGN KEY (Bestellnr) REFERENCES Bestellung(Bestellnr)
)

/*Tabelle Standorte anlegen*/
CREATE TABLE Standorte (
StandortId int Primary Key,
Liefernr int,
Stadt varchar(50),
Landkz varchar(50),
Land varchar(50),
```

```
Bundesland varchar(50),
Region varchar(50),
Leitername varchar(50),
Leitervorname varchar(50),
Mitarbeiteranzahl int,
Fuhrparkgröße int,
Gelaendegroeße int,
FOREIGN KEY (Liefernr ) REFERENCES Lieferdienst(Liefernr )
```

3. Zuerst wurde versucht die Daten Tebenfalls über die PHPmyAdmin GUI in die Datenbank zu laden. Jedoch begrenzte der Apache-Server die Anzahl der Daten und die Laufzeit der Skripte so stark, das schlussendlich über die mitgelieferte MariaDB Console ein Bulkload der CSV-Dateien erfolgen musste. Diese nahm in etwa 2 Stunden in anspruch.

```
      /*Bulkload CSV in die Datenbank*/
/*Die Reihenfolge der Füllung ist wegen der FK zu beachten*/

/*1.) Tabelle Kunde laden*/
LOAD DATA LOCAL INFILE
'C:/Users/stud_admin/Desktop/Gäfe/Tabellen/Kunde.csv'
INTO TABLE kunden FIELDS TERMINATED BY ',' ;
/*Feldergrenze muss angegeben werden, Rest kann Default bleiben*/

/*2.) Tabelle Bestellungen laden*/
LOAD DATA LOCAL INFILE
'C:/Users/stud_admin/Desktop/Gäfe/Tabellen/Bestellungen.csv'
INTO TABLE bestellung FIELDS TERMINATED BY ',' ;

/*3.) Tabelle Lieferdienst laden*/
LOAD DATA LOCAL INFILE
'C:/Users/stud_admin/Desktop/Gäfe/Tabellen/Lieferdienst.csv'
INTO TABLE lieferdienst FIELDS TERMINATED BY ',' ;

/*4.) Tabelle Standorte laden*/
```

```
LOAD DATA LOCAL INFILE
'C:/Users/stud_admin/Desktop/Gäfe/Tabellen/Standorte.csv'
INTO TABLE standorte FIELDS TERMINATED BY ',' ;
```

Memcache-dll installieren

1. Anschließend muss die .dll in den

   ```
   php\ext
   ```

   Ordner der XAMPP Installation eingefügt werden und über das XAMPP Control Panel in die php.ini in die entsprechende Zeile das Ëxtension=memcached.dll eingefügt werden. Bild Bild Starten der "php.exeïm /php Unterordner der XAMPP Installation um die Extension zu verifizieren und eventuelle Fehlermeldungen zu erhalten. Neustarten des Apache-Servers falls bereits gestartet. Memcached Server sind per default auf dem Port 11211 erreichbar.

2. PHP-Funktionen der Erweiterung

   ```
   $memcached = new Memchached();
$memcached->connect( string $host [, int $port [, int $timeout ]] ) ;

$memcached->set('key', 'value');
$memcached->get('key');
   ```

Memcached Installation

1. Da Memcached ursprünglich für Linux entwickelt wurde sind die meisten Distributionen natürlich für dieses Betriebssystem ausgelegt. Allerdings werden auch Windowsversionen für 32-Bit und 64-Bit Systeme angeboten. Im nachfolgenden wurde die 32-Bit Variante ausgewählt, da unsere Installierte PHP-Engine ebenfalls auf der 32-Bit Architektur basierte.

2. Die Installation eines Memcache Servers ist sehr einfach. Das Programm wurde von http://downloads.northscale.com/memcached-1.4.5-x86.zip her-untergeladen und Installiert.

3. Dazu einfach die heruntergeladene Ausführbare Datei ausführen und das Programm wird in dem gewählten Ordner installiert

6 Anwendungsbeispiel und Vergleichender Performancetest

6.1 Anwendungsbeispiel

6.2 Erzeugung der Datengrundlage

6.3 Einrichten der Datenbank mit den Testdaten

6.4 Performancevergleich

7 Quellen

Einleitung:
- https://de.wikipedia.org/wiki/Microsoft_SQL_Server (15.01.2018, 14.15 Uhr)
- https://www.red-gate.com/simple-talk/sql/learn-sql-server/
introducing-sql-server-in-memory-oltp/ (15.01.2018, 15.33 Uhr)
Kompression:
-(07.01.2018 14.51)