
Fachbereich 12 Informatik und Mathematik
Institut für Informatik

**Diplomarbeit**

# Green IT und Datenbanken

Tobias Gontermann, Adam Hermann, Marten Rosselli
Studiengang: Diplom-Informatik

Frankfurt am Main
16. Juni 2011

Betreuer: Prof. Dott.-Ing. Roberto V. Zicari

Gesetzt am 16. Juni 2011 um 17:59 Uhr mit LaTeX.

# Danksagung

Diese Diplomarbeit entstand am Lehrstuhl für Datenbanken und Informationssysteme der Johann-Wolfgang-Goethe-Universität Frankfurt am Main unter der Leitung von Herrn Prof. Dott.-Ing. Roberto V. Zicari. Besonders möchten wir uns bei Herrn Zicari für die Möglichkeit bedanken, in seiner Arbeitsgruppe diese Diplomarbeit anzufertigen. Nicht zuletzt durch die freundliche und engagierte Betreuung hat uns diese Arbeit viel Freude bereitet. Außerdem gilt unser Dank Andreas Beckmann, der uns unkompliziert mit Rat und Tat bei Fragen zu Strommessungen zur Seite stand. Auch gilt unser Dank Frau Brigitte Hermann und Herrn Markus Michalek vom Hochschulrechenzentrum der Universität Frankfurt am Main die uns für die Dauer der Diplomarbeit einen Server bereitstellten.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

Der Energieverbrauch in Rechenzentren steigt jährlich. 2008 betrug der Energieverbrauch der etwa 50.000 Serverräume und Rechenzentren in Deutschland 10,1 TWh (Terrawattstunden). Dies entspricht der jährlichen Energieproduktion von 4 mittelgroßen Kohlekraftwerken und etwa 1,8 Prozent des Gesamtenergieverbrauchs in Deutschland. Die damit einhergehenden $CO_2$-Emissionen betrugen knapp 6,4 Mio. Tonnen.

Während es in Deutschland bis jetzt noch keine konkreten Vorschläge gibt, Unternehmen zur Einsparung von Strom zum Zwecke des Klimaschutzes zu verpflichten, und weitestgehend auf eine Selbstverpflichtung gesetzt wird, ist in Großbritannien Anfang 2010 das europaweit erste Gesetz in Kraft getreten, das Unternehmen dazu verpflichtet, sich aktiv mit der $CO_2$-Reduktion ihrer Rechenzentren auseinanderzusetzen.

Unternehmen werden mit Bonuszahlungen auf der einen Seite und Strafgebühren auf der anderen Seite dazu „motiviert“, Energie zu sparen. Es ist vermutlich also nur noch eine Frage der Zeit, bis auch in Deutschland ähnliche Gesetze erlassen werden. Der Gedanke und die Umsetzung desselben, nämlich in der IT-Industrie Energie zu sparen und die Umwelt zu schonen, ist nicht neu. Schon vor über zehn Jahren wurde zum Beispiel das *Energy Star*-Logo eingeführt, um elektrischen Geräten ihre stromsparenden Eigenschaften zu bescheinigen.

Die Bemühungen, Energie in der IT-Branche zu sparen, werden unter dem Begriff *Green IT* zusammengefasst. Inzwischen gibt es viele Bemühungen seitens der Hardwarehersteller, energieeffiziente Hardware zu produzieren. Es gibt Netzteile mit hohem Wirkungsgrad, energieeffiziente Prozessoren und sogar stromsparende Arbeitsspeicher. Aber auch im Bereich der Software gibt es verschiedene Ansätze, Energie zu sparen. Hier wird zumeist der Ansatz verfolgt, über Energiesparpläne von zum Beispiel Betriebssystemen nicht benötigte Hardware zu deaktivieren oder in einen Energiesparmodus zu versetzen.

## 1.2 Ziele

Aus ökonomischen und ökologischen Gründen ist eine energieeffiziente Nutzung der Ressourcen in Rechenzentren besonders wichtig. In Rechenzentren werden typischer-

weise große Datenmengen in Datenbanken verwaltet. In dieser Diplomarbeit wird untersucht, wie eine Datenbank energiesparend betrieben werden kann. Durch Untersuchungen am Beispiel der Datenbank *Caché* der Firma *Intersystems* werden verschiedenen Fragen beantwortet. Kann die Hardwarenutzung einer Datenbank hinsichtlich der Energieeffizienz optimiert werden? Welchen Einfluss haben der Prozessor, die Festplatte (HDD) oder Flash-Speicher und welche Möglichkeiten gibt es, den Energieverbrauch einer Datenbank zu reduzieren? Welchen Einfluss hat die Arbeitsspeicher- und die Datenbankgröße auf den Energieverbrauch? Kann durch Indexstrukturen Energie gespart werden? Die Antworten auf diese Fragen erlauben es einem Anwender, die Datenbank *Caché* energieeffizienter zu betreiben.

## 1.3 Überblick

In **Kapitel 2** werden allgemeine Grundlagen, den Kontext dieser Arbeit betreffend erläutert. Es wird eine Definition für den Begriff Green-IT angegeben und erklärt, was sich dahinter verbirgt. Im Besonderen werden dort Maßnahmen vorgestellt, mit denen in Rechenzentren bereits jetzt Energie gespart wird. Außerdem werden Angaben darüber gemacht, welche Optimierungsmethoden und Ansätze sich grundsätzlich für die Einsparung von Energie bei Datenbanken eignen. Ein weiterer Teil von Kapitel 2 vermittelt einen kurzen Überblick über die Datenbank *Caché* von *Intersystems*, die für alle unsere Untersuchungen verwendet wurde, und beschreibt, welchen Workload wir für unsere Testreihen genutzt haben.

Im **Kapitel 3** werden die für die Messungen verwendete Hardware, der Versuchsaufbau und die von uns entwickelte Software zum Durchführen der Messungen vorgestellt. Es wird erläutert, welche Leistungsindikatoren des Servers wir für unsere Versuche messen und welches Datenbankschema verwendet wird.

Das **Kapitel 4** beschäftigt sich mit der Einrichtung der Testumgebung und beschreibt das Programm, mit dem unsere Messungen durchgeführt wurden, detaillierter. Außerdem wird dort beschrieben, wie besondere Einstellungen, die für einige Tests vorgenommen wurden und nicht öffentlich dokumentiert sind, realisiert wurden.

In **Kapitel 5** wird untersucht, welchen Einfluss die Arbeitsspeicher- und Datenbankgröße auf den Stromverbrauch hat. Zusätzlich wird untersucht, ob sich gängige SQL-Befehle in logische Gruppen unterteilen lassen, welche jeweils unterschiedlich stark auf die Parameter Arbeitsspeicher- und Datenbankgröße reagieren.

Das **Kapitel 6** unterteilt sich, neben einem Unterabschnitt mit einigen Grundlagen zum verwendeten Datenmodell und Besonderheiten von *Caché* in Bezug auf die Implementierung des TPC-H-Schemas, in drei Teile. Alle Tests in diesem Kapitel wurden mit den 22 TPC-H-Abfragen durchgeführt. Der Stromverbrauch nach den einzelnen Optimierungen wird mit dem Stromverbrauch verglichen, den die Datenbank mit dem vom Transaction Processing Performance Council für den TPC-H-Benchmark

vorgegebenen Datenmodell auf einer gewöhnlichen magnetischen Festplatte ohne Optimierungen verbraucht hat.
Der erste Teil beschäftigt sich mit der Möglichkeit, Strom durch den Einsatz von Indizes zu sparen. Hier wird das Datenmodell untersucht und geeignete Indizes werden eingerichtet. Anschließend wird der Stromverbrauch, der so modifizierten Datenbank gemessen und der Grund für die Ersparnis analysiert.
Der zweite Teil dieses Kapitels beschäftigt sich mit dem Einsatz von Solid State Discs (SSD) anstelle von herkömmlichen Festplatten. Hier wird untersucht, welches Energiesparpotenzial eine SSD besitzt und wie dieses Potenzial durch Softwareoptimierungen weiter gesteigert werden kann.
Im dritten Teil dieses Abschnittes wird untersucht, ob durch Spannungs- und Frequenzskalierung des Prozessors der Stromverbrauch gesenkt werden kann, ohne dass die gesparte Energie durch eine zu hohe Laufzeit wieder eingebüßt wird.

Das **Kapitel 7** fasst alle von uns untersuchten Maßnahmen zur Senkung des Stromverbrauchs übersichtlich zusammen.

# 2 Grundlagen

## 2.1 Green-IT

### 2.1.1 Begriffsdefinition

Der Begriff Green IT fasst den energiesparenden Umgang mit IT-Ressourcen zusammen. Als IT-Ressourcen können in diesem Zusammenhang Hardware, Software, Services und Geschäfts- beziehungsweise IT-Strategien verstanden werden. Gadatsch [Gad10] beschreibt den Begriff wie folgt:

> Unter Green IT wird der Einsatz energiesparender Technologien (Hardware, Software, Kühlung), organisatorischer und technischer Konzepte (Anordnung technischer Einrichtungen und Geräte, Gebäudekonstruktionen) und Regelungen (Arbeitsanweisungen) verstanden [Gad10].

Der Begriff Green IT ist demnach ein vielfältiger Begriff. Vom betriebswirtschaftlichen Standpunkt betrachtet sind organisatorische Konzepte, Regelungen und Arbeitsanweisungen von Interesse. Für die Ingenieurwissenschaften spielen Gebäudekonstruktionen und Kühlsysteme eine Rolle. Für die Informatik sind energiesparende Technologien im Bereich der Hardware und unter dem Aspekt des Energieverbrauchs effizienter Software von besonderem Interesse.

### 2.1.2 Entwicklung von Umweltsiegeln in der IT-Branche

Schon seit den frühen 90er-Jahren gibt es Bestrebungen im Sektor der Informationstechnik auf Umweltfreundlichkeit und Energieeinsparungen zu achten. So existieren seit 1992 die Produktbezeichnung *Energy Star*, welche elektrischen Geräten bescheinigt, Stromsparkriterien der Umweltschutzbehörden EPA (Environmental Protection Agency) und des *US-Department of Energy* einzuhalten. Im Jahr 2003 wurde diese Produktbezeichnung schließlich durch eine EU-Verordnung auch in Europa übernommen.

Im Jahre 1992 wurde ebenso das erste TCO-Prüfsiegel (TCO) eingeführt. Das Siegel wird von der Organisation TCO Development vergeben. Überprüft wurden ursprünglich neben den Strahlenemissionen von Monitoren die Umweltverträglichkeit und der Stromverbrauch von Computer-Monitoren. Der Geltungsbereich des Siegels wurde in den Folgejahren regelmäßig erweitert. So kamen neben Kriterien für Monitore auch Kriterien für Drucker, Tastaturen, Laptop- und Desktop-Computer hinzu.
In der Folge wurde eine ganze Reihe weiterer Initiativen und energiespezifischer

Umwelt-Gütesiegel eingeführt. Dazu zählt beispielsweise das Siegel *80 Plus* [1] aus dem Jahr 2004 welches für Netzteile einen Mindestwirkungsgrad von 80 % vorsieht.

Die im Jahr 2009 neugefasste Energy Star-Spezifikation 5.0 [EPA] legt beispielsweise für den Computer-Zustand *Off* und *Sleep* einen maximal zulässigen Stromverbrauch von 2 Watt fest. Der typische maximale Energieverbrauch (TEC, Typical Energy Consumption) eines Desktop-Computers der Kategorie A im Betrieb ist beispielsweise 148 Watt. Wird dieser Wert überschritten, so bleibt dem Hersteller das *Energy Star*-Gütesiegel verwehrt. Für Spielekonsolen, Server, Desktop-Computer unterschiedlicher Leistungsklassen und viele weitere Geräteklassen existieren eigene TEC-Werte.

### 2.1.3 Bedeutung von Green IT für Rechenzentren

Besonders in Rechenzentren gibt es Bestrebungen, den Stromverbrauch zu senken, da dort der Stromverbrauch und das Einsparungspotenzial außerordentlich hoch sind. Eine Untersuchung des Borderstep Instituts für Innovation und Nachhaltigkeit im Jahr 2008 [FC08] im Auftrag des Bundesministeriums für Umwelt, Naturschutz und Reaktorsicherheit (BMU) gibt Auskunft über den Stromverbrauch und die Stromkosten von Rechenzentren. Es wurde für die Jahre 2000 bis 2005 errechnet, dass der weltweite Stromverbrauch durch Server von 58 Milliarden KWh im Jahr 2000 auf 123 Milliarden KWh im Jahr 2005 angestiegen ist. Das entspricht rund 1 % des weltweiten Gesamtstromverbrauchs. Bei dieser Berechnung wurde der Stromverbrauch, welcher für die Kühlung der Server und für die Aufrechterhaltung einer unterbrechungsfreien Stromversorgung (USV) benötigt wird, mit einbezogen. Eine weitere Studie aus dem Jahr 2008 besagt, dass in Rechenzentren die Stromkosten, nach den Kosten für die Hardware und die Kühlung, den drittgrößten Ausgabeposten bilden [Ham08].

In der Untersuchung des Borderstep Instituts [FC08] wird davon ausgegangen, dass der Stromverbrauch für Rechenzentren in den kommenden Jahren weltweit weiterhin erheblich zunehmen wird. Allein in Deutschland ist die Anzahl der Server in Deutschland in den letzten zehn Jahren rasant angestiegen. Der Stromverbrauch der rund 50.000 deutschen Serverräume und Rechenzentren 2008 lag bei 10,1 TWh, die Stromkosten bei rund 1,1 Mrd. € [2]. Dies entspricht einem Anteil am Gesamtstromverbrauch in Deutschland von rund 1,8 %. Bei völliger Ausschöpfung der Möglichkeiten, welche Green IT bietet, lässt sich laut dieser Untersuchung mehr als die Hälfte des Stromverbrauchs der Rechenzentren in Deutschland einsparen. Dies verdeutlicht Abbildung 2.1.
Die Effektivität der Kühlsysteme von Rechenzentren ist ein besonderes Thema. Auf einem Rechenzentrum-Expertentreffen in Zürich 2010 [Kla10] wurde festgestellt, dass die Kosten für die Kühlung in Rechenzentren erstaunlich hoch sind. Der Effizienzgrad eines Rechenzentrums wird heute mit dem sogenannten PUE-Wert (*Power Usage Effectiveness*) gemessen. Der PUE-Wert setzt den insgesamt im Rechenzentrum ver-

[1] 2006 wurden schließlich die *Energy Star*-Richtlinien um die *80 Plus*-Kriterien erweitert.
[2] Bei der Berechnung wurde für das Jahr 2008 von einem Strompreis von 0,11 €/KWh ausgegangen.

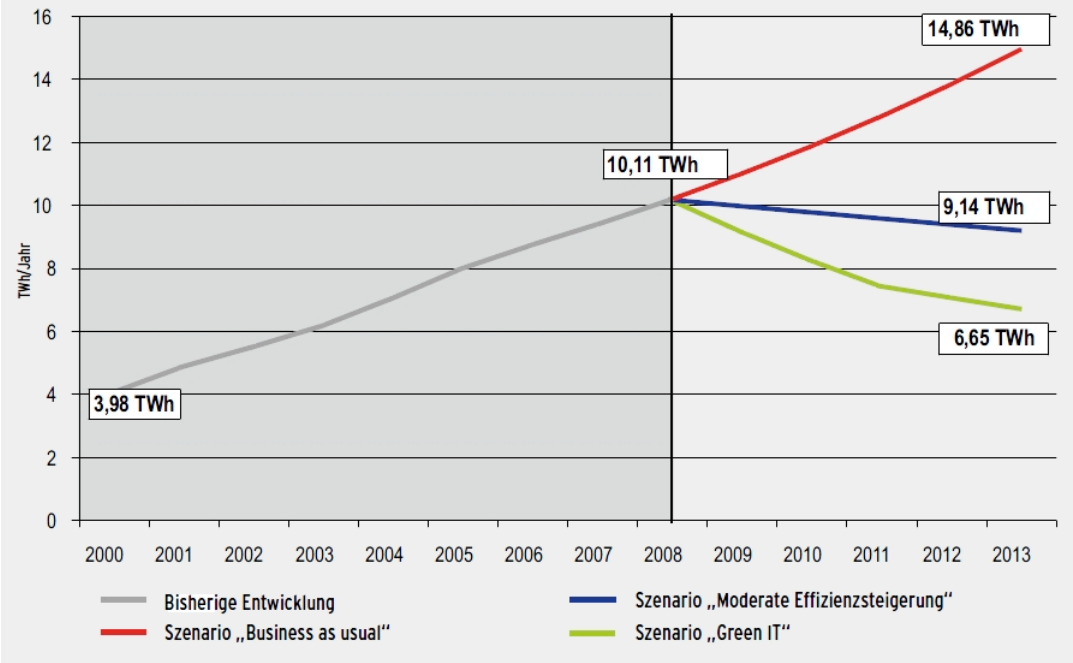



Abbildung 2.1: Stromverbrauch von Servern und Rechenzentren in Deutschland [FC08].

brauchten Strom ins Verhältnis zum Stromverbrauch der Server selbst. Der Wert für den Gesamtstromverbrauch eines Rechenzentrums bezieht also den Stromverbrauch für zum Beispiel die Kühlung und Beleuchtung mit ein. Ältere Rechenzentren haben einen PUE-Wert von 2,5 bis 4,0, während hingegen neue Rechenzentren einen PUE-Wert zwischen 1,4 und 2,0 aufweisen. Es ist demnach gerade bei älteren Rechenzentren so, dass mehr Strom für die Kühlung der Server und die Beleuchtung verbraucht wird als für den Betrieb der Server selbst.

Umso wichtiger ist es, den Energiebedarf für den Betrieb der Server zu senken, da bei einer geringeren Leistungsaufnahme auch weniger Wärme entsteht. Neben der reinen Verbesserung der Kühlmechanismen sind ebenso eine geringere Leistungsaufnahme und eine energieeffizientere Nutzung der Server wünschenswert, da dadurch die Prozessoren weniger Kühlung benötigen würden. Aufgrund des hohen Stromverbrauchs ist das Interesse bei Betreibern von Rechenzentren hoch, den Stromverbrauch aus betriebswirtschaftlichen Gründen heraus und der Umwelt zuliebe zu senken.

### 2.1.4 Maßnahmen in Rechenzentren und in Unternehmen

In einer Studie von Kolbe et al. [KZSE10] wurden Unternehmen befragt, welche Maßnahmen bisher in ihren Rechenzentren umgesetzt wurden. Die Ergebnisse der Umfrage werden in diesem Abschnitt kurz dargestellt. Abbildung 2.2 zeigt die Implementierungsrate von Maßnahmen bezüglich Green IT in Rechenzentren.

Servervirtualisierung wurde von den befragten Unternehmen am häufigsten genannt und diese Maßnahme wird bei 78 % aller Unternehmen eingesetzt. Kühl- und Klimaoptimierungen, Speichervirtualisierung, Netzwerk- und Elektrikoptimierung und kontinuierliche Kontrolle des Stromverbrauchs sind Maßnahmen, die von 50 bis 70 % der Unternehmen durchgeführt wurden. Aktives Abschalten von Systemen, Hinzuholen von externen Beratern und Grid-Computing wurden hingegen nur von 10 bis 30 % durchgeführt. Überraschend selten, mit nahezu 0 %, wurde von Unternehmen das Cloud-Computing als Maßnahme genannt. Reine Softwaremaßnahmen wie Server- und Speichervirtualisierungen nehmen eine vordere Rolle bei den Maßnahmen ein.

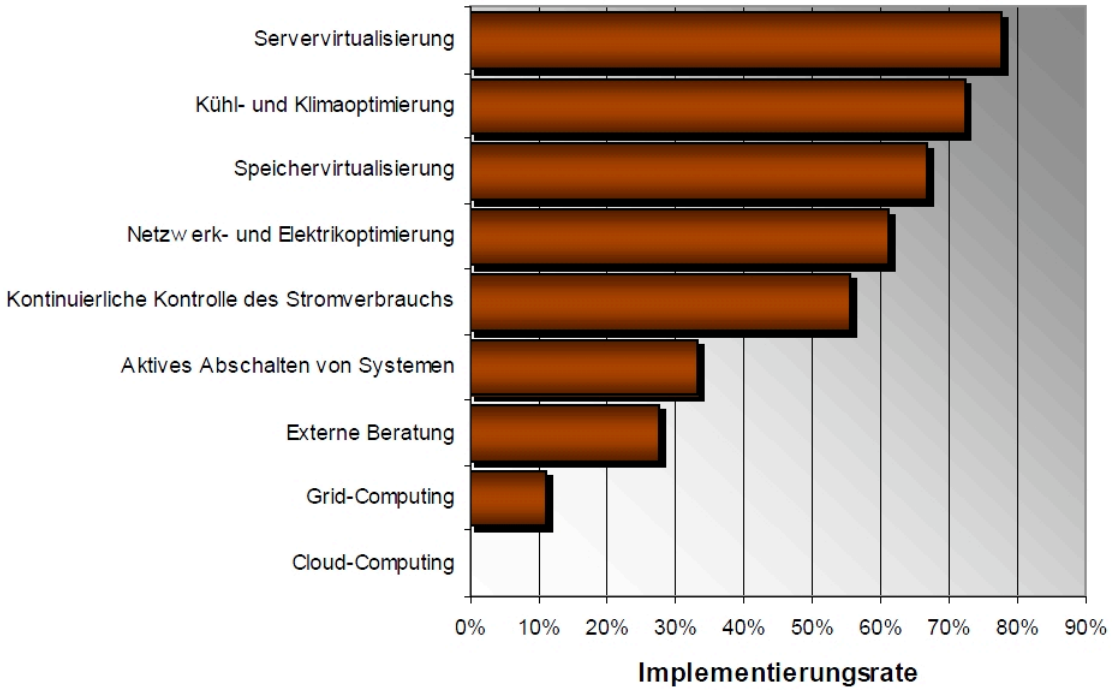



Abbildung 2.2: Implementierungsrate von Maßnahmen bezüglich Green IT in Rechenzentren [KZSE10].

In Büroumgebungen sind stromsparende Maßnahmen laut Kolbe et al. [KZSE10] bisher weniger stark verbreitet als in Rechenzentren. Videokonferenzsysteme und Umwelttraining der Mitarbeiter werden laut der Umfrage in 40 bis 50 % der befragten Unternehmen umgesetzt und nehmen damit die vorderen Plätze im Maßnahmenkatalog ein. Konzepte wie Desktopvirtualisierung, Wake-On-Lan, Shared-Desk-Konzept und überwiegende Nutzung von Notebooks sind hingegen nur bei weniger als 20 % der Unternehmen im Einsatz. In der Büroumgebung wird häufig auf Softwarelösungen (Videokonferenzen, Desktopvirtualisierung) und auf organisatorische Maßnahmen (Mitarbeitertraining) zurückgegriffen, so die Ergebnisse der Umfrage. Es gibt einige Beispiele welche aufzeigen, wie Betreiber von Rechenzentren und Entwickler

von Server-Hardware dem Thema Green IT begegnen. Ein aktuelles Beispiel ist das vom *Facebook*-Konzern Anfang 2011 der Öffentlichkeit vorgestellte *Open Compute Project*[3]. Ziel des Projekts ist es, ein besonders energieeffizientes Rechenzentrum zu entwickeln. Das Ergebnis ist laut Facebook ein Rechenzentrum, welches um 38 % effizienter ist und 24 % der Kosten einspart. Die Besonderheit bei diesem Projekt ist, dass die entwickelten Technologien und das gewonnene Wissen mit der Öffentlichkeit geteilt werden.

Als Mainboards für die Server kommen energieoptimierte AMD- und Intel-Boards zum Einsatz. Funktionen der Mainboards, welche für den Betrieb in einem Rechenzentrum nicht nötig sind, wurden von den Boards entfernt. Netzteile mit einem Wirkungsgrad größer als 90 % sorgen für die Stromversorgung der Boards. Dieser hohe Wirkungsgrad wird schon bei Lasten ab 20 % zugesichert. Auf den Boards sind Prozessoren aus der Intel Xeon 5500- und 5600-Reihe mit einer Verlustleistung (Thermal Design Power, TDP) von 60 bis 95 Watt [Intf] und Prozessoren der AMD Opteron 6100-Reihe mit 85 Watt TDP [AMD] verbaut. Der PUE-Wert liegt laut Facebook bei 1,07. Dieser ungewöhnlich hohe Wert bedeutet, dass 93 % der Energie direkt von den Servern verbraucht werden, und es gibt nur einen geringen Overhead (Energie für die Kühlung, Beleuchtung usw.). Die Facebook-Techniker setzen demnach auf aktuelle Prozessoren mit einer hohen Leistung. Zugleich ist die Verlustleistung der Prozessoren, verglichen mit anderen Prozessoren dieser Leistungsklasse, gering aufgrund der 45-nm-Fertigung.

Ein anderes Konzept wurde in dem von SeaMicro Ende 2010 vorgestellten SM10000-Server [SMS] verwirklicht. Dieser Server wurde speziell auf Energieeffizienz ausgelegt. In diesem Server sind 512 Intel N570-Atom-Prozessoren mit einer 45-nm-Struktur verbaut. Die Atom-Prozessoren haben eine relativ geringe Leistung, aber sie zeichnen sich vor allem durch einen niedrigen Stromverbrauch aus. Die Verlustleistung liegt bei nur 8,5 Watt bei einer relativ geringen Taktrate von 1,66 GHz. Das Interessante an diesem Beispiel ist, dass auch auf Kosten der Performance, auf eine energiesparende Prozessor-Architektur gesetzt wird. Um dem Nachteil der schwächeren Leistung zu begegnen, sind die Server für eine horizontale Skalierung ausgelegt. Bei dieser Art der Skalierung wird eine zu große Last (*Workload*) auf zusätzliche Server verteilt statt die Performance eines einzelnen Servers durch zum Beispiel Prozessoren mit höheren Taktraten zu erhöhen.

Diese Beispiele unterstreichen die hohe Bedeutung von Green IT für Rechenzentren.

## 2.2 Green IT und die Rolle der Datenbanken

Der Rolle der Datenbanken bezüglich des Energieverbrauchs in Rechenzentren, in denen üblicherweise große Mengen an Daten in Datenbanken verwaltet werden, wur-

[3]Die Internetseite des *Open Compute Projects* ist http://opencompute.org/.

de erst in den letzten Jahren Beachtung in der Literatur geschenkt. Ökonomische und ökologische Faktoren erfordern es, dass Datenbanken auch unter dem Aspekt der Energieeffizienz betrieben werden. Dieser Abschnitt stellt Lösungsansätze und neuere Publikationen vor, welche die Energieeffizienz von DBMS steigern. Manche dieser Vorschläge wurden bisher noch nicht ausreichend untersucht und noch nicht implementiert. Für andere Techniken gibt es wiederum schon Implementierungen und es liegen Ergebnisse vor, unter anderem ein speziell auf Energieeffizienz optimiertes DBMS mit dem Namen *WattDB*.

### 2.2.1 Energieverbrauch versus Performance

Lang und Patel [Lan09] stellten 2009 zwei Methoden vor, mit denen sich der Energieverbrauch einer Datenbank direkt beeinflussen lässt. Bei beiden dieser Techniken wird Energie gegen Performance getauscht. Dies bedeutet, dass beispielsweise durch die Implementierung einer der beiden Techniken die Antwortzeit von Abfragen zunimmt, aber insgesamt weniger Energie für die Bearbeitung einer Abfrage benötigt wird. Lang und Patel konnten in ihren Versuchen Konfigurationen erreichen, für welche die prozentuale Reduktion des Energieverbrauchs größer als die prozentuale Zunahme der Antwortzeit der Abfragen ist.

Die erste Methode, ***P**rocessor **V**oltage/Frequency **C**ontrol (**PVC**)* genannt, ermöglicht es dem DBMS, einen Prozessor gezielt mit einer geringeren Spannung und einer niedrigeren Taktrate zu betreiben. Dies basiert darauf, dass sich bei neueren Prozessoren die Spannung und die Frequenz durch den Wechsel des Prozessorzustandes (die sogenannten *P-States*, siehe auch Abschnitt 6.4.6) über Software-Befehle steuern lassen. Die Anwendung der Methode reduziert die Leistungsaufnahme, aber gleichzeitig auch die Performance des Prozessors. Lang und Patel konnten unter Verwendung eines nicht weiter genannten kommerziellen DBMS den Energieverbrauch um bis zu 49 % senken, während gleichzeitig die Antwortzeit (*engl. Responsetime*) der Abfragen um 3 % zunahm. Bei Verwendung von MySQL konnte der Energieverbrauch um 20 % gesenkt werden bei einer Zunahme der Antwortzeit von nur 6 %.

Die zweite Methode nannten Lang und Patel ***Q**uery **E**nergy-efficiency by Introducing Explicit **D**elays (QED)*. Diese Methode fällt in den Bereich des Workload-Managements. Bei dieser Methode wird die Startzeit der Bearbeitung von Abfragen gezielt verzögert, damit sich eine Warteschlange (*Queue*) mit einigen Abfragen bildet. Diese Queue kann dann vom Query-Optimizer auf Gemeinsamkeiten hin untersucht und ein sinnvoller Schedule kann erstellt werden. Die Besonderheit der QED-Methode ist, dass der Startzeitpunkt mancher Abfragen gezielt und absichtlich verzögert wird, damit sich eine Queue von Abfragen bildet, die sich sonst nicht bilden würde. Dies erhöht die Antwortzeit der Abfragen, welche in die Queue aufgenommen wurden, aber der Stromverbrauch sinkt aufgrund der erhöhten Performance des DBMS und der kürzeren Bearbeitungszeiten der Abfragen. Bei einfachen Selektions-Abfragen nahm in den Versuchen von Lang und Patel der Energieverbrauch bei Anwendung von QED um 54 % ab, während die durchschnittliche Antwortzeit um 43 % zunahm.

### 2.2.2 Energieeffizienz durch Flash-Speicher

Die Kombination von Solid State Disks (SSD und häufig auch NAND-Flash-Speicher oder kurz Flash-Speicher genannt) und Datenbankmanagementsystemen (DBMS) ist ein weiteres vielversprechendes Thema bezüglich Green IT und DBMS. Mit Flash-Speichern kann nicht nur Energie gespart, sondern gleichzeitig auch die Performance gesteigert werden. Schmidt et al. [SHKR08] sehen in Flash-Speichern die Zukunft für Datenbanksysteme aufgrund der hohen Geschwindigkeit von Lese- und Schreiboperationen und aufgrund des niedrigen Energieverbrauchs von Flash-Speichern im Vergleich zu herkömmlichen Festplatten (HDDs)[4]. Schmidt et al. stellen Möglichkeiten vor, wie die SSD-spezifischen Eigenschaften (größere Blockgröße, Schreiboperationen langsamer als Leseoperationen) für die Verwendung in einem Datenbank-Server genutzt werden können. Herausgehoben werden der niedrige Energieverbrauch und die höhere Geschwindigkeit gegenüber HDDS bei wahlfreiem/direkten Speicherzugriff (*Random read*).

Laut Schmidt et al. müssen DBMS-Algorithmen speziell für SSDs angepasst werden, da bei einer SSD eine bereits beschriebene Seite (*Page*) in einem Block nicht geändert, sondern nur neu geschrieben werden kann. Dazu muss allerdings der komplette Block neu geschrieben werden. Dies macht Schreiboperationen verglichen mit Leseoperationen aufwendig und damit teuer (SSD Write-Read-Asymmetrie). Diese Probleme müssten bei dem Design von SSD-spezifischen I/O-Algorithmen bedacht werden, so Schmidt.

Zudem nutzt eine SSD größere Blöcke (ein Block besteht aus 32 bis 128 Seiten und die Größe einer Seite liegt im Bereich von 512 Byte und 2 KB) als ein DBMS (üblicherweise 4 KB bis 64 KB große Blöcke). Es wird empfohlen, bei DBMS-internen Algorithmen erst mit dem *teuren* Löschen von Blöcken zu beginnen, wenn es keine leeren Blöcke mehr gibt. In einem leeren Block kann sofort geschrieben werden, ohne dass dieser vorher gelöscht werden muss. Als zweite Maßnahme sollte das DBMS gewährleisten, dass SSD-Blöcke möglichst vollständig gefüllt werden, damit die Fragmentierung gering gehalten wird. Um dies zu gewährleisten werden, Änderungen nicht direkt in einem Block geschrieben, sondern die Änderungen werden gesammelt (*flash-aware DB buffers*) und erst bei einer ausreichend großen Datenmenge in einen Block geschrieben, der dann vollständig gefüllt ist. In einer Fallstudie von Beckmann, Meyer, Sanders und Singler 2010 [BMSS10], in der es zwar nicht um ein DBMS, aber um das energieeffiziente Sortieren einer großen Datenmenge (JouleSort-Benchmark [RSRK07]) geht, wurde dieses Problem so gelöst, dass beispielsweise nur große Datenmengen auf einmal von einer SSD gelesen oder auf diese geschrieben werden. Zu solchen Besonderheiten des Flash-Speichers bezüglich DBMS gibt es noch eine Reihe weiterer Lösungsansätze in der Literatur [SOH09], [HOB09], [OH10], [SHH10],

[4] Die Autoren geben keine genauen Werte an, aber in unseren Messungen benötigte die von uns verwendete SSD gegenüber einer HDD ungefähr 10 Watt weniger im Leerlauf. Bei anderer Hardware (vgl. [BMSS10]) wurden bei einem Vergleich zwischen HDD und SDD ein Unterschiede von ungefähr 3 Watt im Leerlauf gemessen.

[OHS10], [HS11b].

### 2.2.3 Die Datenbank WattDB

Die Nutzung von Flash-Speichern allein macht eine Datenbank noch nicht energieeffizient, weil der Energieverbrauch des Speichers nur einen Teil des Gesamtenergieverbrauchs einer Datenbank ausmacht. Haerder et al. beziffern das Einsparungspotenzial durch die Nutzung von Flash-Speichern auf weniger als 10 % [HHOS11]. Aus diesem Grund entwickelte das Team um Haerder die Datenbank *WattDB*, welche aus mehreren Rechen-Knoten besteht. Jeder Knoten kann dynamisch in die Datenbank eingelinkt oder ausgelinkt werden, abhängig von dem aktuellen Workload. Ein einzelner Knoten kann beispielweise ein Computer sein. Dieses Konzept nennen sie *energieproportionales Datenbank-Management*. Die Idee hinter diesem Konzept ist, dass eine Hardware-Komponente nur Energie verbrauchen sollte, wenn diese auch tatsächlich genutzt wird. Der Energieverbrauch soll sich proportional zu dem Grad der Hardware-Nutzung verhalten. Das ideale Verhältnis wird durch die grüne Linie in Abbildung 2.3 ausgedrückt. Der Abbildung 2.3 liegt nur ein einzelner Knoten zu Grunde. Der Knoten

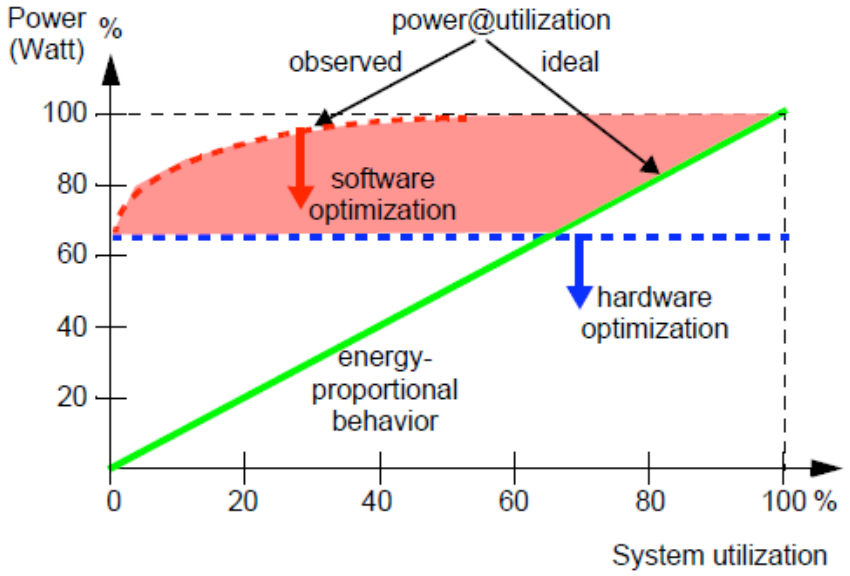



Abbildung 2.3: Energieverbrauch im Verhältnis zur Systemnutzung (bei einem Knoten). Die Abbildung wurde aus [HHOS11] entnommen.

in der Abbildung benötigt allerdings auch im Leerlauf bereits 65 Watt (Startpunkt der roten Linie), so dass der ideale Watt-Wert von Null bei einer Systemauslastung von 0 % nicht erreicht werden kann. Durch Hardware-Optimierungen (blaue Linie und blauer Pfeil nach unten), könnte der Stromverbrauch im Leerlauf gesenkt werden. Im Leerlauf benötigte der Knoten, auf welchem die Abbildung basiert, ungefähr 65 Watt. Damit wird mit diesem Knoten das Ideal, dargestellt durch die grüne Linie, nie erreicht. Um das Ideal bei einem Cluster aus mehreren Knoten zu erreichen,

muss laut Haerder gewährleistet sein, dass einzelne Knoten je nach Nutzungsgrad des Systems dynamisch aus dem Cluster entfernt und diesem wieder hinzugefügt werden können. Außerdem müsste jeder einzelne Knoten eine so niedrige Leistungsaufnahme wie möglich haben.

Haerder stellt eine solche neuartige Systemarchitektur mit ihren Eigenschaften vor [HHOS11]. Das WattDB-Projekt basiert auf dieser Systemarchitektur. Statt eines einzelnen speicher- und rechenstarken Servers wird für WattDB ein Cluster aus vielen einzelnen und energiesparenden Knoten verwendet. Der Cluster-Kern besteht aus einem einzigen Knoten, welcher Koordinationsaufgaben und die Verwaltung des vorhandenen Speichers übernimmt. Weitere Knoten für die Berechnung von Abfragen können jederzeit dynamisch, je nach Umfang des Workloads, hinzugefügt (und später auch wieder entfernt) werden. Durch dieses schrittweise Hinzufügen oder Entfernen von energieeffizienten Knoten erfüllt das Gesamtsystem die Eigenschaft der Energieproportionalität (vgl. grüne Linie in Abbildung 2.3).

Eine Schwierigkeit dieses Systemdesigns ist, dass jeder Knoten Zugriff auf alle Daten der Datenbank haben muss. Dazu musste eine verteilte I/O-Architektur entwickelt werden, bei der alle Knoten zu jedem Zeitpunkt Zugriff auf alle externen Speichermedien (Flash-Speicher, Festplatten) haben. Details zu der Hardware-Ausstattung und der Leistungsaufnahme der einzelnen Konten sind in [HS11a] beschrieben. Die Idee einer *energie-proportionalen* Datenbank verspricht ein hohes Energie-Einsparpotenzial, da heutige Systeme im Leerlauf oder bei geringer Last häufig zu viel Strom verbrauchen. Bei dem Server der für die Tests für diese Arbeit verwendet wurde, beträgt die Leistungsaufnahme im Leerlauf beispielsweise schon 90 Watt (vgl. Abschnitt 5.1).

### 2.2.4 Weitere Lösungsansätze

Goetz Graefe von den *Hewlett-Packard Laboratories* veröffentlichte 2008 eine Sammlung von Lösungsansätzen [Gra08], um die Energieeffizienz von Datenbankmanagementsystemen zu verbessern. Das Paper sei mehr als Herausforderung denn als eine Sammlung von Lösungen anzusehen, so Goetz.

In diesem Abschnitt werden einige der Lösungsansätze und Ideen von Goetz kurz dargestellt, um weitere Möglichkeiten zur Steigerung der Energieeffizienz einer Datenbank aufzuzeigen. Im ersten Teil dieses Abschnitts werden Lösungsansätze im Bereich der Hardware vorgestellt. Im zweiten Teil des Abschnitts stehen Software-Lösungen im Mittelpunkt.

Im Bereich der **Hardware** gibt es schon seit Längerem Bestrebungen, Energie zu sparen. Dies liegt an der großen Bedeutung von sparsamer Hardware für Laptops und andere mobile Geräte. Zu den Energiespartechniken gehören das dynamische Anpassen von Spannung und Taktrate eines Prozessors an die momentane Auslastung eines Prozessors, temperatur-gesteuerte Lüfter, Stromsparmodi von Festplatten und Mehrkern-Prozessoren statt immer höherer Taktraten.

Vorschläge für Server-Hardware sind das Aktivieren und Deaktivieren von Arbeitsspeicher-Riegeln, der Austausch von HDDs durch Flash-Speicher und Virtualisierung. Ein Austausch von Festplatten gegen Flash-Speicher ist aus ökonomischer Sicht aufgrund der (noch) hohen Kosten von Flash-Speichern nicht immer sinnvoll, aber darüber hinaus muss insbesondere für DBMS laut Schmidt et al. das Speichermanagement von DBMS speziell für Flash-Speicher optimiert werden [SHKR08].

Goetz schreibt, Flash-Speicher könnten entweder als (langsame) Erweiterung des Arbeitsspeichers oder als (schnelle) Erweiterung des Massenspeichers dienen. Die Nutzung als Arbeitsspeicher-Erweiterung käme möglicherweise ohne Änderungen am DBMS aus. Die Nutzung als Massenspeicher erschiene jedoch angemessener, da Flash-Speicher als persistente Speicher dienen können. Zur Überwindung der Write-Read-Asymmetrie von Flash-Speichern käme ein write-optimierter B-Baum [Gra04] als Datenstruktur in Frage.

Im Bereich der **Software** sieht Goetz ein Einsparungspotenzial, welches bisher kaum ausgenutzt wird. Ein Abfrageoptimierer (query optimizer) könnte zum Beispiel Abfragepläne aufgrund des Energieverbrauchs statt aufgrund der Performance vergleichen. Das Ergebnis könnten andere, bezüglich des Energieverbrauchs optimierte Abfragepläne sein. Zum Beispiel könnte ein *Index Nested Loop Join* (I/O und CPU werden abwechselnd beansprucht) einem **Hash Join** (Scan (I/O) und das Hashing (CPU) überlappen) vorgezogen werden trotz einer höheren Antwortzeit.

Für die Abfrageoptimierung ist es wichtig, in welcher Schicht der Speicherhierarchie ein Index verfügbar ist [RD05]. Wenn in der traditionellen Abfrageoptimierung die I/O-Bandbreite für die Auswertung eines Prädikats überschritten wird, dann sind die Kosten für das Scannen eines Index auf der Festplatte oder im Speicher dieselben. Bei einer Optimierung hinsichtlich des Energieverbrauchs wäre dies nicht der Fall.

Die bessere Nutzung von Mehrkernprozessoren ist eine weitere Maßnahme zur Verbesserung der Energieeffizienz. Die Granularität der Parallelität könnte feiner werden. Beispielsweise könnte ein Thread (ein Prozessorkern) bereits Daten laden, während ein anderer Thread die Daten schon weiterverarbeitet.

Die von mehreren Threads parallel durchgeführte Bearbeitung einer Abfrage beeinflusst die Energieeffizienz auf zwei gegensätzliche Arten. Zum einen entsteht durch die Parallelität ein erhöhter Koordinationsbedarf. Dies ziehe einen zusätzlichen Energiebedarf nach sich, so Goetz. Dem gegenüber steht ein niedrigerer Energiebedarf pro Rechenkern, da die Arbeit auf mehrere Kerne verteilt wird und die Rechenkerne nicht mit der vollen Taktrate laufen müssen. Dadurch dass der Energiebedarf eines Rechenkerns nicht linear zu der Spannung und Frequenz sei, könnte sich trotz eines insgesamt erhöhten Rechenbedarfs ein energiesparender Effekt einstellen.

Kompressionsverfahren können nicht nur unter dem Aspekt der Performance (zum

Beispiel in [GP87]), sondern auch unter dem Aspekt der Energieeffizienz betrachtet werden. Eine noch offene Frage ist, ob die zum Komprimieren und Dekomprimieren benötigte Energie für den Prozessor größer als die Energie-Einsparung durch die geringere I/O-Aktivität ist.

Eine ungewöhnliche Energie-Einsparmöglichkeit bieten Update-Operationen. Updates von Daten müssen zum Beispiel auch für alle Indizes gemacht werden. Dies muss aber nicht notwendigerweise direkt geschehen, so Goetz. Durch das Aufschieben von Index-Aktualisierungen können Lastspitzen vermieden werden. Eine mögliche Form der verzögerten Index-Aktualisierung ist, die Aktualisierung zwar vorläufig zu speichern, aber noch nicht an der finalen, suchoptimierten Stelle in zum Beispiel einem B-Baum.

## 2.3 Die Datenbank Caché

Im Jahre 1997 wurde die erste Version von Caché [Inte] als Nachfolger von M/SQL -bzw. Open M von Intersystems veröffentlicht. Bei M/SQL und Open M handelt es sich um integrierte Umgebungen zur Entwicklung und Ausführung von Datenbankanwendungen. Sowohl M/SQL als auch Open M vereinen die eigens für Datenbankanwendungen entwickelte prozedurale Programmiersprache M (alias MUMPS) mit der schon damals weit verbreiteten Abfragesprache für relationale Datenbanken, der Structured Query Language (SQL).

Da es sich bei MUMPS um eine in den späten 1960er Jahren von Neil Pappalardo und Kollegen am Massachusetts General Hospital (Utility Multi-Programming System) entwickelte Programmiersprache für die Entwicklung von multi-user datenbankgestützten Anwendungen handelt, wird auch deutlich, dass die Wurzeln von Caché im Gesundheitswesen liegen.

Caché ist eine objektorientierte, im Kern hierarchische Datenbank, die auf klassischen $B^+$-Bäumen basiert, die mit multidimensionalen Arrays, den sogenannten Globals, kombiniert sind. Diese multidimensionale Daten-Engine erlaubt es, komplexe Datenstrukturen kompakt zu speichern.

Eine Besonderheit von Caché ist der optionale Zugriff auf diese multidimensionale Datenstruktur über Objekte oder SQL. Es kann also wahlweise wie eine relationale oder objektorientierte Datenbank verwendet werden, gerade so, wie es für ein gegebenes Problem am günstigsten ist. Damit dies funktioniert, erzeugt Caché bei der Definition einer Oberklasse automatisch eine für SQL nutzbare relationale Datenstruktur. Umgekehrt wird beim Import einer DDL-Definition in das Data-Dictonary eine Objektbeschreibung der Daten erzeugt. Caché sorgt in beiden Fällen dafür, dass die Strukturen der beiden Ansätze stets konsistent sind und nur eine Definition gepflegt werden muss. Die Daten werden in einem einzigen integrierten Data Dictionary nur ein Mal beschrieben und stehen sofort für alle Zugriffsmethoden zur Verfügung.

Die Zuordnung der Objekte und Tabellen in die multidimensionale Struktur erzeugt Caché automatisch, obwohl den Programmierern auch die Möglichkeit gegeben wird, diese Zuordnung selbst festzulegen.

"Die multidimensionale Daten-Engine selbst kennt kein Data Dictionary und folglich auch keine Datendefinitionen."[Cac]

Ein weiterer Vorteil von Caché ist die Möglichkeit, dass Entwickler direkt auf die Globals zugreifen können, wodurch besonders performante Anwendungen entwickelt werden können, da der Umweg über die virtuellen Maschinen für die SQL -und Objektinterpretation vermieden wird. Für den direkten Zugriff auf die Globals stehen Scriptsprachen wie *Caché Object Script* (basierend auf MUMPS) zur Verfügung.

In den Globals können Daten mit einer beliebigen Anzahl von Subscripts gespeichert werden, die nicht notwendigerweise vom gleichen Typ sein müssen. So kann ein Subscript vom Typ Integer und ein anderer vom Typ String sein, obwohl sie sich auf der gleichen Subscript-Ebene befinden. Nach Angaben von Intersystems laufen Anwendungen auf Caché, die mit gewöhnlichen SQL-Abfragen arbeiten, aufgrund der effizienten multidimensionalen Struktur performanter als auf traditionellen relationalen Datenbanken. [Cac]

Trotz der guten Unterstützung des relationalen Datenbankmodells, unter anderem auch durch standardisierte Schnittstellen wie ODBC- und JDBC-Treiber, ist Caché, wie bereits erwähnt, eine objektorientierte Datenbank. Da das Objektmodell von Caché auf dem ODMG-Standard (Object Database Management Group) basiert, werden alle aus der objektorientierten Programmierung bekannten Konzepte wie Kapselung, Mehrfachvererbung, Polymorphismus und Collections unterstützt. Um die Arbeit für Entwickler zusätzlich zu vereinfachen, stellt Caché eine Sammlung von Klassen bereit, mit deren Hilfe Objekte direkt aus .NET, C++ oder Java bearbeitet werden können, um nur einige zu nennen.

Eine weitere Besonderheit von Caché ist die transaktionale Bitmap-Indizierung, durch die die Performance komplexer Aufgaben signifikant erhöht werden kann.[Cac] Die Bitmap-Indizierung stellt für jeden Wert einer Spalte oder Eigenschaft einen Bitstring bereit. Jede Stelle des Bitstrings steht für ein in der Datenbank gespeichertes Objekt. Eine 1 im Bitstring bedeutet, dass dieses Objekt die Eigenschaft aufweist, eine 0, dass es diese nicht aufweist. Durch diese Art der Darstellung können komplexe Abfragen durch Boolesche Operationen auf den Bitstrings durchgeführt werden, wodurch die Antwortzeiten von Abfragen auf großen Datenmengen um den Faktor 100 beschleunigt werden können. [Cac] Die Nachteile einer herkömmlichen Bitmap-Indizierung, wie langsame Aktualisierungszeit und großer Speicherplatzbedarf, werden bei transaktionalen Bitmap-Indizes durch die multidimensionale Struktur eliminiert.

Für die Verwaltung der Datenbank und den Zugriff auf die Daten bietet Caché drei Werkzeuge (Systemmanagement-Portal, Studio, Terminal) an, die im Folgenden kurz

beschrieben werden. Alle Werkzeuge können über das Infobereichssymbol, den „blauen Würfel“ in der Taskleiste, gestartet werden.

### 2.3.1 Das Systemmanagement-Portal

Das Systemmanagement-Portal ist eine browserbasierte Oberfläche, über die eine Vielzahl von häufig benötigten Einstellungen von Caché vorgenommen werden kann. Abbildung 2.4 zeigt die Startseite des Portals. Über das Portal können z.B. neue Datenbanken erzeugt oder Ad-hoc-Abfragen mittels SQL gestellt werden. Auch die Verwaltung von Backups oder der Datenimport mittels CSV-Dateien sind über das Systemmanagement-Portal möglich.

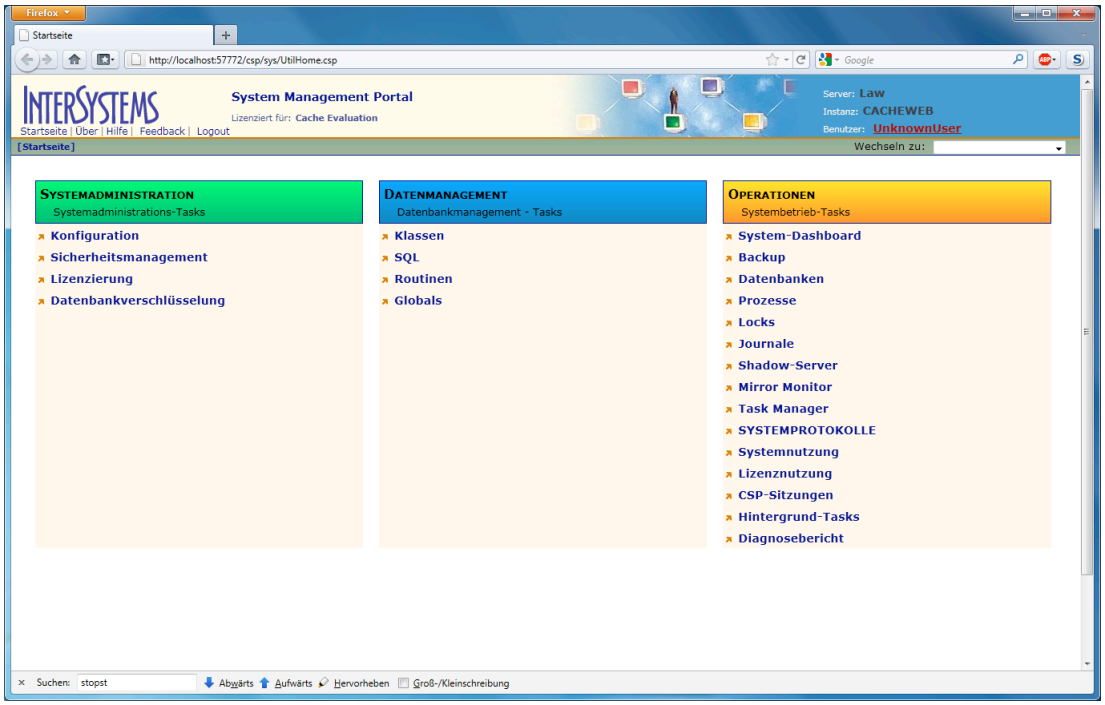



Abbildung 2.4: Die Startseite des Systemmanagement-Portals.

### 2.3.2 Das Caché Studio

Im *Caché Studio* können die Metadaten der Datenbank bearbeitet werden. Hier werden z.B. Primary-Keys, Foreign-Keys und Indizes sowie Klassen und deren Attribute verwaltet. Abbildung 2.5 zeigt das Caché Studio mit einer geöffneten Klasse. Interessant ist, dass Tabellen, die über *Data Definition Language*-Befehle (DDL) im Systemmanagement-Portal angelegt wurden, im Studio als Klasse erscheinen und dort bearbeitet werden können. Der umgekehrte Weg ist ebenfalls möglich: Im Studio definierte Klassen können über das Systemmanagement-Portal betrachtet werden.

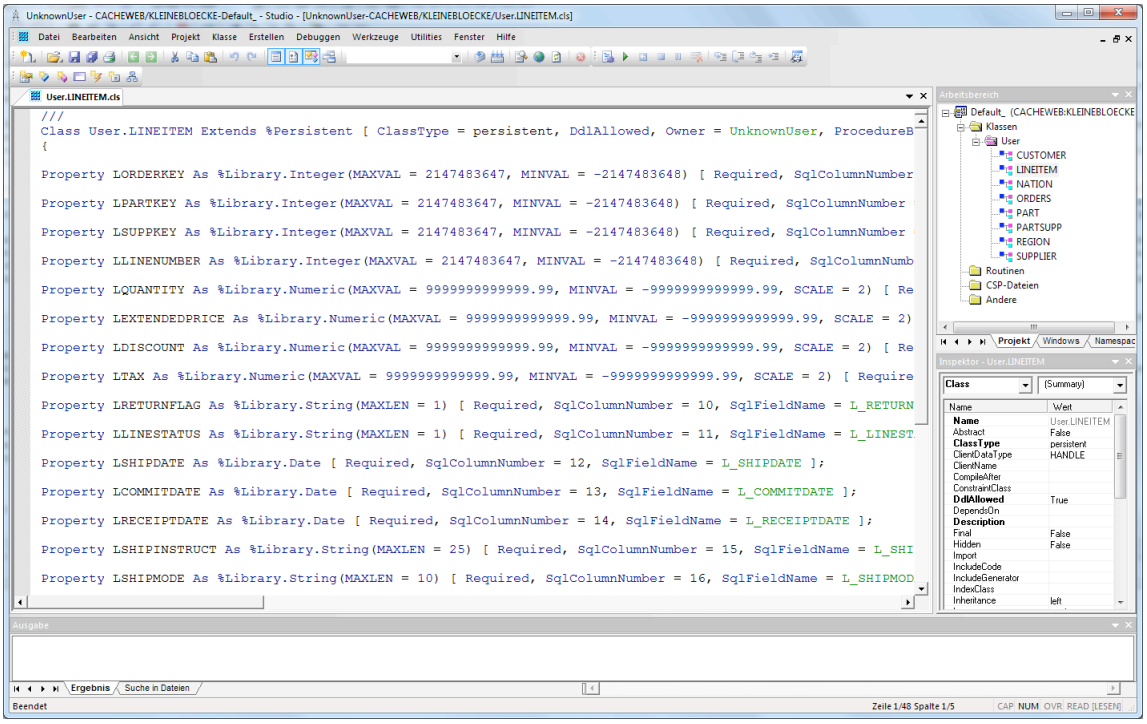



Abbildung 2.5: Geöffnete Klassendefinition in Caché Studio.

### 2.3.3 Das Caché Datenbank-Terminal

Das Terminal ist eine einfache Konsolenanwendung, aber gleichzeitig das mächtigste dieser drei Werkzeuge. Während das Studio und das Systemmanagementportal nur die nötigsten Funktionen enthalten und daher relativ anwenderfreundlich sind, können über das Terminal auch sehr spezielle Einstellungen vorgenommen werden. Hierfür wird die bereits erwähnte Scriptsprache *Caché Object Script* verwendet. Ein Beispiel für einen einfachen *Object Script*-Befehl ist der Befehl „do $system.SQL.Shell()“, der innerhalb des Terminals eine Anwendung zum Ausführen von SQL-Befehlen startet.

## 2.4 Leistungsbewertung von Datenbanken

### 2.4.1 Grundlagen des Benchmarkings

Ein Benchmark ist ein Programm, welches in der Lage ist, ein Computersystem auf seine Leistung hin zu überprüfen. Dies kann sehr allgemein sein, wie beispielsweise das Messen der Geschwindigkeit der CPU bezogen auf Operationen pro Sekunde. Einem Benchmark liegt immer ein Effizienzmaß zugrunde.

Die ersten Benchmarks entstanden 1970. Das Maß nach dem bewertet wurde, waren hierbei die Instruktionen pro Sekunde (*MIPS*) und die Gleitkomma-Operationen

pro Sekunde (*FLOPS*). Durch die Weiterentwicklung der Informationstechnologie und den damit steigenden Grad an Komplexität der Anwendungen, war in der Folge eine Entwicklung von Benchmarks nötig, die eine genauere Bewertung für Computersysteme zuließen. Dies sind anwendungsbezogene Benchmarks. Diese zeichnen sich durch ihr Effizienzmaß aus. Es steht hierbei nicht die Leistungsbewertung der Hardware im Vordergrund, sondern es geht vielmehr darum festzustellen, wie effizient eine bestimmte Anwendung auf dem getesteten Computersystem ausgeführt werden kann.

### 2.4.2 Datenbank-Benchmark

Um Datenbanken und deren zugrundenliegenden Computersysteme zu testen, existieren spezielle Datenbank-Benchmarks. Für Datenbanken im Unternehmensumfeld wird zwischen zwei Anwendungsszenarien unterschieden: *Online Transaction Processing (OLTP)* und *Online-Analytical-Processing (OLAP)*. Beide Anwendungsszenarien stellen unterschiedliche Anforderungen an die Datenbank.

Das OLTP-Anwendungsszenario ist charakterisiert durch ein hohes Aufkommen an einfachen Transaktionen, die von der Datenbank verarbeitet werden müssen. Diese sind das Resultat des Tagesgeschäftes eines Unternehmens, wie der Wareneingang oder der Warenausgang. Die Transaktionssicherheit und die parallele Ausführung von Anfragen stehen hierbei im Vordergrund. Es muss durch das Datenbanksystem sichergestellt sein, dass die Konsistenz der Daten immer gewahrt bleibt, trotz vieler Lese- und Schreiboperationen [Wer07].

OLAP-Abfragen sind relevant für die Optimierung von Geschäftsprozessen. Sie zeichnen sich durch eine hohe Komplexität und durch Operationen auf einem großen Datenvolumen aus. Die Antwortzeiten für die Bearbeitung der Abfrage stehen hierbei im Vordergrund, da auf Basis des Ergebnisses Entscheidungen getroffen werden müssen, die teils als unternehmenskritisch zu betrachten sind. Die Verwendung von OLAP ist beispielhaft in Abbildung 2.6 zu sehen.

Das *Transaction Processing Performance Council (TPC)* ist ein 1988 gegründetes Konsortium, das aus vielen Unternehmen der IT-Branche besteht. Ziel dieses Konsortiums ist eine unabhängige Leitungsbewertung von Datenbanksystemen zu ermöglichen. Dabei stellt der TPC verschiedene Benchmarks bereit, die die Bemessung der Leistung einer Datenbank auf ein bestimmtes Anwendungsszenario ermöglichen.
Das TPC bietet aktuell drei Benchmarks an: TPC-C, TPC-E und TPC-H. TPC-E und TPC-C spiegeln ein Anwendungsszenario (Workload) eines Unternehmens mit überwiegend OLTP-Abfragen wieder. TPC-E ist eine Weiterentwicklung des TPC-C. Als Effizienzmaß dienen bei diesen beiden Benchmarks die Transaktionen pro Minute (tpmC und tpmE), bei gleichzeitiger Wahrung der Konsistenz der Daten. Zusätzlich werden die Transaktionen pro Minute noch in Relation zu den Gesamtkosten des Systems gesetzt [TPC].

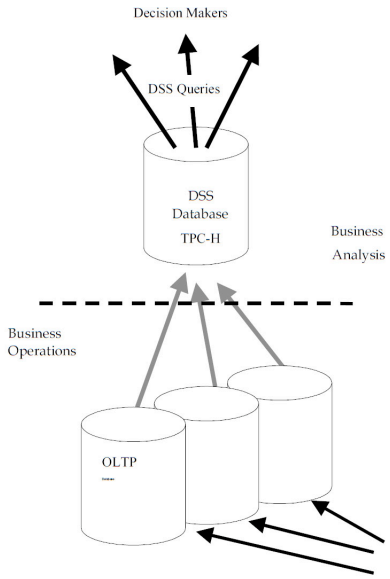



Abbildung 2.6: Verknüpfung von OLAP und OLTP.

### 2.4.3 Der TPC-H Benchmark

Der TPC-H ist ein Entscheidungshilfe-Benchmark. Er zeichnet sich durch eine hohe Komplexität der Abfragen aus und die Einbeziehung großer Datenmengen. Der Benchmark simuliert ein Handelsunternehmen mit verschiedenen Kunden, Zulieferern und Standorten. Die Anfragen werden ad-hoc gestellt und dabei ist es möglich, dass mehrere Benutzer parallel Anfragen stellen. Das Effizienzmaß beim TPC-H besteht aus Abfragen pro Stunde bei einer gegebenen Datenbankgröße (QphH@Size) [TCP].

Der TPC-H Benchmark stellt die Programme *DBGEN* und *QGEN* zur Verfügung. DBGEN dient zur Generierung der Daten mit einem bestimmten Skalierungsfaktor, wobei der Faktor 1 eine Datenbank der Größe von 1 GB erzeugt. Diese enthält Daten von 10.000 Händlern und Zulieferern und hat ca. 10.000.000 Zeilen. Die Skalierung wirkt auf jede Tabelle unterschiedlich.

QGEN dient zur Generierung von 22 Abfragen, die einen OLAP-Workload simulieren. Diese Abfragen liefern Antworten auf geschäftsrelevante Prozesse. Sie beziehen einen großen Teil der von DBGEN generierten Daten mit ein. Die Abfragen zeichnen sich durch hohe Komplexität aus. Diese ergibt sich durch die Nutzung mehrere Tabellen für die Abfragebearbeitung. Abbildung 2.7 zeigt das Schema des TPC-H Datenmodells und die Relationen der Tabellen zueinander.

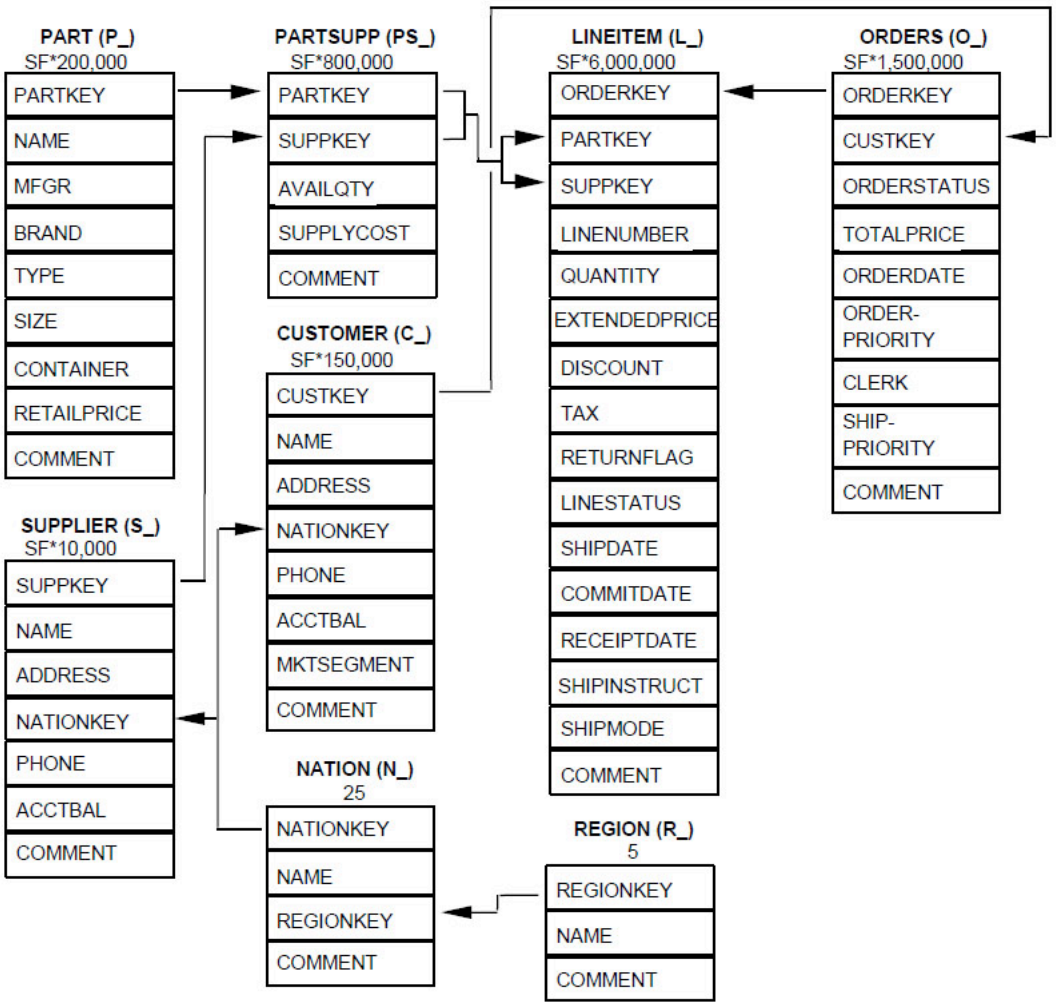



Abbildung 2.7: TPC-H Datenbankschema.

### 2.4.4 Die 22 Abfragen des TPC-H Benchmarks

Die Caché-konforme SQL-Syntax der 22 Abfragen des TPC-H Benchmarks [TCP] befindet sich im Anhang. Im Folgenden werden die 22 Abfragen des TPC-H Benchmarks inhaltlich beschrieben.

**Q1 Preisbericht**
Es wird ein Preisbericht erstellt in dem alle Auftragspositionen in einem bestimmten Zeitraum verzeichnet sind. Diese sollen nach den Attributen LINESTATUS und RETURNFLAG gruppiert werden. Für jede Gruppe sollen umsatzrelevante Daten ausgegeben werden, das sind zum Beispiel der Gesamtpreis oder die Anzahl.

**Q2 Günstigster Lieferant**
Es wird ermittelt welcher Zulieferer für ein bestimmtes Teil gewählt werden soll. Kriterien sind hierbei der Standort des Lieferanten und die Lieferkosten. Generiert wird ein Bericht, der alle qualifizierten Lieferanten mit Kontakt- und Umsatzdaten enthält.

**Q3Versand Priorität**
Es wird ausgeben welche Aufträge aus einem bestimmten Marktsegment noch nicht ausgeliefert wurden. Kriterien sind hierbei, dass nur die 10 Aufträge mit dem höchsten Umsatz ausgegeben werden, wobei der Fokus auf der Lieferpriorität liegt.

**Q4 Bestellpriorität Verifizierung**
Hierbei wird festgestellt wie gut das System zur Festlegung der Lieferpriorität funktioniert. Hierbei werden alle Datensätze ausgegeben, die in einem bestimmten Zeitraum bestellt wurden, aber nach dem zugesagten Lieferdatum ankamen.

**Q5 Lokales Händler Volumen**
Es soll festgestellt werden ob es sich lohnt in einer bestimmten Region Verteilungszentren zu errichten. Basis für diese Erhebung sind die Umsatzdaten einer Region, wobei nur Händler und Kunden aus derselben Region untersucht werden.

**Q6 Ertragsvorhersage**
Es wird untersucht um welche Summe sich der Ertrag in einem bestimmten Jahr erhöht hätte, falls ein Rabatt für eine bestimmte Menge an Teilen nicht gewährt worden wäre.

**Q7 Transportvolumen**
Die Abfrage gibt an welcher Warenwert in einem Zeitraum zwischen zwei Nationen verschickt wurde. Ziel ist es die Lieferverträge zu überarbeiten.

**Q8 Marktanteil**
Es wird errechnet welchen Marktanteil eine Nation in einer Region bei einem gegebenen Produkt in den letzten zwei Jahren hat.

**Q9 Ertrag**
Es wird ermittelt mit welcher Produktlinie in einem Zeitraum der größte Gewinn erwirtschaftet wurde.

**Q10 Rückgabe**
Es werden Kundendaten ausgegeben, die die Kontaktdaten enthalten. Das Kriterium ist hierbei der Umsatzverlust, der durch Rücksendungen entstanden ist. Es werden die Top 20 ausgegeben.

**Q11 Lagerbestand**
Es soll ermittelt werden welche Teile in einer bestimmten Region lagernd sind und einen Signifikaten Anteil am Gesamtwarenwert haben. Der Gesamtwarenwert bezieht sich nur auf die in einer Nation lagernden Teile.

**Q12 Versand Analyse**
Es wird eine Aussage darüber getroffen, ob das Wechseln zu einer günstigeren Lieferart, kritische Aufträge negativ beeinflussen kann.

**Q13 Umsatzverlust**
Es wird festgestellt welcher Sachbearbeiter, in einem bestimmten Zeitraum, den höchsten Umsatzverlust durch Rücksendungen verursacht hat.

**Q14 Werbung**
Es wird ermittelt, ob durch die Schaltung von Werbung zu einem höheren Umsatz gekommen ist. Betrachtet werden hierbei nur die Produkte die beworben wurden.

**Q15 Top Händler**
Es wird der beste Lieferant ermittelt, wobei hier der Anteil am Gesamtumsatz ausschlaggebend ist.

**Q16 Händlerlager**
Es wird ermittelt welche Lieferanten eine bestimmte Sorte von Teilen auf Lager haben, wobei Lieferanten ausgeschlossen werden, die eine negative Bewertung bekommen haben.

**Q17 Kleine Lieferungen**
Es wird der Verlust berechnet, der durch die Ablehnung kleiner Bestellungen eines bestimmten Teils, entstehen würde.

**Q18 Große Bestellungen**
Es werden die Top 100 Kunden ausgegeben, die das größte Umsatzvolumen haben, wobei die Bestellungen eine Mindestanzahl an Einheiten haben müssen.

**Q19 Gesamtumsatz**
Es soll bestimmt werden welches Produkt den höchsten Umsatz erzielt hat, wobei

entscheidend ist wie das Produkt ausgeliefert wurde.

**Q20 Werbe Aktion**
Es wird ermittelt, ob es in einer bestimmten Nation Lagerbestände existieren, die sich für eine Rabattaktion eigenen.

**Q21 Versandverzögerung**
Die Abfrage gibt die Händler wieder, die mit der Auslieferung von Teilen in Verzug geraten sind.

**Q22 Geographische Nachfrage**
Die Abfrage schlüsselt auf, ob Kunden existieren die wieder kontaktiert werden sollten. Die Aufschlüsselung erfolgt über den Gesamtumsatz des Kunden und den Zeitraum in dem von Ihm nichts bestellt wurde.

# 3 Entwurf

## 3.1 Versuchsaufbau

Abbildung 3.1 stellt die Versuchsanordnung bei unseren Messungen dar. Das Strommessgerät (LMG95) wurde zwischen Datenbankserver und Stromanschluss zwischengeschaltet damit auf diese Weise der Stromverbrauch des Servers überwacht werden kann. Neben dem Datenbankserver wurde ein zweiter Rechner, ein *Überwachungsrechner*, verwendet. Auf diesem Überwachungsrechner wurden die Daten des Strommessgeräts und die Performance-Daten des Datenbankservers zentral gesammelt und protokolliert. Die Auslagerung der Protokollierung auf einen eigenen Rechner stellte sicher, dass der Datenbankserver nicht durch die Erstellung von Protokolldateien während einer Messung belastet wurde. Dies hätte die Performance des Datenbankservers beeinflussen können und die Protokolle verfälscht.

Auf dem Datenbankserver hingegen lief ausschließlich die Datenbank Caché auf dem Betriebssystem Windows Server 2008. Unter diesem Betriebssystem gibt es ein Microsoft-Programm mit dem Namen *Leistungsüberwachung*. Mit diesem Programm können Leistungsdaten in Echtzeit angezeigt oder in einer Protokolldatei gespeichert werden. Auf dieses Programm wurde über das Netzwerk vom Überwachungsrechner zugegriffen. Die Protokolldateien wurden ausschließlich auf dem Überwachungsrechner gespeichert.

Um während der Messungen allerdings nicht nur die Performance-Daten des Datenbankservers, sondern gleichzeitig auch die Daten des Strommessgeräts aufzeichnen zu können, wurde eine weiteres Programm entwickelt, welches auf dem Überwachungsrechner lief. Dieses Programm wurde in der Programmiersprache C# im *.Net-Framework 4.0* geschrieben. Dem Programm gaben wir den Namen *GreenDB*. Die Abfragen wurden mit diesem Programm über das Netzwerk von dem Überwachungsrechner an den Datenbankserver geschickt. Außerdem konnten mit dieses Programm die Messungen weitestgehend automatisiert durchgeführt werden. Auf diese Weise konnte der Datenbankserver beispielsweise auch über Nacht für Messreihen verwendet werden.

## 3.2 GreenDB

Das Programm *GreenDB* musste Folgendes leisten:

- Die für den Stromverbrauch relevanten Leistungsindikatoren (CPU-Nutzung

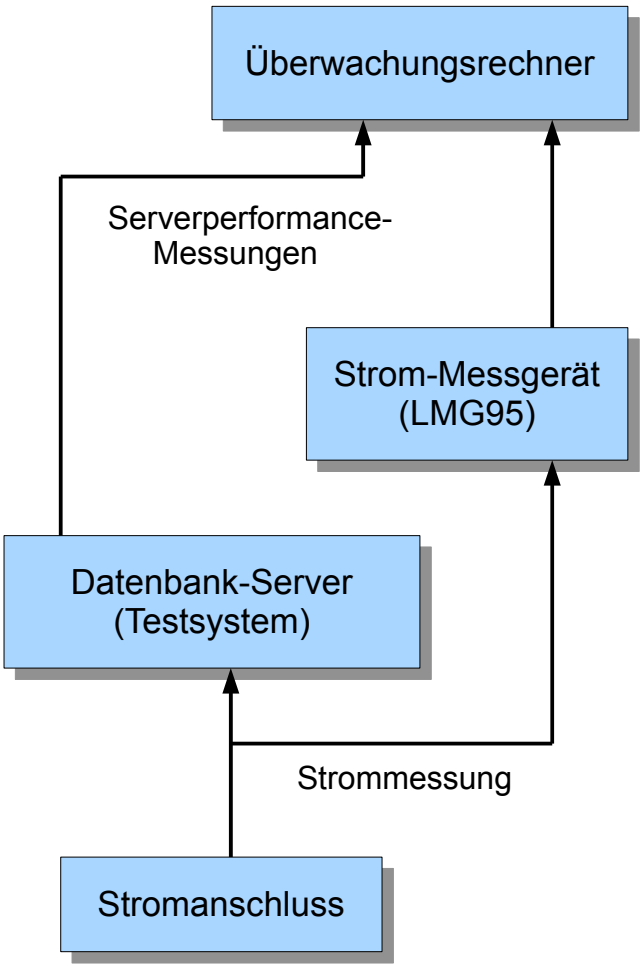



Abbildung 3.1: Struktur des Versuchsaufbaus.

und Lese-und Schreibzugriffe auf der Festplatte) mussten von dem Datenbankserver auf den Überwachungsrechner übertragen werden.

- Die beobachteten Leistungsindikatoren sollten mit dem vom LMG95 ermittelten Stromverbrauch zusammen mit einem Zeitstempel in einem Datensatz (Logdatei) zusammengeführt und für die spätere Verarbeitung im CSV-Format gespeichert werden.

- Die erfassten Daten sollten für jede Abfrage in einer eigenen Datei gespeichert werden

- Das Programm benötigte eine Funktion, mit der jede Abfrage auch auf einer

frisch gestarteten Datenbank durchgeführt werden konnte (Kaltstartsimulation).

- Das Programm selbst durfte den Datenbankserver nicht belasten.

Um die Leistungsindikatoren des Datenbankservers zu erhalten, verwendeten wir die Klasse *PerformanceCounter* aus dem Namespace *System.Diagnostics*, da diese alle benötigten Indikatoren liefern konnte und zudem die Möglichkeit besitzt, diese Daten auch von einem entfernten Computer zu ermitteln.

Leider mussten wir feststellen, das bei dem ursprünglich angedachten Messintervall von weniger als einer Sekunde der „PerformanceCounter“ die Messwerte nicht rechtzeitig vom Datenbankserver erhielt, wenn die CPU des Datenbankservers zu 100 % ausgelastet war. Die Folge war, dass GreenDB auf diese Werte warten musste, wodurch die Zeitachse und die Messwerte nicht mehr korrespondierten bzw. Werte auf der Zeitachse ausgelassen wurden.

Um dieses Problem zu beheben, testeten wir, ob der PerformanceCounter auch bei lokal ermittelten Leistungsindikatoren zu langsam ist. Glücklicherweise war dies nicht der Fall. Wir versuchten also dieses Problem zu umgehen, indem wir die Leistungsindikatoren nicht mehr über die dafür vorgesehene Schnittstelle des entfernten Datenbankservers abgriffen, sondern einen ressourcenschonenderen *PerformanceServer* entwickelten, der lokal auf dem Datenbankserver ausgeführt wurde. Dieser sendete dann, auf eine Anfrage von GreenDB hin, die lokal ermittelten Leistungsindikatoren mittels UDP-Protokoll an den Messrechner. Nun können also mittels GreenDB und PerformanceServer die Messwerte der Leistungsindikatoren vom Datenbankserver zusammen mit dem vom LMG95 gelieferten Stromverbrauch in GreenDB erfasst und gespeichert werden.

Damit die verschiedenen Datenbankabfragen automatisiert hintereinander ablaufen konnten, verfügt GreenDB über ein Textfenster, in welches alle SQL-Befehle hintereinander, jeweils durch ein Pipe-Symbol (|) getrennt, geschrieben werden können. GreenDB arbeitet diese Befehle nun der Reihe nach ab, wobei es vor und nach jedem neuen Befehl eine Wartezeit von 10 Sekunden einhält. Während dieser Wartezeit werden die Leistungsindikatoren und der Stromverbrauch in Watt schon beziehungsweise weiter erfasst. Sinn dieses Vorgehens ist es, erstens vor jeder Abfrage zu sehen, ob sich der Datenbankserver im Leerlauf befindet, und zweitens nach jeder Abfrage eventuell durch „Aufräumarbeiten“ der Datenbank auftretende Effekte zu messen.

Um jeden Befehl auch auf einer gerade frisch gestarteten Datenbank messen zu können, implementierten wir eine Funktion, mit der die Datenbank vor jedem Befehl neu gestartet werden kann. Dies ließ sich sehr effektiv über das Programm *PerformanceServer* verwirklichen, da dieses die Datenbank lokal mit den dafür vorgesehenen Programmen und Parametern neu starten kann, wenn GreenDB einen entsprechenden Befehl sendet. Bei einer Kaltstartmessung wartet GreenDB solange, bis die Datenbank wieder verfügbar ist, und zusätzlich noch 20 weitere Sekunden, bevor ein neuer

SQL-Befehl zur Messung an die Datenbank gesendet wird.

Der Name einer Protokolldatei setzt sich wie folgt zusammen:

```
yyyyMMdd_HHmmss_fff.csv
```

Der Dateiname besteht aus dem Datum (amerikanisches Format) und der Uhrzeit in Stunden, Minuten, Sekunden und Millisekunden. Dieser Zeitstempel gibt den exakten Startzeitpunkt der Messung wieder.

Bei den ersten Messreihen mit Messintervallen von weniger als 0,5 Sekunden mussten wir leider feststellen, dass zwar die Werte der Leistungsindikatoren jetzt zur Verfügung standen, aber das LMG95 den Stromverbrauch nicht so schnell ermitteln bzw. übertragen konnte. Wir erhöhten also das Messinterval auf eine Sekunde, was bei den zu erwartenden, langen Laufzeiten der Abfragen durchaus zu vertreten ist. Die Server-Client-Konstruktion behielten wir dennoch bei, da über diese, wie bereits erwähnt, der Kaltstart gesteuert wird und sie keinen messbaren Einfluss auf den Stromverbrauch hat.

## 3.3 Verwendete Hardware

Der verwendete Server, auf dem die Datenbanksoftware Caché lief, war ein *PRIMERGY Econel 50 (Mono Prozessor Economy Server)* von Fujitsu Siemens mit folgenden Komponenten:

- **Mainboard:** D1859 (Chipsatz Intel 7221).
- **Prozessor:** Intel Pentium 4 mit 3,20 GHz (Modell-Nummer 541).
- **Arbeitsspeicher:** 4 GB (zwei DDR-2 800 Riegel mit jeweils 2 GB).
- **Festplatte:** Maxtor DiamondMax mit 300 GB Kapazität, 7200 RPM. Die mit *HD Tach* [1] durchschnittlich gemessene Lesegeschwindigkeit beträgt 53,6 MB/s. Die maximale Lesegeschwindigkeit beträgt 134,9 MB/s. Die Zugriffszeit (*Random access*) beträgt 14,2 ms.
- **Solid State Drive:** Eine SSD mit 128 GB Kapazität, angeschlossen über eine SATA II-Schnittstelle. Bei manchen Tests verwendeten wir die SSD statt der HDD (Maxtor DiamondMax-Festplatte). Die mit *HD Tach* durchschnittlich gemessene Lesegeschwindigkeit beträgt 122,2 MB/s. Die maximale Lesegeschwindigkeit beträgt 132,6 MB/s. Die Zugriffszeit (Random access) beträgt 0,2 ms.
- **Laufwerk:** CD/DVD-ROM Laufwerk.

[1] *HD Tach* (http://www.simplisoftware.com/) ist ein Programm mit welchem die Lesegeschwindigkeit einer Festplatte gemessen werden kann.

- **Netzteil:** HIPRO 300 Watt ATX-Netzteil.

- **Betriebssystem:** Windows Server 2008 SP2 (64-Bit) [2].

Der Grund für die Auswahl dieses Servers war, dass wir auf diesen Server über den gesamten Zeitraum des Schreibens unserer Arbeit uneingeschränkten Zugriff hatten. Für unsere Messungen war die Server-Performance akzeptabel, da der Energieverbrauch und nicht etwa die Geschwindigkeit des Rechners während der Messungen im Vordergrund stand.

Als Messgerät zur Aufzeichnung der Leistungsaufnahme des Servers wurde das *Einphasen Präzisions-Leistungsmessgerät LMG95* der Firma ZES ZIMMER Electronic Systems verwendet. Mit einer Grundgenauigkeit von 0,025 % ist es laut Herstellerangaben das genauste seiner Klasse und wird daher als Referenzgerät für Energiezähler, Leistungsmessumformer sowie für Effektivwerte von Strom und Spannung eingesetzt. Auch die Leistungsaufnahme von Geräten im Standby-Betrieb, welche bei den meisten Geräten im Bereich von 0,5 bis 2 Watt liegt, lässt sich mit dieser Grundgenauigkeit zuverlässig messen [LMG].

## 3.4 Gemessene Leistungsindikatoren

Das Programm *GreenDB* zeichnet folgende Leistungsindikatoren während eines Testlaufes auf [3]. Das Abtastintervall beträgt für alle Leistungsindikatoren bei unseren Versuchen eine Sekunde.

- **Laufzeit:** Die vergangene Zeit ab dem Abschicken eines SQL-Befehls bis zur Rückmeldung des Servers, dass der SQL-Befehl berechnet wurde und die Ergebnisse vorliegen. Die Zeit wurde in Millisekunden gemessen.

- **Prozessorzeit (Prozessor):** Dieser Wert gibt die prozentuale Angabe der verstrichenen Prozessorzeit an, die zum Ausführen von Threads benötigt wird. Die Prozessorzeit kann weiterhin in Benutzer- und Systemzeit aufgeteilt werden.

- **Benutzerzeit (Prozessor):** Die verstrichene Prozessorzeit im Benutzermodus in Prozent. Der Benutzermodus ist ein eingeschränkter Modus für Benutzeranwendungen. Das Betriebssystem schaltet Anwendungsthreads aus dem Benutzermodus in den privilegierten Modus um, um auf Betriebssystemdienste zugreifen zu können.

[2] Sämtliche für die Messungen nichts erforderlichen aktiven Betriebssytem-Dienste wurden deaktiviert, damit diese die Messungen nicht beeinflussen.

[3] Sämtliche Logdateien unserer Messungen befinden sich auf dem dieser Arbeit beiligenden Datenträger.

- **Systemzeit (Prozessor):** Der Prozentanteil der verstrichenen Zeit für Prozessthreads, die im privilegierten Modus ausgeführt werden. Wenn ein Windows-Systemdienst aufgerufen wird, wird dieser oft im privilegierten Modus ausgeführt, um Zugriff auf Systemdaten zu erhalten. Solche Daten sind vor dem Zugriff von Threads, die im Benutzermodus ausgeführt werden, geschützt.
- **Bytes gelesen/s (Physikalischer Datenträger):** Die Rate, mit der Bytes während Lesevorgängen vom Datenträger übertragen werden.
- **Bytes geschrieben/s (Physikalischer Datenträger):** Die Rate, mit der Bytes während Schreibvorgängen auf Datenträger geschrieben werden.
- **E/A-Lesevorgänge/s (Prozess):** Die Rate, mit der der Prozess E/A-Vorgänge (Lesevorgänge) zuweist. Dieser Indikator zählt alle vom Prozess generierten E/A-Aktivitäten, um Datei-, Netzwerk- und Geräte-E/As einzubeziehen.
- **E/A-Schreibvorgänge/s (Prozess):** Die Rate, mit der der Prozess E/A-Vorgänge (Schreibvorgänge) zuweist. Dieser Indikator zählt alle vom Prozess generierten E/A-Aktivitäten, um Datei-, Netzwerk- und Geräte-E/As einzubeziehen.
- **Leistungsaufnahme:** Die Leistungsaufnahme in Watt gemessen mit dem Einphasen-Präzisions-Leistungsmessgerät LMG95.

# 4 Implementierung

## 4.1 Einrichtung der Testumgebung

Für die Messungen wurden zwei Computer, ein Datenbankserver und ein Überwachungsrechner verwendet, die über ein Netzwerk miteinander verbunden kommunizieren konnten.

Auf dem Datenbankserver wurde das Betriebssystem Windows Server 2008 SP2 (64-Bit) installiert und alle Hintergrunddienste wurden deaktiviert, die nicht notwendigerweise vom Betriebssystem benötigt wurden. Auch wurde beispielsweise die automatische Updatefunktion von Windows deaktiviert, nachdem die neusten Updates eingespielt waren. Auf diese Weise sollten die Leistungsindikatoren nicht durch andere Prozesse als die der Datenbank beeinflusst werden.

Erste Beobachtungen im Taskmanager zeigten, dass die CPU-Auslastung des Datenbankservers bei dieser Konfiguration bei nahezu 0 % lag und auch keine Lese- oder Schreiboperationen auf der Festplatte durchgeführt wurden. Die später durchgeführten Idle-Messungen bestätigten diesen ersten Eindruck.

Im nächsten Schritt wurde die Datenbank *Caché* (Version 210.2.2.600.0) von *Intersystems* auf dem Datenbankserver aufgespielt und die Datenbank so mit dem Caché-Server-Manager eingerichtet, dass eine Verbindung vom Überwachungsrechner aufgebaut werden konnte. Zuletzt wurden der von uns entwickelte PerformanceServer auf dem Datenbankserver gestartet, und daraufhin getestet, ob dieser die CPU-Auslastung verändert. Dies war nicht der Fall.

## 4.2 GreenDB und PerformanceServer

Das Programm *GreenDB* sowie der PerformanceServer zum Erfassen der Leistungsparameter wurden vollständig in der Programmiersprache C# im *.NET-Framework 4.0* unter Verwendung von *Visual Studio 2010* programmiert. Das Programm *GreenDB* besteht aus den folgenden Klassen:

**Logger:** Diese Klasse bietet Methoden zum Öffnen und Schließen einer CSV-Datei sowie zum Schreiben von Zeilen in die CSV-Datei.

**LMG95SerialPort:** Mit Hilfe dieser Klasse kann eine COM-Verbindung über die serielle Schnittstelle des Überwachungsrechners zum LMG95 aufgebaut werden. Weiterhin gibt es eine Methode, bei deren Aufruf der aktuell vom LMG95 gemessene

Stromverbrauch des Datenbankservers ausgelesen werden kann.

**UDPClient:** Diese Klasse sendet Steuerbefehle mittels UDP-Protokoll an den PerformanceServer und wartet auf dessen Antwort.

**RemotePerfMon:** Diese Klasse stellt eine ODBC-Verbindung zum Datenbankserver her und sendet die in der Oberfläche von GreenDB eingegebenen SQL-Befehle nacheinander an die Datenbank. Die oben genannten Klassen werden hier verwendet, um den gemessenen Stromverbrauch und die Parameter (CPU-Auslastung, Lese -und Schreibzugriffe der Festplatte, Laufzeit der SQL-Abfrage) in jeweils einer CSV-Datei pro SQL-Abfrage zu speichern.

Das Programm PerformanceServer besteht aus den folgenden Klassen:

**PerfMonClient:** Diese Klasse wartet auf eingehende UDP-Verbindungen und liest deren Datenstrom aus. Anschließend sendet sie je nach empfangenem Steuerbefehl entweder die aktuellen Leistungsindikatoren des Datenbankservers an den Client zurück, oder führt einen Restart von Caché durch. Nach einem Restart wird ein Steuerbefehl an den Client gesendet. Der Restart wird durch den Aufruf der Datei „css.exe“ mit den Parametern „stopstart <instance name>“, in unserem Fall mit „stopstart Cacheweb“, durch den PerfMonClient realisiert.

**PerformanceCounter:** Diese Klasse stellt eine Komponente eines Windows NT-Leistungsindikators dar. Diese Klasse wurde nicht von uns entwickelt, sondern ist Teil des Namespace „System.Diagnostics“ der Assembly „System.dll“ von Windows. Mit Hilfe dieser Klasse können die CPU-Auslastung oder der Lese-und Schreibzugriff auf der Festplatte ermittelt werden.

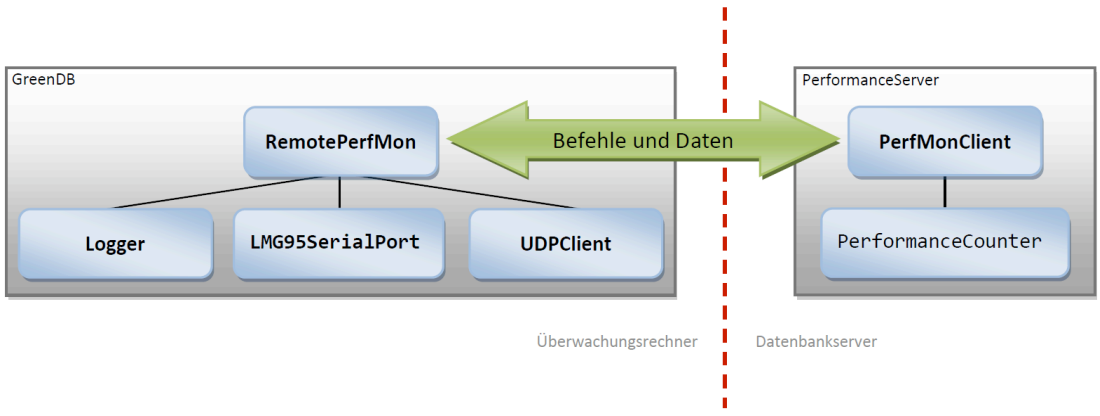



Abbildung 4.1: Das Client-Server-Modell.

Die Klasse Performancecounter wurde im PerformanceServer eingebunden, obwohl sie auch über die Möglichkeit verfügt, Daten direkt an einen entfernten Computer zu senden. Erste Versuche den PerformanceCounter direkt vom Überwachungsrechner anzusteuern, zeigten allerdings, dass der PerformanceCounter zeitweise nicht mehr reagierte und Messungen in gleichmäßigen Intervallen nicht möglich waren. Um Implementierungsfehler auszuschließen, wurde versucht mit dem Programm „Leistungsüberwachung“ von Microsoft die Leistungsindikatoren des Datenbankservers auf einem entfernten Computer auszulesen. Auch in diesem Fall konnten keine Messwerte in gleichmäßigen Abständen ermittelt werden. Aus diesem Grund wurde das Server-Client-Modell aus GreenDB und PerformanceServer entwickelt.

Die Übertragung der Leistungsindikatoren erfolgt hier direkt auf Anforderung, so dass eine Messung im Sekundentakt möglich ist. Hierfür wartet der PerformanceServer auf einen Befehl von GreenDB, um die Leistungsindikatoren zu ermitteln. In diesem Moment liest der PerformanceServer die Leistungsindikatoren lokal auf dem Datenbankserver aus und sendet die Werte zurück an GreenDB auf dem Messrechner. Um erkennen zu können, ob sich der Datenbankserver vor dem Absenden und nach der Bearbeitung eines SQL-Befehls im Idle-Zustand befindet, wartet GreenDB vor und nach jedem SQL-Befehl 10 Sekunden, bevor ein neuer Befehl gesendet wird. Diese Wartezeit ist besonders gut in den Diagrammen von SQL-Befehlen mit kurzen Laufzeiten sichtbar, wie Abbildung 4.2 zeigt.

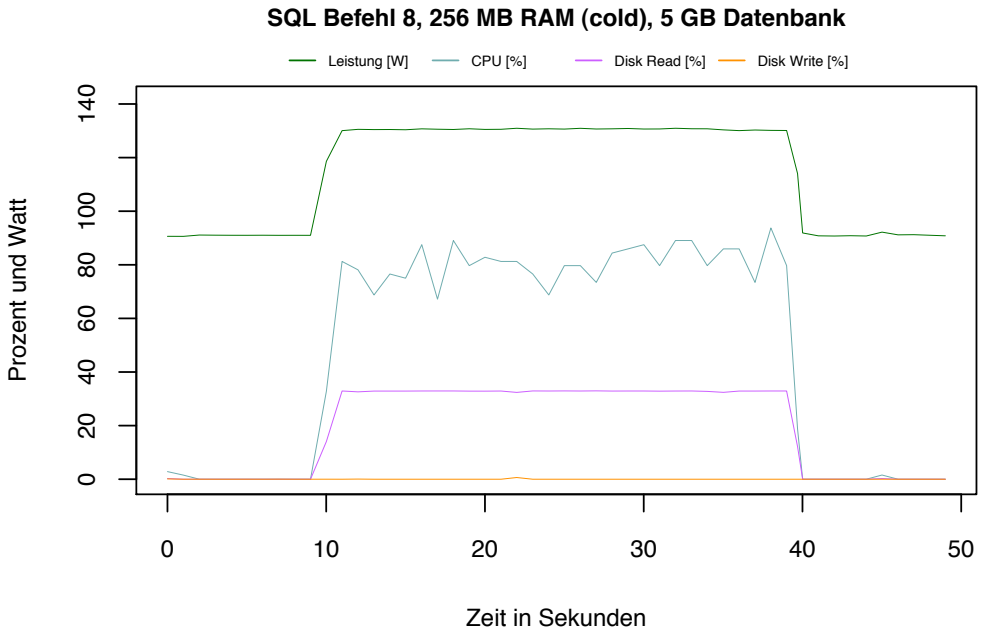



Abbildung 4.2: Diese Grafik verdeutlicht die 10 Sekunden Wartezeit vor und nach der SQL-Abfrage.

Die Verbindung zu Caché wurde über die Open Database Connectivity-Schnittstelle (ODBC) mittels der Klasse „OdbcConnection“ hergestellt. Zum Absenden eines SQL-Befehls wurde die Methode „ExecuteNonQuery()“ statt „ExecuteQuery()“ verwendet. Diese Befehle unterscheiden sich dadurch voneinander, dass „ExecuteNonQuery()“ nach Bearbeitung des SQL-Befehles nur die Anzahl der betroffenen Datensätze zurückgibt, der Befehl „ExecuteQuery()“ hingegen die gesuchten Datensätze selbst. Da das Zurückliefern der Datensätze (bei einem einfachen *Crossjoin* können dies beispielsweise mehrere Gigabyte sein) über das Netzwerk bei ersten Testdurchläufen zu lange dauerte und für unsere Messungen nur die Ausführungszeiten von Abfragen auf dem Datenbankserver von Interesse waren, haben wir uns für die Methode „ExecuteNonQuery()“ entschieden. Was die Ausführungszeiten der Datenbank für SQL-Befehle betrifft, haben Versuche gezeigt, dass es keinen Unterschied zwischen den beiden Methoden gibt.

## 4.3 Daten-Import in Caché

Der nächste Schritt bestand darin, die mit Hilfe des durch das Transaction Processing Performance Council zur Verfügung gestellten Programmes *DBGEN* erzeugten Daten in Caché zu importieren. Hierfür verwendeten wir das Systemmanagement-Portal von Caché, welches über den „blauen Würfel“ in der Taskleiste gestartet werden kann.

Um die Daten importieren zu können, muss zunächst eine Datenbank angelegt und mit einem Namespace verbunden werden. Dies erfolgt im Menü „[Startseite] > [Konfiguration] > [Lokale Datenbanken]“ unter „Neue Datenbank erstellen“. Der Namespace wird im Menü „[Startseite] > [Konfiguration] > [Namespaces]“ erzeugt und mit der Datenbank verbunden. Nach diesem Schritt ist die Datenbank über den Namespace-Namen erreichbar, allerdings enthält sie noch keine Tabellen. Um die Tabellen mit den richtigen Attributen zu erzeugen, nutzen wir die ebenfalls von *DBGEN* erzeugte DDL-Datei. In dieser Datei befinden sich die SQL-Befehle zum Erzeugen der Tabellen. Diese Befehle können, nach Auswahl des richtigen Namespace, nacheinander im Abfragefenster im Menü „[Startseite] > [SQL] > [SQL-Abfrage ausführen]„ ausgeführt werden.

Für die Testreihen ohne Primary-Key und Foreign-Key ist nach diesem Schritt die Datenbank fertig eingerichtet. Für die Testreihen mit Schlüsseln müssen nach diesem Schritt und **vor dem Import der Daten** im Studio die Schlüssel angelegt werden. Details zum Anlegen der Schlüssel und zur Modellierung des TPC-H Schemas in Caché sind in Abschnitt 6.1.2 angegeben.

Der eigentliche Datenimport erfolgt über das Systemmanagement-Portal im Menü „[Startseite] > [SQL]“ unter „Assistent für Datenimport“. Die von *DBGEN* erzeugten Datein mit den Rohdaten, können über den Assistenten für den Datenimport in die Datenbank eingelesen werden. Abbildung 4.3 zeigt, welche Einstellungen für den Import-Vorgang gewählt wurden.

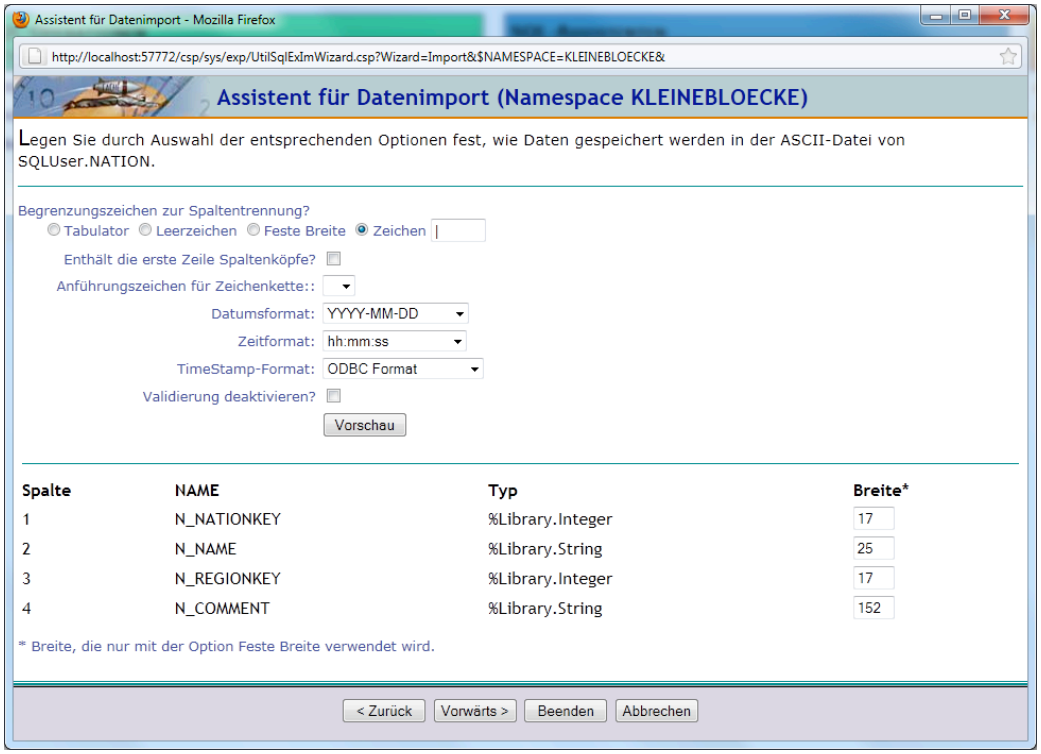



Abbildung 4.3: Datenimport.

## 4.4 Einrichten der SSD

Um die Auswirkungen beim Einsatz einer SSD anstelle einer herkömmlichen Festplatte auf den Stromverbrauch zu untersuchen, wurde Windows 2008-Server mit exakt der gleichen Konfiguration auf der SSD installiert und eingerichtet wie zuvor auf der HDD. Anschließend wurde Caché installiert und die Datenbankdatei auf die SSD kopiert und in Caché gemounted.

Die Auswertung der Ergebnisse zeigte bereits, dass sich der durchschnittliche Energieverbrauch durch den Einbau der SSD um ca. 10 Watt senken ließ und sich die Ausführungszeit der meisten Abfragen verkürzte. Dieses Einsparpotential wurde erreicht, ohne das weitere Optimierung der Datenbank vorgenommen wurden und kann daher direkt auf die Technologie der SSD zurückgeführt werden.

Für weitere Experimente wurde die Datenbank so konfiguriert, dass diese die Vorteile einer SSD optimal ausnutzt. Hier bestand die einzige Optimierungsmöglichkeit für uns darin, die Blockgröße der Datenbank so zu wählen, dass diese besser mit der Blockgröße der SSD harmoniert. Die Idee dahinter ist, dass die Datenbank mehr Informationen pro Lesevorgang erhalten kann, wodurch sich die Bearbeitungszeit einer Abfrage verkürzt. Um dies zu testen, wählten wir statt der Standard-Datenbankblockgröße von 8 KB in weiteren Messungen 2 KB und 64 KB große Blöcke. Da diese Einstellung aber

aus Sicherheitsgründen nicht über das Portal von Caché eingestellt werden kann (für herkömmliche HDDs ist die Vorauswahl von 8 KB optimal), müssen die folgenden, nicht in der Caché-Dokumentation enthaltenen, Schritte durchgeführt werden um die Datenbank-Blockgröße zu ändern.

1. Öffnen der Datei „Cache.cpf“ im Homeverzeichnis von Cache mit einem Texteditor.

2. Suche des Abschnitts „[Startup]“ und des darin enthaltenen Schlüssels „DBSizesAllowed=8192“. Hier werden, durch Komma getrennt, alle Blockgrößen in KB angegeben, die Caché unterstützen soll. Es werden maximal Blockgrößen von bis zu 64 KB unterstützt. Damit Caché die Erstellung von Datenbanken mit den Blockgrößen 2 KB, 8 KB und 64 KB unterstützt, ist der Eintrag wie folgt zu ändern: „DBSizesAllowed=2048,8192,65536“

3. Suche des Abschnitts „[Config]“ und den darin enthaltenen Schlüssels „globals=0,0,3072,0,0,0“. Dieser Eintrag bedeutet, dass Caché für 8 KB große Globals 3072 MB Arbeitsspeicher zur Verfügung stehen. Die sechs Positionen in der Zeichenfolge stehen für die Blockgrößen von 2 KB, 4 KB, 8 KB, 16 KB, 32 KB und 64 KB. Die dort eingetragene Zahl steht für den für diese Blockgröße reservierten Speicher. Um für 64 KB große Globals einen Speicherbereich von 3 GB zu reservieren, muss der Eintrag wie folgt geändert werden: „globals=0,0,0,0,0,3072“. Für 2 KB große Globals wird der Eintrag wie folgt geändert: „globals=3072,0,0,0,0,0“.

Nach diesen Vorbereitungen kann eine neue Datenbankdatei angelegt werden, welche die neue Konfiguration verwendet. Hierzu kann das Datenbank-Terminal (siehe Abbildung 4.4) von Caché verwendet werden, da diese Funktion nicht über das Management-Portal zur Verfügung gestellt wird. Im Terminal müssen dazu folgende Schritte ausgeführt werden, um eine Datenbank mit einer bestimmten Blockgröße anzulegen.

1. Start des Terminals über den „blauen Würfel“ in der Taskleiste und Wechsel des Namespaces mit dem Befehl „zn ”SYS”“.

2. Eingabe des Befehls „do ^DATABASE“, um ein Konsolenprogramm zum Verwalten der Datenbanken zu starten.

3. Auswahl des Menüpunkts „1) Create a database“, um eine neue Datenbank anzulegen

4. Eingabe des Speicherpfads der Datenbank.

5. Auswahl von „Y“ bei „Change default database properties?“, um weiter Einstellungen vornehmen zu können.

6. Auswahl des Menüpunkts „2)* Block size (bytes):“ und der Blockgröße 65.536 KB.

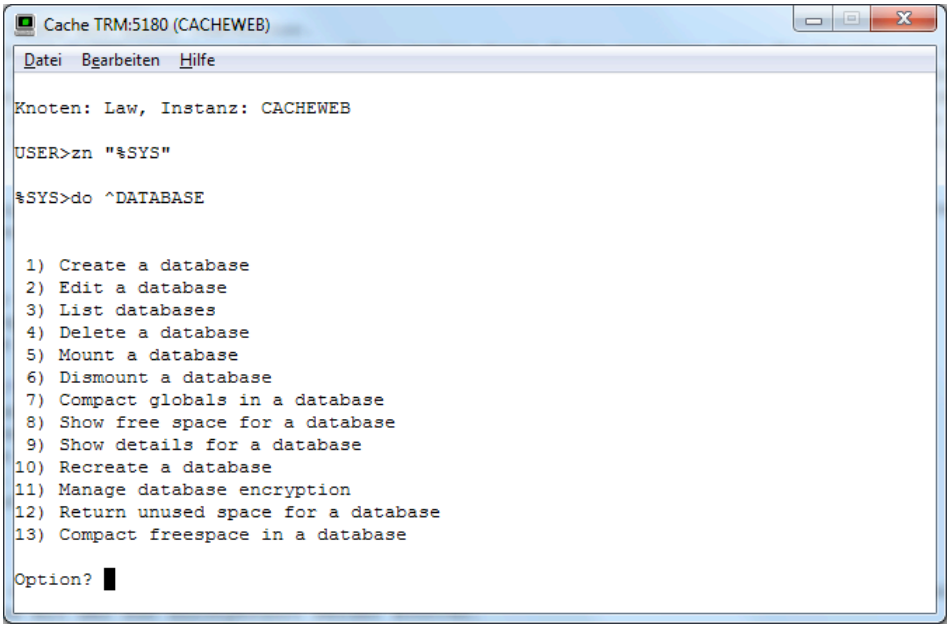



Abbildung 4.4: Das Caché Datenbank-Terminal.

7. Auswahl des Menüprunkts „4) Current size (MB)“ und Eingabe der zu erwartenden Datenbankgröße.

8. Bestätigung der Einstellung und Eingabe des Datenbanknamens.

Nach diesen Schritten und dem Anlegen der Datenbank über das Terminal kann die auf diese Weise erzeugte Datenbank über das Systemmanagement-Portal von Caché verwaltet werden. Nach dem anschließenden Import der Rohdaten konnten die Messreihen mit unterschiedlichen Datenbank-Blockgrößen auf der SSD durchgeführt werden.

# 5 Energieverbrauch der Datenbank

Dieses Kapitel liefert Erkenntnisse über die Auswirkungen von unterschiedlichen Arbeitsspeicher- und Datenbankgrößen auf den Energieverbrauch der Datenbank Caché. Weiterhin wird untersucht, welchen Einfluss die Arbeitsspeicher- und Datenbankgröße auf unterschiedliche Typen von SQL-Befehle hinsichtlich des Energieverbrauchs hat. Vor dieser Untersuchung wird zunächst die Leistungsaufnahme des Servers im Leerlauf bestimmt, damit sich alle weiteren Ergebnisse besser einordnen lassen. Für die Messungen in Kapitel 6 wird dann eine festgelegte Konfiguration von Arbeitsspeicher- und Datenbankgröße gewählt, damit andere Einflussfaktoren auf den Energieverbrauch untersucht werden können.

## 5.1 Leistungsaufnahme des Servers im Leerlauf

Um die Leistungsaufnahme in Watt des Datenbankservers im Leerlauf (*Idle*) zu ermitteln, wurden vier zehnstündige Messungen durchgeführt. Neben der Leistungsaufnahme in Watt, wurde gleichzeitig die Prozessornutzung und Festplattenaktivität aufgezeichnet. Bei zwei dieser Messungen war die Datenbank-Software Caché deaktiviert und bei zwei Durchläufen war Caché aktiviert, jedoch ohne durch SQL-Anfragen belastet zu werden. Ziel der Messungen war es, die Leistungsaufnahme des Servers mit und ohne Caché im Leerlauf zu ermitteln. Der Zeitraum jedes der vier Testläufe betrug zehn Stunden. Das Messintervall betrug eine Sekunde, was zu 36.000 Datensätzen (10 Stunden × 60 Minuten × 60 Sekunden = 36.000) für einen Testlauf führte.

| Testlauf | Min. | 0,25-Quantil | Median | ∅ | 0,75-Quantil | Max. |
|---|---|---|---|---|---|---|
| mit Caché | 89,10 | 90,28 | 90,59 | 90,60 | 90,93 | 120,50 |
| mit Caché | 89,97 | 90,06 | 90,38 | 90,39 | 90,71 | 117,40 |
| ohne Caché | 89,53 | 90,14 | 90,38 | 90,48 | 90,71 | 142,00 |
| ohne Caché | 89,55 | 90,10 | 90,37 | 90,46 | 90,73 | 116,60 |

Tabelle 5.1: Leistungsaufnahme (in Watt) des Servers im Leerlauf.

Die Ergebnisse und statistische Kennzahlen bezüglich der Messungen sind in Tabelle 5.1 aufgeführt. Für die beiden Testläufe mit **aktivierter Datenbanksoftware** liegen die Durchschnittswerte bei 90,60 Watt und 90,39 Watt. Bei **deaktivierter Datenbanksoftware** liegen die Durchschnittswerte mit 90,48 Watt und 90,46 Watt in einem nahezu identischen Bereich. Ein relevanter Unterschied zwischen aktivierter und deaktivierter Datenbanksoftware konnte nicht festgestellt werden.

Der bei allen Testläufen kleinste gemessene Wert liegt nahe am Durchschnittswert. Demzufolge gibt es keine Ausreißer nach unten. Die meisten Werte liegen sehr dicht beieinander, und vor allem die Werte im Bereich des unteren Quantils bis zum oberen Quantil liegen in keinem Fall um einen Wert größer als 0,65 Watt auseinander.

Der Durchschnittswert ist gegenüber dem Median ein wenig nach oben verschoben, was darauf hindeutet, dass es Ausreißer nach oben hin gibt, welche zwar den Durchschnitt nach oben hin beeinflussen, nicht jedoch den Median in vergleichbarem Maße. Tatsächlich wurden vereinzelte Werte deutlich über dem Durchschnitt gemessen (siehe Max-Werte in 5.1). Die Anzahl solcher Ausreißer ist gering. Bei jeweils 36.000 Werten sind es im ersten Durchlauf 11, im zweiten Durchlauf 29, im dritten Durchlauf 64 und im letzten Durchlauf lediglich 76 Werte über 100 Watt.

In denjenigen Datensätzen der Logdateien, in denen die Leistungsaufnahme deutlich über der durchschnittlichen Leistungsaufnahme liegt, lässt sich eine deutliche Festplatten- und Prozessoraktivität erkennen. Um solche ungewollten Betriebssystem-Aktivitäten zu vermeiden, wurde das Betriebssystem nur auf die notwendigen Dienste beschränkt. Gänzlich verhindern ließen sich solche Systemaktivitäten jedoch nicht. In einem realistischen Anwendungsszenario sind jedoch häufig weitere Dienste aktiv und es ist mit einer erhöhten Anzahl solcher Betriebssystem-Aktivitäten, welche den Durchschnittswert leicht nach oben verschieben, zu rechnen.

## 5.2 Entwurf

Um die Leistungsaufnahme und die Ressourcennutzung (Prozessor-, Festplatten-, Arbeitsspeichernutzung) des Server unter Last zu verstehen, wurden 30 SQL-Befehle formuliert, welche verschiedene Anforderungen an eine Datenbank abdecken. Diese SQL-Befehle wurden möglichst einfach aufgebaut, damit eine Interpretation der Messergebnisse hinsichtlich der Leistungsaufnahme und der Hardwarenutzung eines SQL-Befehls leichter gelingt. Insbesondere konnte genau bestimmt werden, auf welchen Datenmengen die Abfragen operieren (vgl. Tabellen 5.2, 5.3 und 5.4 im nachfolgenden Abschnitt). Aufgrund der in diesem Kapitel gewonnenen Erkenntnisse, wird in Kapitel 6 eine hinsichtlich des Energieverbrauchs geeignete Arbeitsspeichergröße gewählt.

In Kapitel 6 wird als Workload dann der TPC-H Benchmark und nicht mehr die Sammlung der 30 SQL-Abfragen aus diesem Kapitel verwendet. Der TPC-H Benchmark simuliert ein realistisches Anwendungsszenario, hat aber den Nachteil, dass die im TPC-Benchmark verwendeten SQL-Befehle, verglichen mit den in diesem Kapitel verwendeten Befehlen, recht komplex sind und sich nicht auf einzelne Bausteine reduzieren lassen.

Die Sammlung der 30 SQL-Befehle enthält einfache Abfragen, welche Ergebnismen-

gen mit eindeutigen Werten (DISTINCT-Prädikat) ausgeben, Abfragen mit Filtern (WHERE-Klauseln), Abfragen mit verknüpften Tabellen (JOIN-Operator), den Datenbestand verändernde Anweisungen (UPDATE-Anweisung), Sortierungen (SORT BY-Klausel), Gruppierungen (GROUP BY-Klausel) oder Aggregat-Funktionen (SUM-, AVG-, COUNT-Funktionen).

Diese 30 SQL-Befehle wurden bei unterschiedlichen Systemkonfigurationen ausgeführt. Die Tabellengrößen, also die Zeilenanzahl und die tatsächlichen Größen auf dem Datenträger, sind genau bekannt, so dass diese als veränderliche Parameter in die Messungen miteinfließen konnten. Ein weiterer veränderlicher Parameter unserer Tests war die gesamte Datenbankgröße. Als Datenbankgrößen wurden 1 GB, 5 GB und 10 GB gewählt. Der Arbeitsspeicher, welcher dem Datenbankmanagementsystem (DBMS) zugewiesen wurde, wurde auf 256 MB beziehungsweise 3 GB eingestellt. Die Versuche wurden im sogenannten *Kaltstart*-Betrieb durchgeführt.

**Kaltstart-Betrieb** bedeutet, dass der Inhalt des Arbeitsspeichers vor dem Abschicken einer neuen Abfrage geleert wird. Dies sichert die Vergleichbarkeit der Messergebnisse zu. Wird eine neue Abfrage an den Datenbankserver geschickt, kann das System also nicht auf bereits im Arbeitsspeicher vorhandene Daten zugreifen und damit I/O vermeiden. Dem gegenüber steht der *Warmstart-Betrieb*.

Im **Warmstart-Betrieb** können sich die Daten einer Tabelle, welche für eine Abfrage benötigt werden, bereits im Arbeitsspeicher befinden und sie müssten in diesem Fall nicht von der Festplatte eingelesen werden. Bei dieser Betriebsart lässt sich nicht immer ermitteln, welche Daten sich bereits im Arbeitsspeicher befinden, und eine Vergleichbarkeit von Messergebnissen ist deshalb nicht gewährleistet.

### 5.2.1 Datenmodell

Die Tests wurden mit drei unterschiedlichen Datenbankgrößen (1 GB, 5 GB und 10 GB) durchgeführt. Die Datensätze der Datenbank wurden mit dem Programm *DBGEN*, bereitgestellt vom *Transaction Processing Performance Council (TPC)*[1] generiert. Über einen Skalierungsfaktor können über dieses Programm unterschiedlich große Datenbestände erzeugt werden. Die Namen der acht Relationen, welche durch das Programm erzeugt wurden, sind, absteigend nach Größe sortiert, *lineitem*, *orders*, *partsup*, *customer*, *part*, *supplier*, *nation* und *region*.

Die mit *DBGEN* generierten Rohdaten erfüllen die Anforderungen der TPC-H-Spezifikation [TCP]. In einer Datums-Spalte muss beispielsweise ein gültiges Datum stehen und dieses Datum muss in einer von dem Programm vorgegebenen

[1] Das Transaction Processing Performance Council (TPC) wurde 1988 von verschiedenen weltweit führenden IT-Unternehmen gegründet. Ziel des TPC ist die Leistungsbewertung von Datenbankmanagementsystemen mithilfe standardisierter Benchmarks. Die Internetseite des Konsortiums ist http://www.tpc.org/.

Zeitperiode liegen. Für Text-Spalten ist entweder die maximal zulässige Anzahl der Zeichen vorgegeben (für zufällige Texte) oder es wird ein Wort aus einer vorgegebenen Sammlung von Wörtern (zum Beispiel „1-URGENT“, „2-HIGH“, „3-MEDIUM“, „4-NOT SPECIFIED“, „5-LOW“) zufällig ausgewählt. Ein Auszug von fünf Datensätzen (ein Datensatz ist aus Platzgründen hier jeweils auf zwei Zeilen aufgeteilt) der generierten Daten für beispielsweise die Relation *orders* sieht wie folgt aus:

```
1|3691|O|194029.55|1996-01-02|5-LOW|Clerk#000000951|0|nstructions
    sleep furiously among |
2|7801|O|60951.63|1996-12-01|1-URGENT|Clerk#000000880|0| foxes.
    pending accounts at the pending, silent asymptot|
3|12332|F|247296.05|1993-10-14|5-LOW|Clerk#000000955|0|sly final
    accounts boost. carefully regular ideas cajole carefully. depos|
4|13678|O|53829.87|1995-10-11|5-LOW|Clerk#000000124|0|sits. slyly
    regular warthogs cajole. regular, regular theodolites acro|
5|4450|F|139660.54|1994-07-30|5-LOW|Clerk#000000925|0|quickly.
    bold deposits sleep slyly. packages use slyly|
```

Die Daten enthalten beispielsweise Texte (Caché-Datentyp *String*), ganzzahlige Werte (Caché-Datentyp *Integer*), Gleitkommazahlen (Caché-Datentyp *Numeric*) oder Datumswerte (Caché-Datentyp *Date*) und decken damit einen großen Bereich an möglichen Datentypen in einer Datenbank ab.

Wie die mit *DBGEN* erzeugten Rohdaten in die Datenbank Caché importiert werden können, wurde bereits in Abschnitt 4.3 beschrieben. Anzumerken ist an dieser Stelle, dass Caché jeder Relationen eine zusätzliche ID-Spalte hinzufügt, über welche jeder Datensatz eindeutig identifizierbar ist. Diese ID-Spalten werden in diesem Kapitel als Primärschlüssel verwendet.

Der prozentuale Anteil der Größe einer Relation bezüglich der gesamten Datenbankgröße bleibt bei allen Skalierungsfaktoren gleich. Diese Aussage gilt nicht für die Relationen *nation* und *region*, da diese Relationen eine feste Größe beziehungsweise Zeilenanzahl haben, die nicht mit der Datenbankgröße skaliert. Die Relation *nation* enthält 25 Zeilen und repräsentiert 25 Länder, und die Relation *region* repräsentiert 5 Kontinente beziehungsweise Regionen.

Die Tabellen 5.2, 5.3 und 5.4 geben Auskunft über die Zeilenanzahl und die Größen aller Relationen. Die Spalte „Anteil (in %)“ gibt den prozentualen Anteil einer Tabelle an der Datenbankgröße an.

### 5.2.2 Workload

Der verwendete Workload besteht aus einer Sammlung von insgesamt 30 SQL-Befehlen. Diese Befehle sind in sieben logische Gruppen unterteilt. Jede Gruppe repräsentiert

| Tabelle | Zeilenanzahl | Größe (in KB) | Anteil (in %) |
|---|---|---|---|
| lineitem | 6.001.215 | 747.915 | 69,0 |
| orders | 1.500.000 | 169.387 | 15,6 |
| partsupp | 800.000 | 116.978 | 10,7 |
| customer | 150.000 | 23.923 | 2,2 |
| part | 200.000 | 23.765 | 2,2 |
| supplier | 10.000 | 1.386 | $< 1$ |
| nation | 25 | 3 | $< 1$ |
| region | 5 | 1 | $< 1$ |

Tabelle 5.2: Kenndaten der 1 GB-Datenbank (Größe auf dem Datenträger: 1.083.358 KB).

| Tabelle | Zeilenanzahl | Größe (in KB) | Anteil (in %) |
|---|---|---|---|
| lineitem | 29.999.795 | 3.801.935 | 69,2 |
| orders | 7.500.000 | 857.117 | 15,6 |
| partsupp | 4.000.000 | 589.677 | 10,7 |
| customer | 750.000 | 120.002 | 2,2 |
| part | 1.000.000 | 119.262 | 2,2 |
| supplier | 50.000 | 6.971 | $< 1$ |
| nation | 25 | 3 | $< 1$ |
| region | 5 | 1 | $< 1$ |

Tabelle 5.3: Kenndaten der 5 GB-Datenbank (Größe auf dem Datenträger: 5.494.968 KB).

| Tabelle | Zeilenanzahl | Größe (in KB) | Anteil (in %) |
|---|---|---|---|
| lineitem | 59.979.500 | 7.652.065 | 69,2 |
| orders | 15.000.000 | 1.722.847 | 15,6 |
| partsupp | 8.000.000 | 1.184.425 | 10,7 |
| customer | 1.500.000 | 240.574 | 2,2 |
| part | 2.000.000 | 239.594 | 2,2 |
| supplier | 100.000 | 13.942 | $< 1$ |
| nation | 25 | 3 | $< 1$ |
| region | 5 | 1 | $< 1$ |

Tabelle 5.4: Kenndaten der 10 GB-Datenbank (Größe auf dem Datenträger: 11.053.451 KB).

einen bestimmten Abfragetyp.

1. **Aggregat-Funktionen:** Die erste Gruppe bilden die Befehle 1 bis 3. Diese Befehle berechnen einfache Aggregat-Funktionen wie Anzahl (COUNT), Summe (SUM) und Mittelwert (AVG).

2. **Gruppierungen:** Die zweite Gruppe besteht aus den Befehlen 4 und 5, welche eine Gruppierung enthalten. Bei Befehl 4 steht allein die Gruppierung mittels GROUP BY im Mittelpunkt. Befehl 5 kombiniert schließlich Befehl 1 (Aggregationsfunktion) und Befehl 4 (Gruppierung).

3. **Sortierungen:** Die dritte Gruppe besteht aus den Befehlen 6 und 7, welche eine Sortierung durch ORDERED BY enthalten. Befehl 7 kombiniert wiederum eine Sortierung mit einer Aggregationsfunktion und einer Gruppierung.

4. **Selektionen:** Die Befehle 8 bis 13 bilden die vierte Gruppe. Diese Gruppe ist dadurch gekennzeichnet, dass durch Bedingungen (WHERE-Klauseln) eine Auswahl von Zeilen für die Ergebnismenge festgelegt wird. Die Bedingungen werden auf Spalten mit unterschiedlichen Datentypen (in Caché gibt es beispielsweise die Datentypen *Integer*, *Numeric*, *Date* und *String*) berechnet. Befehl 13 enthält zusätzlich eine Sortierung.

5. **Duplikate:** Befehl 14 bildet eine weitere Gruppe. Mit diesem Befehl soll die Auswirkung des DISTINCT-Schlüsselwortes auf den Energieverbrauch untersucht werden.

6. **Verbund:** Befehle 15 bis 26 bilden die Gruppe der Befehle, welche Verknüpfungen von Tabellen (JOIN-Operator) enthalten. Bei diesen Abfragen wurde Wert auf eine möglichst große Vielfalt unterschiedlicher Größen der involvierten Tabellen gelegt. Bei den Befehlen ohne WHERE-Klausel lässt sich anhand der Tabellen 5.2, 5.3 und 5.4 exakt die Zeilenanzahl der Ergebnismenge nach einer Verknüpfung zweier Tabellen bestimmen. Bei den Befehlen mit WHERE-Klauseln wurde darauf geachtet, dass diese Klauseln die jeweiligen Tabellen auf ein Bruchteil verkleinern. Für die 1 GB-Datenbank sind die Zeilenanzahlen genau angegeben.

7. **Updates:** Befehl 27 bis 30 bilden schließlich die Gruppe der Update-Befehle. Anhand dieser Befehle sollen das Verhalten und der Energieverbrauch der Datenbank bei Update-Operationen auf unterschiedlich großen Tabellen untersucht werden.

### 5.2.3 Die 30 SQL-Befehle im Detail

Die folgende Übersicht gibt die SQL-Syntax der 30 Befehle an. Die Beschreibung zu jedem Befehl gibt Auskunft über die Größe (prozentualer Anteil an der gesamten

Datenbank) der beteiligten Relationen. Die ersten 14 Befehle operieren auf der Relation *lineitem* und durch die Einfachheit der Abfragen kann nachvollzogen werden, wie groß die Datenmenge ist, auf der operiert wird. Die Auswirkung einer Veränderung der Arbeitsspeichergröße und eines Wechsels der Datenbankgröße kann so leichter nachvollzogen werden. Bei manchen der 30 Befehle ist zusätzlich angegeben, wie viele Tupel die Ergebnismenge enthält. Insbesondere für die Abfragen mit verknüpften Tabellen erlaubt dies eine Abschätzung, wie viele Zeilen die Verknüpfung enthält.

**SQL Befehl 1:** Berechnet die Zeilenanzahl der Tabelle *lineitem*. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM lineitem
```

**SQL Befehl 2:** Berechnet die Summe der Werte in der Spalte *l_quantity* aus der Tabelle *lineitem*. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT sum(l_quantity) FROM lineitem
```

**SQL Befehl 3:** Berechnet den Durchschnitt der Werte in der Spalte *l_discount* aus der Tabelle *lineitem*. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT avg(l_discount) FROM lineitem
```

**SQL Befehl 4:** Gruppiert den Inhalt der Tabelle *lineitem* nach der Spalte *l_shipmode*. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus. Die Spalte *l_shipmode* enthält 7 verschiedene Werte.

```
SELECT * FROM lineitem GROUP BY l_shipmode
```

**SQL Befehl 5:** Berechnet die Zeilenanzahl der Tabelle *lineitem* und gruppiert das Ergebnis nach der Spalte *l_shipmode*. Die Spalte *l_shipmode* enthält 7 verschiedene Werte. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT count(*) AS Anzahl, l_shipmode FROM lineitem
GROUP BY l_shipmode
```

**SQL Befehl 6:** Berechnet eine Sortierung der Tabelle *lineitem* nach der Spalte *l_quantity*. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT * FROM lineitem ORDER BY l_quantity
```

**SQL Befehl 7:** Dieser SQL-Befehl kombiniert Berechnungen welche schon in den vorherigen SQL-Befehlen durchgeführt wurden. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT count(*) AS Summe, l_shipmode FROM lineitem
GROUP BY l_shipmode ORDER BY l_quantity
```

**SQL Befehl 8:** Selektion mit einer WHERE-Bedingung auf zwei Zahlenspalten der Tabelle *lineitem*. Die WHERE-Klausel beschränkt beispielsweise die Tabelle *lineitem* auf 295 Zeilen in der 1 GB-Datenbank. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT * FROM lineitem
WHERE l_extendedprice > 16000 AND l_extendedprice < 16005
```

**SQL Befehl 9:** Selektion mit einer WHERE-Bedingung auf der Datumspalte *l_shipdate* der Tabelle *lineitem* unter Verwendung der *ToDate*-Funktion. Die WHERE-Klausel beschränkt beispielsweise die Tabelle *lineitem* auf 105 Zeilen in der 1 GB-Datenbank. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT * FROM lineitem
WHERE l_shipdate < ToDate('1992-01-05','YYYY-MM-DD')
```

**SQL Befehl 10:** Selektion mit einer WHERE-Bedingung auf der Datumpalte *l_shipdate* der Tabelle *lineitem* unter Verwendung der *DateAdd*-Funktion. Dieser SQL-Befehl entspricht inhaltlich dem vorherigen SQL-Befehl. Die WHERE-Klausel beschränkt beispielsweise die Tabelle *lineitem* auf 105 Zeilen in der 1 GB-Datenbank. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT * FROM lineitem
WHERE l_shipdate < DateAdd('m',1,'1991-12-05')
```

**SQL Befehl 11:** Selektion mit einer WHERE-Bedingung auf der Spalte *l_comment* der Tabelle *lineitem* unter Verwendung der *Substring*-Funktion. Es wird untersucht, ob der Wert der Spalte *l_comment* im Bereich der ersten 21 Zeichen den Text „unusual, final excuse“ enthält. Die WHERE-Klausel beschränkt beispielsweise die Tabelle *lineitem* auf 6 Zeilen in der 1 GB-Datenbank. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT * FROM lineitem WHERE substring(l_comment from 1 for 21)
IN ('unusual, final excuse')
```

**SQL Befehl 12:** Selektion mit einer WHERE-Bedingung auf der Spalte *l_comment* der Tabelle *lineitem* unter Verwendung von LIKE. Es wird untersucht, ob der Wert der Spalte *l_comment* den Text „ges sleep after the“ enthält. In der 1 GB-Datenbank gibt es beispielsweise keine Zeile, in welcher die Spalte *l_comment* den vorgegebenen Text enthält. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT * FROM lineitem
WHERE l_comment LIKE '\%ges sleep after the\%'
```

**SQL Befehl 13:** Selektion mit einer WHERE-Bedingung auf der Spalte *l_comment* der Tabelle *lineitem* unter Verwendung von LIKE. Es wird untersucht, ob der Wert der Spalte *l_comment* den Text „ges sleep after the“ enthält. Das Ergebnis wird sortiert ausgegeben. In der 1 GB-Datenbank gibt es beispielsweise keine Zeile, in welcher die Spalte *l_comment* den vorgegebenen Text enthält. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT * FROM lineitem WHERE l_comment
LIKE '\%ges sleep after the\%' ORDER BY l_quantity
```

**SQL Befehl 14:** Eine Projektion auf der Tabelle *lineitem*. Ausgegeben wird die Spalte *l_shipmode*, welche zusätzlich nur distinkte Werte enthalten soll. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
SELECT DISTINCT l_shipmode FROM lineitem
```

**SQL Befehl 15:** Berechnet die Zeilenanzahl nach einem Join der Tabellen *lineitem* und *orders*. Zusätzlich wird eine Selektion auf einer Datumsspalte und einer numerischen Spalte berechnet. Die WHERE-Klausel beschränkt beispielsweise die Tabelle *orders* auf 12 Zeilen und die Tabelle *lineitem* auf 105 Zeilen in der 1 GB-Datenbank. Die verwendeten Tabellen machen jeweils 69 % (*lineitem*) und 16 % (*orders*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM lineitem, orders
WHERE lineitem.l_shipdate < ToDate('1992-01-05','YYYY-MM-DD')
AND orders.o_totalprice < 901.05
```

**SQL Befehl 16:** Berechnet die Zeilenanzahl nach einem vollständigen Join (Kreuzprodukt) der Tabellen *supplier* und *customer*. Die verwendeten Tabellen machen jeweils 2 % (*customer*) und ungefähr 1 % (*supplier*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM supplier, customer
```

**SQL Befehl 17:** Berechnet die Zeilenanzahl nach einem vollständigen Join (Kreuzprodukt) der Tabellen *supplier* und *part*. Die verwendeten Tabellen machen jeweils ungefähr 2 % (*part*) und ungefähr 1 % (*supplier*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM supplier, part
```

**SQL Befehl 18:** Berechnet die Zeilenanzahl nach einem vollständigen Join (Kreuzprodukt) der Tabellen *supplier* und *partsupp*. Die verwendeten Tabellen machen jeweils ungefähr 1 % (*supplier*) und 11 % (*partsupp*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM supplier, partsupp
```

**SQL Befehl 19:** Berechnet die Zeilenanzahl nach einem Join von *lineitem* und *orders* mit der Bedingung, dass die Spalte *l_orderkey* aus der Tabelle *lineitem* der Spalte *o_orderkey* aus der Tabelle *orders* entspricht. Die Spalte *o_orderkey* ist der Primärschlüssel der Tabelle *orders*. Die verwendeten Tabellen machen jeweils 69 % (*lineitem*) und 16 % (*orders*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM lineitem, orders
WHERE l_orderkey = o_orderkey
```

**SQL Befehl 20:** Berechnet die Zeilenanzahl nach einem Join von *lineitem* und *part* mit der Bedingung, dass die Spalte *l_partkey* aus der Tabelle *lineitem* der Spalte *p_partkey* aus der Tabelle *part* entspricht. Die Spalte *p_partkey* ist der Primärschlüssel der Tabelle *part*. Die verwendeten Tabellen machen jeweils 69 % (*lineitem*) und 2 % (*part*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM lineitem, part WHERE l_partkey = p_partkey
```

**SQL Befehl 21:** Berechnet die Zeilenanzahl nach einem Join von *orders* und *customer* mit der Bedingung, dass die Spalte *o_custkey* aus der Tabelle *orders* der Spalte *c_custkey* aus der Tabelle *customer* entspricht. Die Spalte *c_custkey* ist der Primärschlüssel der Tabelle *customer*. Die verwendeten Tabellen machen jeweils 16 % (*orders*) und 2 % (*customer*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM orders, customer WHERE o_custkey = c_custkey
```

**SQL Befehl 22:** Berechnet die Zeilenanzahl nach einem vollständigen Join (Kreuzprodukt) der Tabellen *orders* und *nation*. Die verwendeten Tabellen machen jeweils 16 % (*orders*) und ungefähr 1 % (*nation*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM orders, nation
```

**SQL Befehl 23:** Berechnet die Zeilenanzahl nach einem vollständigen Join (Kreuzprodukt) der Tabellen *lineitem* und *region*. Die verwendeten Tabellen machen jeweils 69 % (*lineitem*) und ungefähr 1 % (*region*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM lineitem, region
```

**SQL Befehl 24:** Berechnet die Zeilenanzahl nach einem vollständigen Join (Kreuzprodukt) der Tabellen *orders* und *region*. Die verwendeten Tabellen machen jeweils 16 % (*orders*) und ungefähr 1 % (*region*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM orders, region
```

**SQL Befehl 25:** Berechnet die Zeilenanzahl nach einem vollständigen Join (Kreuzprodukt) der Tabellen *customer* und *part*. Die verwendeten Tabellen machen 2 % (*customer*) und 2 % (*part*) der gesamten Datenbankgröße aus.

```
SELECT count(*) FROM customer, part
```

**SQL Befehl 26:** Berechnet die Zeilenanzahl nach einem Join von *lineitem* und *orders*. Zusätzlich wird eine Selektion auf verschiedenen Spalten beider Tabellen berechnet. Die WHERE-Klausel beschränkt beispielsweise die Tabelle *lineitem* auf 8 Zeilen und die Tabelle *orders* auf eine Zeile in der 1 GB-Datenbank. Die verwendeten Tabellen machen jeweils 69 % (*lineitem*) und 16 % (*orders*) der gesamten Datenbankgröße aus.

```
SELECT * FROM lineitem, orders
WHERE l_linenumber = 7 AND l_extendedprice > 16000
AND l_extendedprice < 16005 AND o_orderstatus = 'F'
AND o_orderdate < ToDate('1992-01-05','YYYY-MM-DD')
```

**SQL Befehl 27:** Der Wert der Spalte *l_linenumber* in der Tabelle *lineitem* wird auf den Wert „1337“ gesetzt. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
UPDATE lineitem SET l_linenumber=1337
```

**SQL Befehl 28:** Der Wert der Spalte *l_linenumber* in der Tabelle *lineitem* wird auf den Wert „1337“ und der Werte der Spalte *l_shipinstruct* in der Tabelle *lineitem* wird auf den Wert „New value“ gesetzt. Die verwendete Tabelle *lineitem* macht 69 % der gesamten Datenbankgröße aus.

```
UPDATE lineitem SET l_linenumber=1338, l_shipinstruct='New value'
```

**SQL Befehl 29:** Der Wert der Spalte *o_shippriority* in der Tabelle *orders* wird auf den Wert „1337“ gesetzt. Die Tabelle *orders* macht 16 % der gesamten Datenbankgröße aus.

```
UPDATE orders SET o_shippriority=1337
```

**SQL Befehl 30:** Der Wert der Spalte *o_shippriority* in der Tabelle *orders* wird auf den Wert „1337“ gesetzt und der Wert der Spalte *o_orderstatus* in der Tabelle *orders* wird auf den Wert „N“ gesetzt. Die verwendete Tabelle *orders* macht 16 % der gesamten Datenbankgröße aus.

```
UPDATE orders SET o_shippriority=1338, o_orderstatus='N'
```

## 5.3 Leistungsaufnahme der Datenbank

### 5.3.1 Messergebnisse

In den Tabellen 5.5, 5.6 und 5.7 sind die Ergebnisse unserer Tests mit den 30 SQL-Befehlen aufgeführt. Die Tests wurden auf drei unterschiedlich großen Datenbanken durchgeführt. Die Datenbankgrößen waren 1 GB, 5 GB und 10 GB. Die erste Spalte der Tabellen nummeriert die SQL-Befehle von 1 bis 30 durch. Die Spalte mit der Bezeichnung „Laufzeit [s]“ gibt die Zeit in Sekunden an, welche benötigt wurde, um

den jeweiligen SQL-Befehl vollständig auszuführen. Die Spalte „Joule“[2] gibt die geleistete elektrische Arbeit für das vollständige Ausführung eines SQL-Befehls an. Die letzte Spalte „Joule/s“ gibt die durchschnittliche Leistungsaufnahme pro Sekunde in Watt an ($Leistung = \frac{Elektrische\ Arbeit}{Zeit} = \frac{Joule}{Laufzeit}$). Die Laufzeit der Befehle wurde auf maximal eine Stunde begrenzt. Messungen, die diese Grenze erreichten, sind mit dem Wort „Timeout“ markiert.

Anzumerken ist, dass bei manchen Befehlen die Laufzeit und der Energieverbrauch nach Erweiterung des Arbeitsspeichers sogar um ein paar Zehntelsekunden länger bzw. größer wurden. Wir führen dies auf zum Teil nicht vom Benutzer steuerbare Betriebssystemaktivitäten währen der Messungen zurück, insbesondere da dem Betriebssystem selbst weniger Arbeitsspeicher zur Verfügung stand, wenn dieser Arbeitsspeicher dem DBMS zugewiesen wurde. Auch wenn die meisten Hintergrunddienste des Betriebssystems deaktiviert wurden, ließen sich solche Aktivitäten nicht gänzlich vermeiden.

### 5.3.2 Analyse der Messergebnisse bezüglich Veränderungen der Arbeitsspeichergröße

**Hypothese:** Die Größe des Arbeitsspeichers beeinflusst die benötigte Energie und benötigte Zeit zur Ausführung eines SQL-Befehls. Eine Vergrößerung des Arbeitsspeichers führt zu kürzeren Ausführungszeiten und zu einem geringeren Energieverbrauch.

Diese Hypothese soll in diesem Abschnitt für die 1 GB-, 5 GB- und die 10 GB-Datenbank überprüft und quantifiziert werden. Bei unseren Messungen bestätigte sich die Aussage für alle Datenbankgrößen, allerdings können die Ausführungszeit und der Energieverbrauch bei einer Erweiterung des Arbeitsspeichers nur verringert werden, falls bei der Berechnung eines SQL-Befehls der zusätzlich verfügbare Arbeitsspeicher auch tatsächlich vom DBMS verwendet wird. Es wurden Tests durchgeführt mit einer Arbeitsspeichergröße von 256 MB und 3 GB durch. Die Testergebnisse sind in den Tabellen 5.5, 5.6 und 5.7 angegeben und werden im Folgenden untersucht.

#### 1 GB-Datenbank

Bei unseren Tests auf der **1 GB-Datenbank** spielte die Größe des Arbeitsspeichers für die Ausführungszeit nur für Sortierungen, Befehl 6 und 7, eine Rolle. Für die Ausführungszeiten aller anderen SQL-Befehle machte es kaum einen Unterschied ob der Datenbank 256 MB oder 3 GB Arbeitsspeicher zur Verfügung standen. Bei den Befehlen 6 und 7, bei denen eine Vergrößerung des Arbeitsspeichers zu kürzeren Laufzeiten führte, nahm der Werte für die benötigte Energie zur Ausführung der Befehle nicht in

[2] Als Maßeinheit für die elektrische Arbeit (elektrische Energie) ist im Alltag die Kilowattstunde (kWh) gebräuchlich. Eine Kilowattstunde entspricht 3.600.000 Wattsekunden (Ws). Eine Wattsekunde (oder auch *Joule*) ist gleich der Energie, die benötigt wird, um für die Dauer einer Sekunde die Leistung von einem Watt aufzubringen.

| | 1 GB Datenbank | | | | | |
|---|---|---|---|---|---|---|
| | 256 MB Arbeitsspeicher | | | 3 GB Arbeitsspeicher | | |
| Nr. | Laufzeit [s] | Joule | Joule/s | Laufzeit [s] | Joule | Joule/s |
| 1 | 20,79 | 2592,21 | 124,68 | 20,87 | 2606,40 | 124,89 |
| 2 | 31,42 | 4082,28 | 129,93 | 31,62 | 4100,02 | 129,65 |
| 3 | 34,31 | 4527,60 | 131,95 | 35,68 | 4676,24 | 131,07 |
| 4 | 152,52 | 20988,80 | 137,61 | 152,60 | 20968,92 | 137,41 |
| 5 | 113,19 | 15266,18 | 134,87 | 113,87 | 15312,41 | 134,47 |
| 6 | 335,59 | 42124,55 | 125,52 | 220,41 | 30901,36 | 140,20 |
| 7 | 168,28 | 22754,40 | 135,22 | 159,08 | 21820,58 | 137,17 |
| 8 | 30,43 | 3971,83 | 130,53 | 30,50 | 3963,41 | 129,94 |
| 9 | 42,92 | 5635,99 | 131,33 | 42,90 | 5670,80 | 132,19 |
| 10 | 73,82 | 9814,69 | 132,96 | 73,91 | 9786,05 | 132,40 |
| 11 | 49,16 | 6615,41 | 134,56 | 48,80 | 6519,61 | 133,60 |
| 12 | 65,05 | 8578,91 | 131,87 | 64,51 | 8529,43 | 132,22 |
| 13 | 64,88 | 8588,33 | 132,38 | 64,96 | 8585,44 | 132,17 |
| 14 | 84,58 | 11263,64 | 133,18 | 83,57 | 11122,49 | 133,09 |
| 15 | 568,16 | 79197,82 | 139,39 | 567,31 | 78926,06 | 139,12 |
| 16 | 2563,32 | 360083,12 | 140,48 | 2565,06 | 359465,62 | 140,14 |
| 17 | 3417,12 | 480127,52 | 140,51 | 3425,62 | 480016,85 | 140,13 |
| 18 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 19 | 98,68 | 13410,98 | 135,90 | 98,01 | 13228,80 | 134,98 |
| 20 | 105,10 | 14362,19 | 136,65 | 106,40 | 14436,50 | 135,68 |
| 21 | 28,48 | 3765,61 | 132,24 | 29,53 | 3963,26 | 134,22 |
| 22 | 67,58 | 9319,67 | 137,91 | 67,77 | 9343,73 | 137,88 |
| 23 | 86,05 | 11575,23 | 134,52 | 86,13 | 11581,49 | 134,46 |
| 24 | 21,83 | 2882,14 | 132,00 | 22,72 | 3010,99 | 132,53 |
| 25 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 26 | 8,25 | 1036,67 | 125,60 | 8,29 | 1090,05 | 131,55 |
| 27 | 444,40 | 58557,75 | 131,77 | 445,64 | 58560,67 | 131,41 |
| 28 | 450,28 | 59436,55 | 132,00 | 453,39 | 59680,83 | 131,63 |
| 29 | 102,95 | 13223,72 | 128,45 | 101,45 | 13134,26 | 129,47 |
| 30 | 103,45 | 13334,99 | 128,91 | 103,81 | 13371,08 | 128,80 |

Tabelle 5.5: Messergebnisse der 30 SQL-Befehle auf der 1 GB-Datenbank bei 256 MB und 3 GB Arbeitsspeicher.

| | 5 GB-Datenbank | | | | | |
|---|---|---|---|---|---|---|
| | 256 MB Arbeitsspeicher | | | 3 GB Arbeitsspeicher | | |
| Nr. | Laufzeit [s] | Joule | Joule/s | Laufzeit [s] | Joule | Joule/s |
| 1 | 105,70 | 13195,65 | 124,84 | 103,61 | 12953,58 | 125,02 |
| 2 | 162,51 | 21107,36 | 129,88 | 159,63 | 20831,31 | 130,50 |
| 3 | 172,51 | 22868,88 | 132,57 | 173,94 | 22966,35 | 132,04 |
| 4 | 750,92 | 103588,40 | 137,95 | 760,37 | 104804,84 | 137,83 |
| 5 | 569,84 | 76940,40 | 135,02 | 566,10 | 76291,17 | 134,77 |
| 6 | 1879,95 | 227585,80 | 121,06 | 1689,44 | 209693,02 | 124,12 |
| 7 | 898,93 | 119718,87 | 133,18 | 837,49 | 112976,77 | 134,90 |
| 8 | 29,69 | 3865,28 | 130,19 | 30,14 | 3897,55 | 129,30 |
| 9 | 111,91 | 14792,02 | 132,18 | 108,65 | 14310,11 | 131,71 |
| 10 | 191,52 | 25516,34 | 133,23 | 184,62 | 24562,47 | 133,05 |
| 11 | 242,85 | 32724,86 | 134,75 | 235,80 | 31704,11 | 134,45 |
| 12 | 68,68 | 9104,96 | 132,57 | 67,41 | 8923,80 | 132,37 |
| 13 | 323,51 | 42873,07 | 132,52 | 322,04 | 42620,98 | 132,35 |
| 14 | 417,61 | 55936,54 | 133,94 | 422,00 | 56505,83 | 133,90 |
| 15 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 16 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 17 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 18 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 19 | 501,01 | 68080,30 | 135,89 | 496,52 | 67549,91 | 136,05 |
| 20 | 583,66 | 80032,60 | 137,12 | 585,56 | 80200,46 | 136,96 |
| 21 | 155,63 | 21296,30 | 136,84 | 156,51 | 21331,79 | 136,30 |
| 22 | 861,52 | 101986,51 | 118,38 | 350,52 | 48215,15 | 137,55 |
| 23 | 431,81 | 58449,80 | 135,36 | 435,51 | 59043,51 | 135,57 |
| 24 | 108,52 | 14713,81 | 135,59 | 108,98 | 14755,38 | 135,40 |
| 25 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 26 | 42,84 | 5510,92 | 128,64 | 43,4 | 5644,19 | 129,91 |
| 27 | 2251,36 | 295135,51 | 131,09 | 2254,41 | 296512,98 | 131,53 |
| 28 | 2295,19 | 301259,25 | 131,26 | 2301,66 | 303278,95 | 131,77 |
| 29 | 509,11 | 66171,83 | 129,98 | 518,46 | 66775,39 | 128,80 |
| 30 | 505,54 | 65961,57 | 130,48 | 498,26 | 65357,43 | 131,17 |

Tabelle 5.6: Messergebnisse der 30 SQL-Befehle auf der 5 GB-Datenbank bei 256 MB und 3 GB Arbeitsspeicher.

| | 10 GB-Datenbank | | | | | |
|---|---|---|---|---|---|---|
| | 256 MB Arbeitsspeicher | | | 3 GB Arbeitsspeicher | | |
| Nr. | Laufzeit [s] | Joule | Joule/s | Laufzeit [s] | Joule | Joule/s |
| 1 | 207,84 | 26114,56 | 125,65 | 208,14 | 26154,00 | 125,66 |
| 2 | 317,43 | 41644,63 | 131,19 | 318,01 | 41662,82 | 131,01 |
| 3 | 345,64 | 46016,46 | 133,13 | 345,97 | 46075,14 | 133,18 |
| 4 | 1522,72 | 210416,27 | 138,18 | 1511,02 | 209074,29 | 138,37 |
| 5 | 1138,07 | 153760,21 | 135,11 | 1141,50 | 154431,55 | 135,29 |
| 6 | 3600,02 | 440827,66 | 122,45 | 3527,00 | 434991,32 | 123,33 |
| 7 | 1802,08 | 240343,54 | 133,37 | 1751,01 | 235843,31 | 134,69 |
| 8 | 30,86 | 3986,45 | 129,18 | 30,97 | 4010,49 | 129,49 |
| 9 | 111,73 | 14853,33 | 132,94 | 110,91 | 14725,00 | 132,77 |
| 10 | 189,77 | 25364,02 | 133,66 | 191,51 | 25591,50 | 133,63 |
| 11 | 485,32 | 65521,21 | 135,01 | 478,42 | 64728,76 | 135,30 |
| 12 | 68,04 | 8974,48 | 131,90 | 68,57 | 9086,34 | 132,52 |
| 13 | 649,27 | 86350,11 | 133,00 | 649,81 | 86552,77 | 133,20 |
| 14 | 836,56 | 112432,93 | 134,40 | 832,21 | 112109,08 | 134,71 |
| 15 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 16 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 17 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 18 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 19 | 1005,92 | 136944,40 | 136,14 | 989,75 | 135031,59 | 136,43 |
| 20 | 1314,70 | 179686,49 | 136,68 | 1318,77 | 180738,80 | 137,05 |
| 21 | 336,73 | 45945,35 | 136,45 | 337,85 | 46195,20 | 136,73 |
| 22 | 1321,79 | 166612,41 | 126,05 | 683,62 | 95144,24 | 139,18 |
| 23 | 871,57 | 118389,34 | 135,83 | 862,53 | 117347,26 | 136,05 |
| 24 | 216,96 | 29359,66 | 135,32 | 218,66 | 29730,23 | 135,97 |
| 25 | Timeout | N/A | N/A | Timeout | N/A | N/A |
| 26 | 82,66 | 10796,60 | 130,61 | 83,16 | 10928,96 | 131,42 |
| 27 | 4393,32 | 582141,43 | 132,51 | 4384,92 | 582489,94 | 132,84 |
| 28 | 4397,61 | 589263,01 | 134,00 | 4526,33 | 599606,37 | 132,47 |
| 29 | 973,68 | 128487,63 | 131,96 | 983,20 | 129873,36 | 132,09 |
| 30 | 981,54 | 129611,24 | 132,05 | 986,92 | 130249,98 | 131,98 |

Tabelle 5.7: Messergebnisse der 30 SQL-Befehle auf der 10 GB-Datenbank bei 256 MB und 3 GB Arbeitsspeicher.

dem selben prozentualen Ausmaß ab. Dies lässt sich aus Tabelle 5.8 ablesen, in welcher für alle 30 SQL-Befehle der prozentuale Unterschied bezüglich der Laufzeit und des Energieverbrauchs bei einer Veränderung der Arbeitsspeichergröße von 256 MB auf 3 GB angegeben ist. Die Tabelle bezieht sich auf die 1 GB-Datenbank und die Daten zu den Befehlen 18 und 25 wurden nicht ausgewertet, da bei der Berechnung die von uns gewählte maximale Berechnungsdauer von einer Stunde überschritten wurde.

Besonders interessant ist eine Analyse von Befehl 6 (eine Sortierung). Nach Erweiterung des Arbeitsspeichers nahm die Laufzeit dieses Befehls um 34,32 % ab, während die benötigte Energie nur um 26,64 % abnahm. Bei diesem Befehl lohnt ein Blick auf die Hardware-Nutzung während der Ausführung. Die Abbildungen 5.1 und 5.2 stellen neben der Leistungsaufnahme des Rechners die Nutzung des Prozessors und


Abbildung 5.1: SQL Befehl 6, 256 MB RAM (cold), 1 GB-Datenbank.

die I/O-Aktivität grafisch dar. Bei 256 MB Arbeitsspeicher (siehe 5.1) ist ab Sekunde 225 eine heftige I/O-Aktivität zu erkennen, die bis zum Ende anhält. Der zu kleine Arbeitsspeicher wird durch ständiges Lesen und Schreiben auf der Festplatte ausgeglichen. Der Prozessor ist in dieser Phase nicht ausgelastet und die Leistungsaufnahme des Systems liegt mit ca. 100 Watt nur 10 Watt über der Leistungsaufnahme im Leerlauf. Dadurch dass der Prozessor nicht ausgelastet ist, verlängert sich die Laufzeit. Bei 3 GB Arbeitsspeicher kann diese I/O-Aktivität vermieden werden, da sich alle benötigten Daten im Arbeitsspeicher befinden (siehe 5.2). Der Prozessor ist über die gesamte Ausführungszeit vollständig ausgelastet und die Ausführungszeit ist damit

| | 1 GB-Datenbank | |
|---|---|---|
| Nr. | Laufzeit-Veränderung [%] | Joule-Veränderung [%] |
| 1 | +0,38 | +0,55 |
| 2 | +0,65 | +0,43 |
| 3 | +3,97 | +3,28 |
| 4 | +0,05 | -0,09 |
| 5 | +0,60 | +0,30 |
| 6 | -34,32 | -26,64 |
| 7 | -5,47 | -4,10 |
| 8 | +0,25 | -0,21 |
| 9 | -0,04 | +0,62 |
| 10 | +0,13 | -0,29 |
| 11 | -0,74 | -1,45 |
| 12 | -0,83 | -0,58 |
| 13 | +0,13 | -0,03 |
| 14 | -1,19 | -1,25 |
| 15 | -0,15 | -0,34 |
| 16 | +0,07 | -0,17 |
| 17 | +0,25 | -0,02 |
| 18 | N/A | N/A |
| 19 | -0,68 | -1,36 |
| 20 | +1,24 | +0,52 |
| 21 | +3,70 | +5,25 |
| 22 | +0,29 | +0,26 |
| 23 | +0,10 | +0,05 |
| 24 | +4,06 | +4,47 |
| 25 | N/A | N/A |
| 26 | +0,39 | +5,15 |
| 27 | +0,28 | +0,00 |
| 28 | +0,69 | +0,41 |
| 29 | -1,46 | -0,68 |
| 30 | +0,35 | +0,27 |

Tabelle 5.8: Prozentuale Veränderung der Laufzeit und des Energieverbrauchs der 30 SQL-Befehle nach Erweiterung des Arbeitsspeichers von 256 MB auf 3 GB auf der 1 GB-Datenbank.


Abbildung 5.2: SQL Befehl 6, 3 GB RAM (cold), 1 GB-Datenbank.

insgesamt kürzer.

Der Grund, warum bei einer Erweiterung des Arbeitsspeichers die Laufzeit um 34,32 % und der Energieverbrauch nur um 26,64 % abnimmt, ist, dass bei 256 MB Arbeitsspeicher zwar die Laufzeit durch die heftige I/O-Phase verlängert wird, aber in dieser Phase der Energieverbrauch nur ca. 100 Watt beträgt. Bei einer vollständigen Auslastung des Prozessors werden hingegen ca. 140 Watt verbraucht. Dies führt letztlich dazu, dass die Abnahme des Energieverbrauchs geringer ausfällt als die Abnahme der Laufzeit, da die I/O-Phase zwar die Laufzeit verlängert, aber hinsichtlich des Energieverbrauchs nicht in demselben Maße zu Buche schlägt.

Im Sinne einer energieeffizienten Datenbank ist es empfehlenswert, die Arbeitsspeichergröße bei speicherintensiven SQL-Befehlen ausreichend groß zu wählen, um eine I/O-Aktivität zu vermeiden. Bei unseren Tests mit den 30 SQL-Befehlen auf der 1 GB-Datenbank sank nach einer Erweiterung des Arbeitsspeichers von 256 MB auf 3 GB die Gesamtlaufzeit aller 30 SQL-Befehle um 1,15 %, während gleichzeitig der Gesamt-Energieverbrauch um 0,99 % abnahm. Im Einzelnen spielt es eine große Rolle, welche SQL-Befehle ausgeführt werden. Bei einer speicherintensiven Sortierung sank die Laufzeit beispielsweise um 34,32 %, während der Energieverbrauch um 26,64 % abnahm.

### 5 GB-Datenbank

Die Ergebnisse unserer Messungen auf der **5 GB-Datenbank** bestätigen die Erkenntnisse. Zusätzlich kann der energiesparende Effekt eines größeren Arbeitsspeichers bei diesen Messungen anhand der Messergebnisse von Befehl 22 beobachtet werden. Für alle 30 SQL-Befehle ist in Tabelle 5.9 wieder der prozentuale Unterschied bezüglich der Laufzeit und des Energieverbrauchs bei einer Veränderung der Arbeitsspeichergröße von 256 MB auf 3 GB angegeben. Die Tabelle bezieht sich auf die 5 GB-Datenbank. Befehl 6, eine Sortierung, profitiert, wie bereits bei den Messungen auf der 1 GB-Datenbank, von dem größeren Arbeitsspeicher. Die Laufzeit nahm um 10,13 % ab, während der Energieverbrauch um 7,86 % sank. Da die Datenbank mit 5 GB nun erheblich größer ist, hatte die Erweiterung des Arbeitsspeichers allerdings keinen so großen Einfluss mehr.

Weiterhin auffällig ist, dass die Laufzeit und der Energieverbrauch von Befehl 22, eine Verknüpfung der sehr kleinen Tabelle *nation* und der größeren Tabelle *orders*, durch eine Erweiterung des Arbeitsspeichers ganz erheblich verkürzt bzw. verringert werden können. Die Laufzeit sank um 59,31 % und der Energieverbrauch sank um 52,72 %. Die Tabelle *nation* hat 25 Zeilen, skaliert nicht mit der Datenbankgröße und belegt 3 KB auf dem Datenträger. Die Größe der Tabelle *orders* skaliert mit der Datenbankgröße und macht 16 % der gesamten Datenbankgröße aus. Auf dem Datenträger belegt diese Tabelle 857.117 KB. Die verkürzte Ausführungszeit wurde durch die Verwendung des zusätzlichen Arbeitsspeichers und die dadurch geringere Festplattenaktivität erreicht.

Der Grund, warum bei Befehl 22 die Laufzeit um 59,31 % und der Energieverbrauch nur um 52,72 % abnimmt, ist ein ähnlicher wie auch schon bei Befehl 6 auf der 1 GB-Datenbank. Jedoch tritt dieser Grund in diesem Fall nicht so deutlich und isoliert in Erscheinung. Gegenüber dem Durchlauf mit 3 GB Arbeitsspeicher ist bei 256 MB Arbeitsspeicher die Laufzeit von Befehl 22 durch eine durchgängige I/O-Aktivität verlängert (siehe Abbildung 5.3). Der Prozessor ist, je nach Umfang der I/O-Aktivität, nur zu 80 % und 30 % ausgelastet. Durch die durchgängige Festplattenaktivität ist der Prozessor nicht komplett ausgelastet und die Laufzeit erhöht sich dadurch. Bei 3 GB Arbeitsspeicher (siehe Abbildung 5.4) kann das DBMS sämtliche benötigten Daten am Anfang in den Arbeitsspeicher laden. In der Folgezeit ist keine Festplattenaktivität mehr nötig und der Prozessor ist komplett ausgelastet. Die Laufzeit ist demnach deutlich kürzer, allerdings ist die Leistungsaufnahme aufgrund des komplett ausgelasteten Prozessors über die gesamte Laufzeit hinweg 140 Watt. Aus diesem Grund fällt die Reduktion des Energieverbrauchs etwas geringer als die Reduktion der Laufzeit aus.

Die Bedeutung der Arbeitsspeichergröße bezüglich der Energieeffizienz des DBMS steigt mit wachsender Datenmenge an. Bei unseren Tests mit den 30 SQL-Befehlen auf der 5 GB-Datenbank sank nach einer Erweiterung des Arbeitsspeichers von 256 MB auf 3 GB die Gesamtlaufzeit aller 30 SQL-Befehle um 5,36 %, während der Gesamt-

| | 5 GB-Datenbank | |
|---|---|---|
| Nr. | Laufzeit-Veränderung [%] | Joule-Veränderung [%] |
| 1 | -1,97 | -1,83 |
| 2 | -1,77 | -1,31 |
| 3 | +0,83 | +0,43 |
| 4 | +1,26 | +1,17 |
| 5 | -0,66 | -0,84 |
| 6 | -10,13 | -7,86 |
| 7 | -6,84 | -5,63 |
| 8 | +1,53 | +0,83 |
| 9 | -2,91 | -3,26 |
| 10 | -3,60 | -3,74 |
| 11 | -2,90 | -3,12 |
| 12 | -1,84 | -1,99 |
| 13 | -0,46 | -0,59 |
| 14 | +1,05 | +1,02 |
| 15 | N/A | N/A |
| 16 | N/A | N/A |
| 17 | N/A | N/A |
| 18 | N/A | N/A |
| 19 | -0,90 | -0,78 |
| 20 | +0,32 | +0,21 |
| 21 | +0,56 | +0,17 |
| 22 | -59,31 | -52,72 |
| 23 | +0,86 | +1,02 |
| 24 | +0,42 | +0,28 |
| 25 | N/A | N/A |
| 26 | +1,41 | +2,42 |
| 27 | +0,14 | +0,47 |
| 28 | +0,28 | +0,67 |
| 29 | +1,84 | +0,91 |
| 30 | -1,44 | -0,92 |

Tabelle 5.9: Prozentuale Veränderung der Laufzeit und des Energieverbrauchs der 30 SQL-Befehle nach Erweiterung des Arbeitsspeichers von 256 MB auf 3 GB auf der 5 GB-Datenbank.

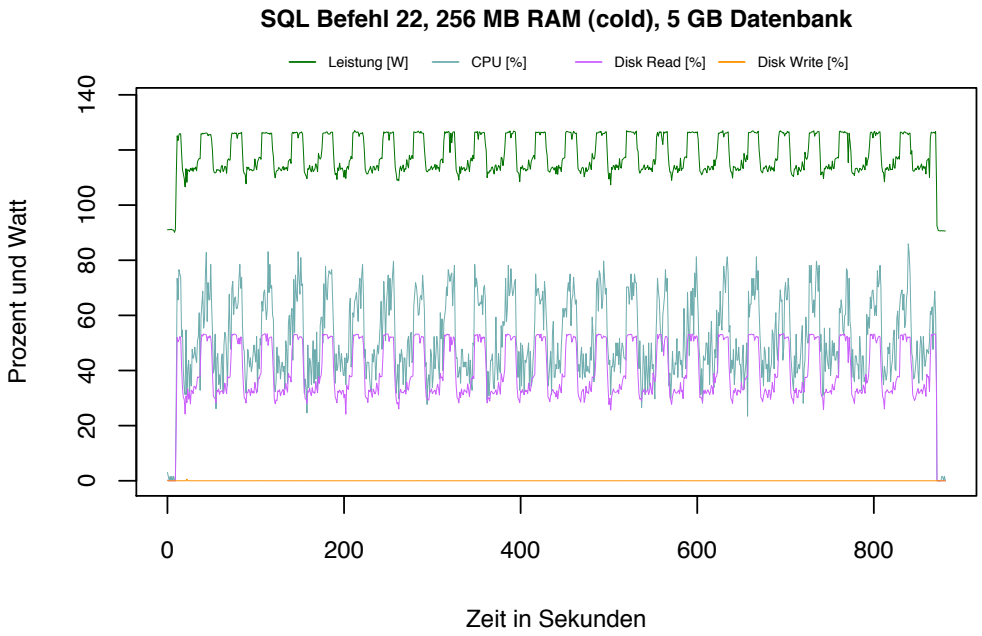



Abbildung 5.3: SQL Befehl 22, 256 MB RAM (cold), 5 GB-Datenbank.

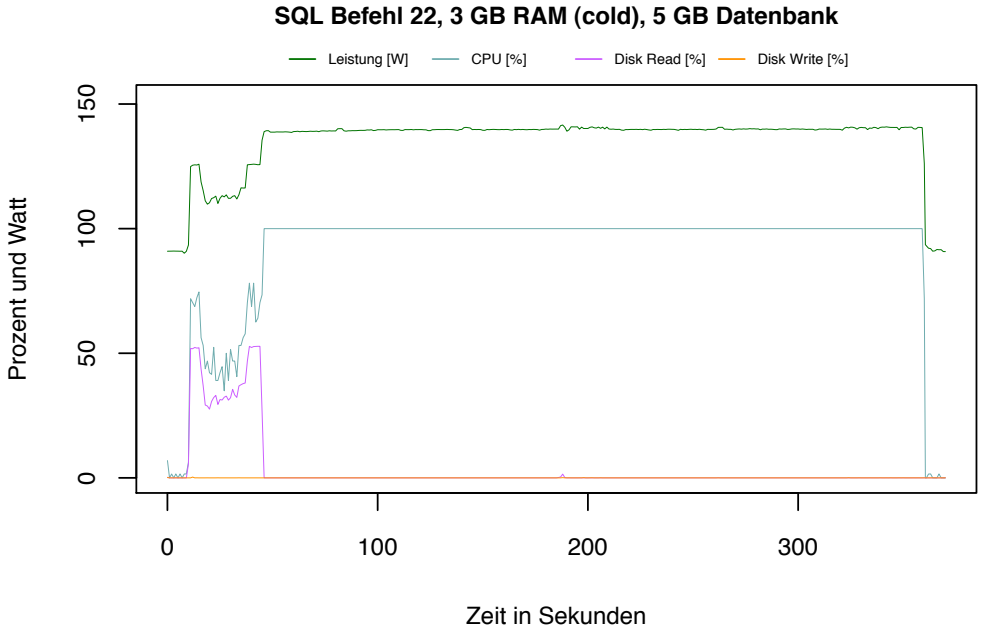



Abbildung 5.4: SQL Befehl 22, 3 GB RAM (cold), 5 GB-Datenbank.

Energieverbrauch um 4,15 % abnahm. Gegenüber den Messungen auf der 1 GB-Datenbank ist bei den Messungen auf der 5 GB-Datenbank die Reduktion der Laufzeit und des Energieverbrauchs größer.

### 10 GB-Datenbank

Die Messungen auf der **10 GB-Datenbank** bringen keine neuen Erkenntnisse, bestätigen aber die Erkenntnisse der vorherigen Messungen. Der prozentuale Unterschied bezüglich der Laufzeit und des Energieverbrauchs bei einer Veränderung der Arbeitsspeichergröße von 256 MB auf 3 GB ist für die Messungen auf der 10 GB-Datenbank in Tabelle 5.10 angegeben.

Befehl 22 profitiert auch bei dieser Datenbankgröße wieder von der Erweiterung des Arbeitsspeichers. Die Laufzeit verringert sich um 48,28 % und der Energieverbrauch nimmt um 42,89 % ab. Der Unterschied im Umfang der Reduktionen wurde bereits zuvor in diesem Abschnitt erklärt und auch hier ist die Begründung dieselbe. Für die meisten anderen Abfragen liegt die Reduktion der Laufzeit und des Energieverbrauchs im Bereich von 0 % bis 3 %. Auch für die 10 GB-Datenbank ist eine Erweiterung des Arbeitsspeichers zu empfehlen. Die Gesamtlaufzeit aller 30 SQL-Befehle verkürzt sich durch einen größeren Arbeitsspeicher um 2,49 % und der Gesamt-Energieverbrauch nimmt um 2,08 % ab.

### Zusammenfassung

Die Hypothese (siehe Abschnitt 5.3.2), welche am Anfang dieses Abschnitts aufgestellt wurde, konnte durch Messungen bestätigt und quantifiziert werden. Die Analyse der Messergebnisse bezüglich einer Veränderung der Arbeitsspeichergröße von 256 MB auf 3 GB hat ergeben, dass über alle Datenbankgrößen hinweg bei einer Erweiterung des Arbeitsspeichers eine Reduktion des Gesamt-Energieverbrauchs und der Gesamtlaufzeit festzustellen ist. Während bei der 1 GB-Datenbank die Reduktion der Gesamtlaufzeit mit 1,15 % gering ausfällt, so beläuft sie sich für die 5 GB- und 10 GB-Datenbank auf 5,36 % und 4,15 %. Die Reduktion des Gesamt-Energieverbrauchs fällt insgesamt geringer aus. Hier beträgt die Reduktion für die 1 GB-Datenbank 0,99 % und für die 5 GB- und 10 GB-Datenbank 4,15 % und 2,08 %.

Diese Werte sind jedoch nicht direkt über die Datenbankgrößen hinweg miteinander vergleichbar, da auf größeren Datenbanken zusätzliche Zeitüberschreitungen auftraten. Während auf der 1 GB-Datenbank und 5 GB-Datenbank nur 2 bzw. 5 Zeitüberschreitungen auftraten, so gab es bei den Messungen auf der 10 GB-Datenbank eine zusätzliche Zeitüberschreitung. Für jede Datenbankgröße wurden nur die Datensätze in die Berechnung mit einbezogen, bei denen keine Zeitüberschreitung auftrat. Um einen direkten Vergleich über die Datenbankgrößen hinweg zu ermöglichen, wurden für die Berechnungen der Werte in Tabelle 5.11 für die 1 GB- und 5 GB-Datenbank zusätzliche Datensätze ausgeschlossen. Ausgeschlossen wurden diejenigen Datensätze, bei denen es auf der größten Datenbank, der 10 GB-Datenbank, eine Zeitüber-

| | 10 GB-Datenbank | |
|---|---|---|
| Nr. | Laufzeit-Veränderung [%] | Joule-Veränderung [%] |
| 1 | +0,14 | +0,15 |
| 2 | +0,18 | +0,04 |
| 3 | +0,10 | +0,13 |
| 4 | -0,77 | -0,64 |
| 5 | +0,30 | +0,44 |
| 6 | N/A | N/A |
| 7 | -2,83 | -1,87 |
| 8 | +0,36 | +0,60 |
| 9 | -0,74 | -0,86 |
| 10 | +0,92 | +0,90 |
| 11 | -1,42 | -1,21 |
| 12 | +0,77 | +1,25 |
| 13 | +0,08 | +0,23 |
| 14 | -0,52 | -0,29 |
| 15 | N/A | N/A |
| 16 | N/A | N/A |
| 17 | N/A | N/A |
| 18 | N/A | N/A |
| 19 | -1,61 | -1,40 |
| 20 | +0,31 | +0,59 |
| 21 | +0,33 | +0,54 |
| 22 | -48,28 | -42,89 |
| 23 | -1,04 | -0,88 |
| 24 | +0,78 | +1,26 |
| 25 | N/A | N/A |
| 26 | +0,60 | +1,23 |
| 27 | -0,19 | +0,06 |
| 28 | +2,93 | +1,76 |
| 29 | +0,98 | +1,08 |
| 30 | +0,55 | +0,49 |

Tabelle 5.10: Prozentuale Veränderung der Laufzeit und des Energieverbrauchs der 30 SQL-Befehle nach Erweiterung des Arbeitsspeichers von 256 MB auf 3 GB auf der 10 GB-Datenbank.

| Datenbankgröße | Laufzeit-Veränderung [%] | Joule-Veränderung [%] |
|---|---|---|
| 1 GB | -0,10 | -0,16 |
| 5 GB | -4,64 | -3,63 |
| 10 GB | -2,49 | -2,08 |

Tabelle 5.11: Prozentuale Veränderung der Gesamtlaufzeit und des Gesamt-Energieverbrauchs nach Erweiterung des Arbeitsspeichers von 256 MB auf 3 GB.

schreitung gab. Dadurch ist sichergestellt, dass für alle Datenbankgrößen dieselben Datensätze ausgewertet werden. Die Werte in Tabelle 5.11 bestätigen, dass nach einer Erweiterung des Arbeitsspeichers die Reduktion des Gesamt-Energieverbrauchs etwas geringer ausfällt als die Reduktion der Gesamtlaufzeit. Dennoch ist im Sinne einer energieeffizienten Datenbank in unserem Fall eine Erweiterung des Arbeitsspeichers sinnvoll.

Für einzelne Befehle kann der prozentuale Wert für die Reduktion der Laufzeit und des Energieverbrauchs in Einzelfällen stark von den Gesamtwerten abweichen. So kann beispielsweise auf der 5 GB-Datenbank bei Befehl 6 (Sortierung) 10,13 % der Laufzeit und 7,86 % des Energieverbrauchs und bei Befehl 22 (Verknüpfung zweier Tabellen) 59,31 % der Laufzeit und 52,72 % des Energieverbrauchs gespart werden. Solche vereinzelten Befehle sind es auch, welche die Werte für die Gesamtlaufzeit und den Gesamt-Energieverbrauch maßgeblich beeinflussen. Überraschend für uns war, dass eine Erweiterung des Arbeitsspeichers für die Laufzeit und den Energieverbrauch von vielen Befehlen kaum eine Rolle spielte. Neben der generellen Empfehlung, einen möglichst großen Arbeitsspeicher, bei unseren 30 SQL-Befehlen bei verschiedenen Datenbankgrößen brachte dies eine Energieeinsparung von insgesamt bis zu 5 %; in einem realistischen Anwendungsszenario ist eine genaue Betrachtung der zu erwartenden SQL-Befehle empfehlenswert. So ist eine Erweiterung des Arbeitsspeichers bei Verknüpfungen und Sortier-Befehlen besonders nützlich. Bei unseren 30 SQL-Befehlen beläuft sich die Energieeinsparung durch einen größeren Arbeitsspeicher bei solchen bestimmten Befehlen auf Werte bis zu 52,72 %.

### 5.3.3 Betrachtung der Befehlsgruppen bezüglich Veränderungen der Arbeitsspeichergröße

Im Folgenden werden nicht mehr die einzelnen SQL-Befehle und deren Energieverbrauch in Abhängigkeit von der Arbeitsspeichergröße betrachtet, sondern es werden Aussagen über die Gruppen getroffen, denen die verschiedenen Befehle angehören. Wie bereits erwähnt, wurden die 30 SQL-Befehle so gewählt, dass sie in Gruppen zusammenfassen werden können. Insgesamt gibt es sieben Gruppen, deren Laufzeit und Energieverbrauch in Abhängigkeit vom Arbeitsspeicher betrachtet werden. Es sollen Erkenntnisse darüber gewonnen werden, bei welcher Art von Befehlen der Energie-

verbrauch durch eine Vergrößerung des Arbeitsspeichers gesenkt werden kann oder welche Art von SQL-Befehlen kaum von der Vergrößerung profitieren. In Tabelle 5.12 sind die durchschnittlichen Veränderungen der Laufzeit und des Energieverbrauchs für die einzelnen Gruppen bei unterschiedlichen Datenbankgrößen aufgelistet. Auch hier wurde, wie bereits im vorherigen Abschnitt, die Vereinbarung getroffen, dass über alle Datenbankgrößen hinweg nur die SQL-Befehle in den Durchschnittswerten der einzelnen Gruppen berücksichtigt werden, die bei allen Datenbankgrößen innerhalb von 60 Minuten erfolgreich bearbeitet werden konnten. Ein Befehl, der auf der 10 GB großen Datenbank ein Timeout hat, wird daher auch nicht in den Gruppen der kleineren Datenbanken berücksichtigt, selbst wenn es dort zu keinem Timeout kam. Eine weitere Vereinbarung in diesem Abschnitt ist, dass nur prozentuale Änderungen von mehr als 2 % des Energieverbrauchs und der Laufzeit als signifikant erachtet werden sollen.

| Gruppe | Datenbankgröße | Laufzeit-Veränderung [%] | Joule-Veränderung [%] |
|---|---|---|---|
| Gruppe 1 | 1 GB | +1,90 | +1,61 |
| | 5 GB | -0,80 | -0,74 |
| | 10 GB | +0,14 | +0,10 |
| Gruppe 2 | 1 GB | +0,29 | +0,07 |
| | 5 GB | +0,43 | +0,31 |
| | 10 GB | -0,31 | -0,18 |
| Gruppe 3 | 1 GB | -5,47 | -4,10 |
| | 5 GB | -6,84 | -5,63 |
| | 10 GB | -2,83 | -1,87 |
| Gruppe 4 | 1 GB | -0,21 | -0,35 |
| | 5 GB | -2,01 | -2,22 |
| | 10 GB | -0,31 | -0,17 |
| Gruppe 5 | 1 GB | -1,19 | -1,25 |
| | 5 GB | +1,05 | +1,02 |
| | 10 GB | -0,52 | -0,29 |
| Gruppe 6 | 1 GB | +0,69 | +0,54 |
| | 5 GB | -18,92 | -15,23 |
| | 10 GB | -12,74 | -10,56 |
| Gruppe 7 | 1 GB | +0,29 | +0,13 |
| | 5 GB | +0,21 | +0,47 |
| | 10 GB | +1,26 | +0,89 |

Tabelle 5.12: Änderung der Laufzeit und des Energieverbrauchs der verschiedenen Gruppen bei unterschiedlichen Datenbankgrößen.

**Gruppe 1 (Aggregatfunktionen):**
Wie die Werte in Tabelle 5.12 zeigen, kann die Laufzeit bzw. der Energieverbrauch der

Aggregatfunktionen von Gruppe 1 nicht durch eine Erweiterung des Arbeitsspeichers gesenkt werden. Alle Veränderungen liegen unter 2 % und sind somit nicht eindeutig genug. Die größte Änderung bei Gruppe 1 konnte bei der 1 GB-Datenbank erreicht werden. Die Laufzeit der Aggregatfunktionen auf dieser Datenbankgröße liegt sowohl bei 256 MB Arbeitsspeicher als auch bei 3 GB Arbeitsspeicher durchschnittlich unter 30 Sekunden. Eine Änderung der Laufzeit bzw. des Energieverbrauchs von einer Sekunde bedeutet bei einer Gesamtlaufzeit von 30 Sekunden aber schon eine Veränderung von 3,33 %, so dass Änderungen von unter 2 %, wie sie hier zu beobachten sind, einer Laufzeitveränderung von weniger als einer Sekunde entsprechen. Derart geringe Laufzeitänderungen, sowohl positive als auch negative, können z.B. leicht durch das Betriebssystem verursacht werden. Die Tatsache, dass die Erweiterung des Arbeitsspeichers keinen Einfluss auf die Aggregatfunktionen hat, war durchaus zu erwarten, da diese Tests alle mit kalt gestarteter Datenbank durchgeführt wurden. Die betroffenen Datensätze mussten demnach alle von der Festplatte gelesen und die entsprechenden Aggregatfunktionen durchgeführt werden. Da diese Datensätze aber nicht für weitere Operationen im Arbeitsspeicher verbleiben müssen, profitiert die Datenbank auch nicht von einem größeren Arbeitsspeicher. Nachdem ein Tupel in der Aggregatfunktion berücksichtigt wurde, wird es nicht mehr benötigt und kann theoretisch sofort aus dem Arbeitsspeicher entfernt werden.

**Gruppe 2 (Gruppierung):**
Für die Gruppierungsbefehle der Gruppe 2 konnte ebenfalls kein positiver Effekt auf die Laufzeit bzw. den Energieverbrauch gemessen werden. Auch hier sind die gemessenen Abweichungen zu gering, als dass eine derartige Aussage getroffen werden kann. Die Messergebnisse lassen sich damit erklären, dass nur eine Teilmenge der Daten, nämlich die Repräsentanten der einzelnen Gruppen, im Arbeitsspeicher gehalten werden muss. Da die Befehle der Gruppe 2 auf der Tabelle „lineitem“ operieren und nur nach dem Attribut „l_shipmode“ gruppieren, handelt es sich hierbei um die 8 Repräsentanten mit dem String AIR, FOB, MAIL, RAIL, REG AIR, SHIP und TRUCK im Attribut „l_shipmode“. Diese 8 Repräsentanten können problemlos sowohl in dem 256 MB großen Arbeitsspeicher als auch in dem 3 GB großen Arbeitsspeicher erfasst werden. Es ist jedoch zu erwarten, das bei sehr vielen Gruppenrepräsentanten ein größerer Arbeitsspeicher ebenfalls von Vorteil ist, da die Liste der Repräsentanten dann immer im Arbeitsspeicher gehalten werden kann, ohne ausgelagert werden zu müssen.

**Gruppe 3 (Sortierung):**
Bei den Sortierbefehlen der Gruppe 3 konnten die von uns durchgeführten Messungen einen positiven Einfluss der Arbeitsspeichererweiterung belegen. Abbildung 5.5 zeigt deutlich, dass die Vergrößerung des Arbeitsspeichers einen positiven Einfluss auf die Laufzeit und den Energieverbrauch bei Sortierungen sowohl bei der 1 GB-Datenbank als auch bei der 5 GB-Datenbank hat. Diese Messergebnisse sind ebenfalls logisch, da bei einer Sortierung wenn möglich, alle zu sortierenden Tupel im Arbeitsspeicher gehalten werden. Hierbei treten umso häufiger Swapping-Effekte auf, je kleiner der Arbeitsspeicher ist. Dadurch verlängert sich bei zu kleinem Arbeitsspeicher die Laufzeit der Abfrage und somit auch der Energieverbrauch. Der positive Einfluss auf den

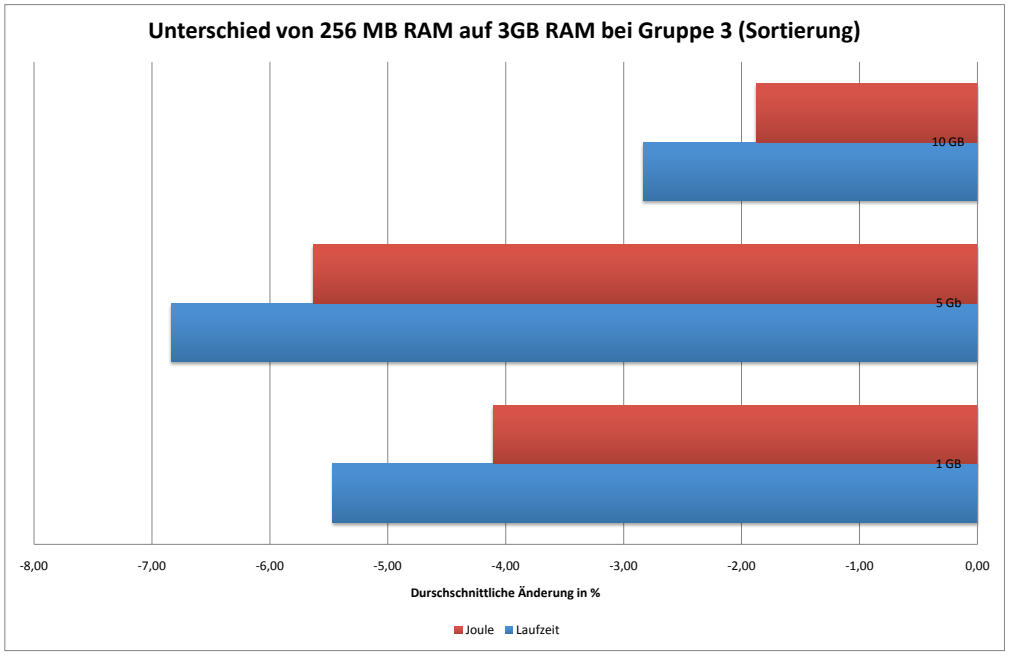



Abbildung 5.5: Unterschied der Laufzeit und des Energieverbrauchs bei Gruppe 3 in Prozent, bei der Erweiterung des Arbeitsspeichers von 256 MB RAM auf 3 GB RAM.

Energieverbrauch und die Laufzeit wird allerdings umso schwächer, je größer die Datenbank ist. Da für eine 10 GB große Datenbank der 3 GB große Arbeitsspeicher ebenso zu klein ist wie der 256 MB große Arbeitsspeicher, profitiert die 10 GB große Datenbank am wenigsten von der Speichererweiterung, wohingegen die 1 GB große Datenbank nach der Vergrößerung des Speichers vollständig in den Arbeitsspeicher geladen werden kann. Während die 1 GB große Datenbank 4,10 % weniger Energie verbraucht, überschreitet die Ersparnis bei der 10 GB großen Datenbank nicht die 2 %-Marke.

**Gruppe 4 (Selektion):**
Bei den Selektionen der Gruppe 4 konnten die Messergebnisse wiederum keinen positiven Einfluss einer Speichererweiterung auf die Laufzeit bzw. auf den Energieverbrauch belegen. Da bei unseren Messungen mit den 30 SQL-Befehlen kein Index oder Primärschlüssel auf den Datenbanken angelegt war, entspricht ein Select-Befehl einem Full-Table-Scan auf der Datenbank. Hierbei verbleiben letztendlich dann nur die Tupel im Arbeitsspeicher, die die WHERE-Klausel erfüllen. Die WHERE-Klauseln wurden von uns so gewählt, dass die Anzahl der zurückgegebenen Tupel nicht zu groß ist. Für die gefundenen Tupel war daher bereits der kleine Arbeitsspeicher wieder ausreichend. Eine Erweiterung des Arbeitsspeichers bringt in diesem Fall keinen Vorteil, wobei auch hier wieder davon auszugehen ist, dass bei Rückgabe sehr vieler Tupel durch einen größeren Arbeitsspeicher unnötiges Swappen verhindert werden kann. Hierdurch würde der Energieverbrauch dann ebenfalls gesenkt werden können.

**Gruppe 5 (Duplikate):**
Die Duplikat-Abfrage der Gruppe 5 profitiert bei unserem Test ebenfalls nicht von einer Vergrößerung des Arbeitsspeichers. Die Messwerte belegen, dass die Laufzeit und der Energieverbrauch durch einen größeren Arbeitsspeicher nicht beeinflusst werden. Bei SQL-Befehlen, die Duplikate ausschließen, wird jedes neu gefundene Tupel in einer Liste der bisher gefundenen Tupel gesucht. Diese Liste wird im Arbeitsspeicher erstellt und gespeichert. Ist ein Tupel noch nicht in dieser Liste enthalten, wird es ausgegeben und in die Liste aufgenommen. Eine Vergrößerung des Arbeitsspeichers wird vermutlich auch in diesem Fall erst Energie sparen, wenn die Liste der bisher aufgetretenen Tupel so groß wird, dass sie nicht mehr in den kleinen Arbeitsspeicher passt. Da der einzige in Gruppe 5 enthaltene Befehl so gewählt war, dass auch dieser nur die Tupel mit den Werten AIR, FOB, MAIL, RAIL, REG AIR, SHIP und TRUCK zurückliefert, kann dieses Messergebnis genau wie bei den Gruppierungen auf die kleine Ergebnismenge zurückgeführt werden. Für diese ist bereits in dem kleinen Speicher ausreichend Platz vorhanden.

**Gruppe 6 (Verbund):**
Die Gruppe 6 der Verbundoperationen profitiert, wie Abbildung 5.6 zeigt, deutlich durch kürzere Laufzeiten und einen geringeren Energieverbrauch von einer Vergrößerung des Arbeitsspeichers. Allerdings zeigt sich hier, dass die 5 GB-Datenbank am meisten von der Vergrößerung des Speichers profitiert, wohingegen die 10 GB große Datenbank weniger von einer Speichererweiterung profitiert. Dies lässt sich dadurch

erklären, dass der Speicher auch nach der Erweiterung für die 10 GB große Datenbank zu klein ist, wohingegen die Erweiterung für die 5 GB große Datenbank schon einen größeren Einfluss hat.

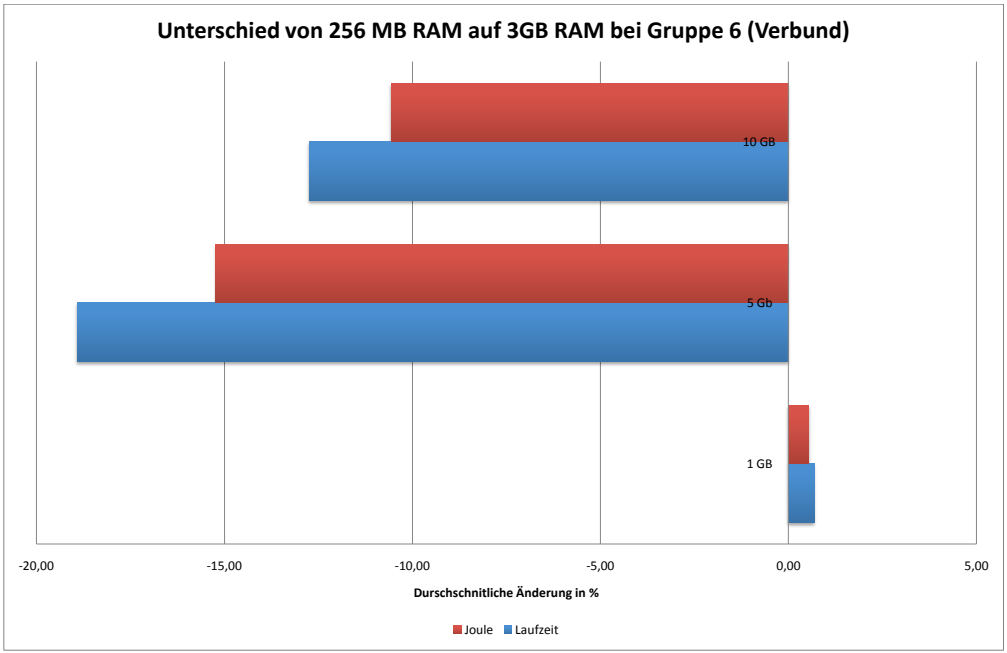



Abbildung 5.6: Unterschied der Laufzeit und des Energieverbrauchs bei Gruppe 6 in Prozent, bei der Erweiterung des Arbeitsspeichers von 256 MB RAM auf 3 GB RAM.

**Gruppe 7 (Updates):**
Bei den Update-Befehlen der Gruppe 7 konnte kein positiver Einfluss auf die Laufzeit bzw. den Energieverbrauch nachgewiesen werden, wenn der Arbeitsspeicher vergrößert wird. Alle Abweichungen sind zu gering, als dass hier von einer Einsparung gesprochen werden kann. Auch dieses Messergebnis scheint plausibel, da die Datenbank nicht alle Daten gleichzeitig im Arbeitsspeicher behalten muss, um diese zu ändern. Hier reicht bereits ein kleiner Speicher aus, um die einfachen Update-Befehle durchführen zu können.

**Zusammenfassung:** Die von uns ermittelten Messwerte belegen, dass besonders die Gruppe 3 der Sortierungen und die Gruppe 6 der Verbunde durch kürzere Laufzeiten und geringeren Energieverbrauch von einer Erweiterung des Arbeitsspeichers profitieren. Alle Befehle der anderen Gruppen profitieren nicht bzw. kaum von einer Speichererweiterung, solange die erzeugten Zwischenergebnisse (Duplikate-Liste, Gruppenrepräsentanten) die Größe des kleineren Arbeitsspeichers nicht überschreiten. In

der Abbildung 5.7 sind die Durchschnittswerte aller Gruppen zusammengefasst. Die

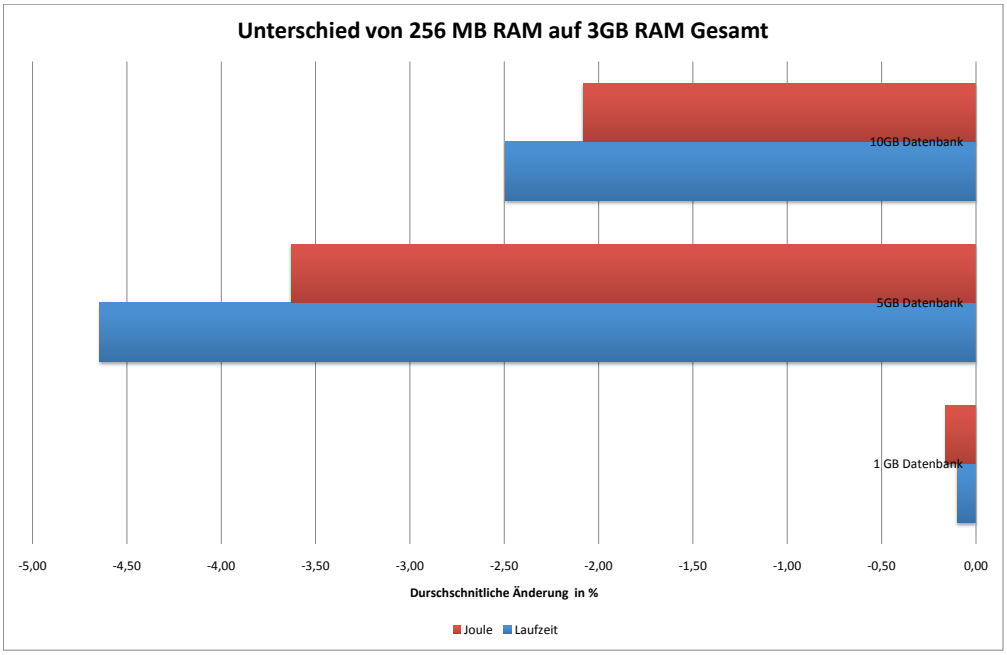



Abbildung 5.7: Unterschied der durchschnittlichen Laufzeit und des durchschnittlichen Energieverbrauchs aller Gruppen in Prozent, bei der Erweiterung des Arbeitsspeichers von 256 MB RAM auf 3 GB RAM.

Abbildung zeigt, dass eine Vergrößerung des Arbeitsspeichers gemittelt über die verschiedenen Gruppen bei unseren Tests insgesamt eine kürzere Laufzeit bzw. einen geringeren Energieverbrauch begünstigt. Eine Vergrößerung des Arbeitsspeichers bei kleinen Datenbanken hat einen weniger großen Einfluss auf den Energieverbrauch, da die meisten Befehle aufgrund der kleinen Datenbankgröße noch effizient auf dem kleinen Arbeitsspeicher abgearbeitet werden können. Am größten ist die Einsparung, wenn der Zugewinn an Speicher möglichst groß ist und in dem kleinen Arbeitsspeicher vor der Vergrößerung nicht ausreichend Platz für die Daten war. Ist der Speicher allerdings auch nach der Erweiterung viel zu klein, so wie die 3 GB-Arbeitsspeicher für die 10 GB-Datenbank, ist das Einsparpotenzial prozentual betrachtet eher gering. Der Grund für eine Verkürzung der Laufzeit und des Energieverbrauchs ist der, dass die Datenbank aufgrund der Speichererweiterung weniger I/O verursacht und Zwischenergebnisse, z.B. für Sortierungen, besser im Arbeitsspeicher gehalten werden können. Am meisten Energie wird in den Fällen gespart, in denen die Datenbank bzw. deren Zwischenergebnisse für die Abfragen komplett in den Speicher geladen werden kann, da hier keine unnötigen Lese- und Schreibvorgänge auf der Festplatte durchgeführt werden müssen. Einige Befehls-Gruppen, wie zum Beispiel die Aggregat-

funktionen, durchlaufen die Tabellen der Datenbank sequentiell (full table scan). Hier hat ein größerer Arbeitsspeicher keinen Einfluss auf die Laufzeit, da die Daten nach deren Berücksichtigung in der Aggregatfunktion selbst nicht mehr im Arbeitsspeicher benötigt werden.

### 5.3.4 Analyse der Messergebnisse bezüglich Veränderungen der Datenbankgröße

**Hypothese:** Die Größe der Datenbank, auf welcher operiert wird, beeinflusst die benötigte Energie und benötigte Zeit zur Ausführung eines SQL-Befehls. Größere Relationen führen zu längeren Ausführungszeiten und zu einem höheren Energieverbrauch, falls ein SQL-Befehl auf diesen Relationen operiert.

Diese Hypothese soll in diesem Abschnitt für den Übergang von der 1 GB- zur 5 GB-Datenbank (Änderungsfaktor 5) und für den Übergang von der 5 GB- zur 10 GB-Datenbank (Änderungsfaktor 2) überprüft werden. Diese Überprüfung wird bei einer festen Arbeitsspeichergröße von 3 GB durchgeführt, da die Auswirkungen bezüglich einer Veränderung der Arbeitsspeichergröße bereits im vorherigen Abschnitt untersucht wurden. Unsere Messergebnisse zeigen eine Veränderung der Laufzeit und des Energieverbrauchs, welche dem jeweiligen Änderungsfaktor bei einem Datenbankwechsel direkt entspricht. Es gibt allerdings auch Ausnahmen und besonders auf diese Ausnahmen wird im Weiteren näher eingegangen. Am Ende dieses Abschnitts folgt eine Zusammenfassung. Die prozentuale Veränderung der Laufzeit und des Energieverbrauchs entspricht bei den meisten SQL-Befehlen der prozentualen Veränderung der Datenbankgröße. In Tabelle 5.13 sind die jeweiligen prozentualen Veränderungen bei einem Datenbankwechsel angegeben. Bei einem Wechsel von der 1 GB-Datenbank zur 5 GB-Datenbank (Zunahme der Datenbankgröße um 400 %) nehmen die Laufzeit und der Energieverbrauch bei den meisten Befehlen ebenfalls um ungefähr 400 % zu. Bei dem Wechsel von der 5 GB-Datenbank zur 10 GB-Datenbank (Zunahme der Datenbankgröße um 100 %) beträgt die Zunahme ungefähr 100 %.

Die deutlichen Ausnahmen bilden Befehl 6 (Sortierung) und die Befehle 8, 9, 10 und 12 (Befehle mit einer WHERE-Klausel). Bei Befehl 6 liegt die Zunahme der Laufzeit bei einem Wechsel von der 1 GB-Datenbank auf die 5 GB-Datenbank mit 666,49 % über dem erwarteten Wert von ungefähr 400 %. Der Energieverbrauch liegt mit 578,59 % ebenfalls deutlich über dem erwarteten Wert. Die unerwartet hohe Zunahme der Laufzeit liegt darin begründet, dass auf der 1 GB-Datenbank die Berechnung noch fast vollständig im Arbeitsspeicher durchgeführt werden konnte (siehe Abbildung 5.8). Auf der 5 GB-Datenbank reichte der Arbeitsspeicher hingegen nicht mehr aus und es kam zu hoher Festplattenaktivität (siehe Abbildung 5.9). Die Zunahme der Laufzeit ist geringer als die Zunahme des Energieverbrauchs aus dem bereits bekannten Grund. In der Phase der hohen Festplattenaktivität, welche die Laufzeit maßgeblich erhöht, ist der Prozessor nicht vollständig ausgelastet und der Energieverbrauch liegt bei nur ungefähr 100 Watt statt bei 140 Watt bei voller Prozessorauslastung. Der

| | Wechsel von 1 GB auf 5 GB | | Wechsel von 5 GB auf 10 GB | |
|---|---|---|---|---|
| Nr. | Laufzeit-Veränderung [%] | Joule-Veränderung [%] | Laufzeit-Veränderung [%] | Joule-Veränderung [%] |
| 1 | +396,50 | +396,99 | +100,88 | +101,91 |
| 2 | +404,77 | +408,08 | +99,22 | +100,00 |
| 3 | +387,54 | +391,13 | +98,90 | +100,62 |
| 4 | +398,26 | +399,81 | +98,72 | +99,49 |
| 5 | +397,14 | +398,23 | +101,64 | +102,42 |
| 6 | +666,49 | +578,59 | +108,77 | +107,44 |
| 7 | +426,47 | +417,75 | +109,08 | +108,75 |
| 8 | -1,18 | -1,66 | +2,75 | +2,90 |
| 9 | +153,27 | +152,35 | +2,08 | +2,90 |
| 10 | +149,77 | +150,99 | +3,74 | +4,19 |
| 11 | +383,21 | +386,29 | +102,89 | +104,17 |
| 12 | +4,50 | +4,62 | +1,71 | +1,82 |
| 13 | +395,77 | +396,43 | +101,78 | +103,08 |
| 14 | +404,95 | +408,03 | +97,21 | +98,40 |
| 15 | N/A | N/A | N/A | N/A |
| 16 | N/A | N/A | N/A | N/A |
| 17 | N/A | N/A | N/A | N/A |
| 18 | N/A | N/A | N/A | N/A |
| 19 | +406,61 | +410,63 | +99,34 | +99,90 |
| 20 | +450,33 | +455,54 | +125,22 | +125,36 |
| 21 | +430,01 | +438,24 | +115,87 | +116,56 |
| 22 | +417,23 | +416,02 | +95,03 | +97,33 |
| 23 | +405,64 | +409,81 | +98,05 | +98,75 |
| 24 | +379,66 | +390,05 | +100,65 | +101,49 |
| 25 | N/A | N/A | N/A | N/A |
| 26 | +424,31 | +417,79 | +91,43 | +93,63 |
| 27 | +405,89 | +406,33 | +94,50 | +96,45 |
| 28 | +407,66 | +408,17 | +96,65 | +97,71 |
| 29 | +411,05 | +408,41 | +89,64 | +94,49 |
| 30 | +379,96 | +388,80 | +98,07 | +99,29 |

Tabelle 5.13: Prozentuale Veränderung der Laufzeit und des Energieverbrauchs der 30 SQL-Befehle nach Änderung der Datenbankgröße von 1 GB auf 5 GB und von 5 GB auf 10 GB.


Abbildung 5.8: SQL-Befehl 6, 3 GB Arbeitsspeicher, 1 GB-Datenbank.

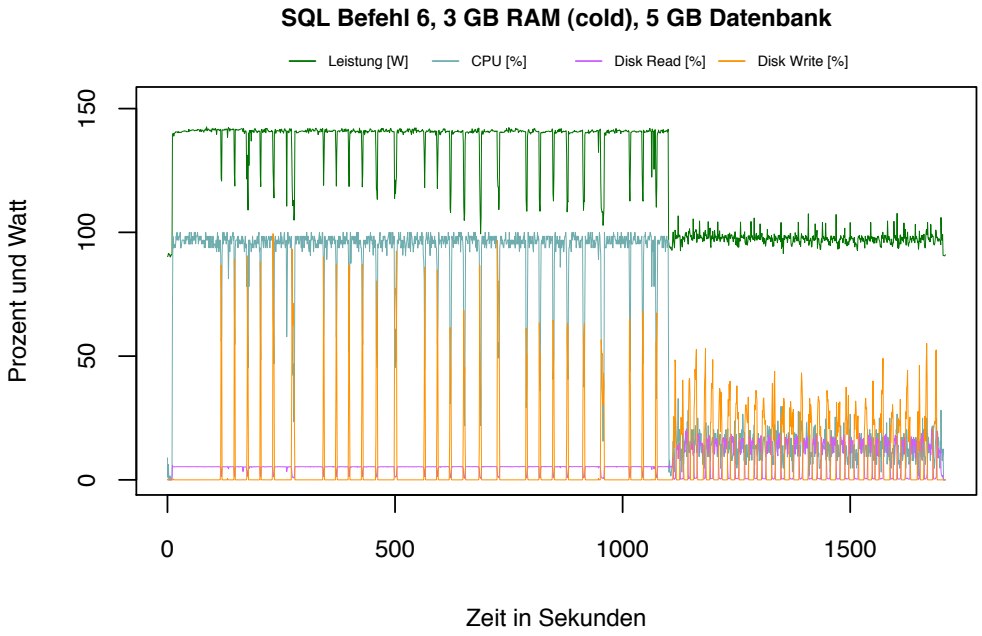



Abbildung 5.9: SQL-Befehl 6, 3 GB Arbeitsspeicher, 5 GB-Datenbank.

Energieverbrauch erhöht sich prozentual nicht in demselben Maße wie die Laufzeit. Bei einem Wechsel von der 5 GB-Datenbank zur 10 GB-Datenbank liegt die Veränderung der Laufzeit und des Energieverbrauchs allerdings wieder im erwarteten Bereich von ungefähr 100 %, da bei beiden Datenbankgrößen die Berechnung nicht mehr im Arbeitsspeicher durchgeführt werden konnte.

Erstaunlich sind die Ergebnisse der Befehle 8, 9, 10 und 12. Hier liegt die Zunahme der Laufzeit und des Energieverbrauchs nur im Bereich von 0 % bis 150 %. Insbesondere bei Befehl 8 und 12 ist trotz unterschiedlicher Datenbankgröße nahezu keine Veränderung festzustellen. Bei diesen Befehlen handelt es sich um einfache Selektionen, bei denen jeweils die Ergebnismenge mit einer WHERE-Klausel auf wenige Zeilen eingeschränkt wird. Einfache WHERE-Klauseln, wie wir sie bei diesen Befehlen verwendet haben, beispielsweise ein Datumsvergleich, können demnach in Caché effizient berechnet werden und die Datenbankgröße wirkt sich kaum auf die Laufzeit und den Energieverbrauch aus.

### Zusammenfassung

Die Hypothese (siehe Abschnitt 5.3.4), welche am Anfang dieses Abschnitts aufgestellt wurde, konnte durch die Messergebnisse bestätigt werden. Die Gesamtlaufzeit und die gesamte benötigte Energie zur Ausführung aller 30 SQL-Befehle wachsen linear mit der Datenbankgröße an. Der lineare Zusammenhang zwischen Datenbankgröße und Gesamtlaufzeit bzw. Energieverbrauch ist in Abbildung 5.10 und Abbildung 5.11 grafisch dargestellt.

Bei einer Einzelbetrachtung trifft dies allerdings auf bestimmte SQL-Befehle nicht zu. Bei speicherintensiven Berechnungen (Sortierungen) ist zu berücksichtigen, dass ab einer bestimmten Datenbankgröße die Berechnung nicht mehr im Arbeitsspeicher durchgeführt werden kann und die Laufzeit durch eine erhöhte I/O-Aktivität in diesem Fall deutlich ansteigt. Selektionen mit einfach zu berechnenden WHERE-Bedingungen konnten bei unseren Messungen auch auf großen Datenbanken effizient berechnet werden und die Laufzeit und der Energieverbrauch nahmen kaum zu.

Die Gesamtlaufzeit aller SQL-Befehle nahm bei einem Wechsel von der
1 GB-Datenbank zur 5 GB-Datenbank (eine Zunahme von 400 % der Datenbankgröße) um 402,96 % zu. Gleichzeitig nahm die gesamte benötigte Energie zur Ausführung aller Befehle um 397,72 % zu. Bei einem Wechsel von der 5 GB-Datenbank zur 10 GB-Datenbank (eine Zunahme von 100 % der Datenbankgröße) beträgt die Zunahme der Gesamtlaufzeit 97,90 % und die Zunahme der gesamten benötigten Energie beträgt 98,81 %. Damit lässt sich insgesamt kein Unterschied zwischen der Veränderung der Laufzeit und der Veränderung des Energieverbrauchs einer Datenbank bei unterschiedlich großen Datenbanken feststellen.

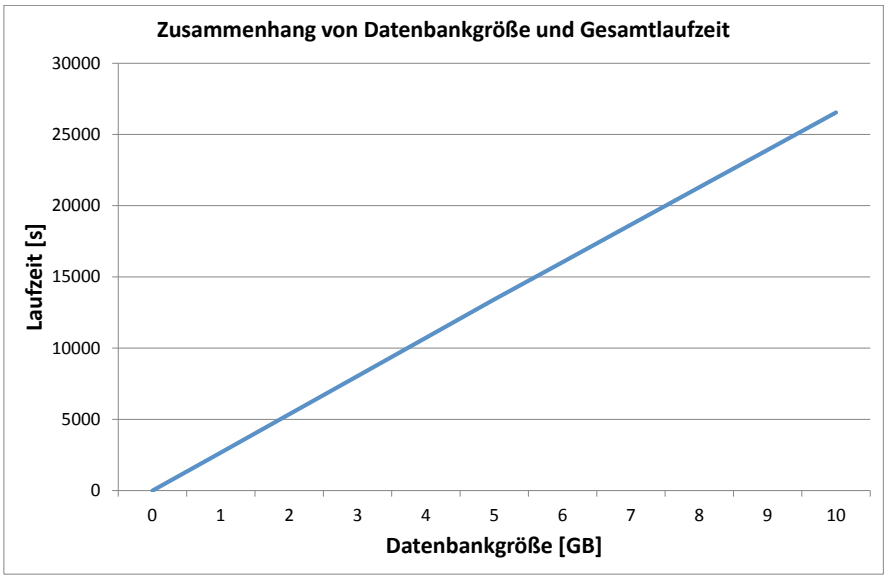



Abbildung 5.10: Zusammenhang von Datenbankgröße und Gesamtlaufzeit.

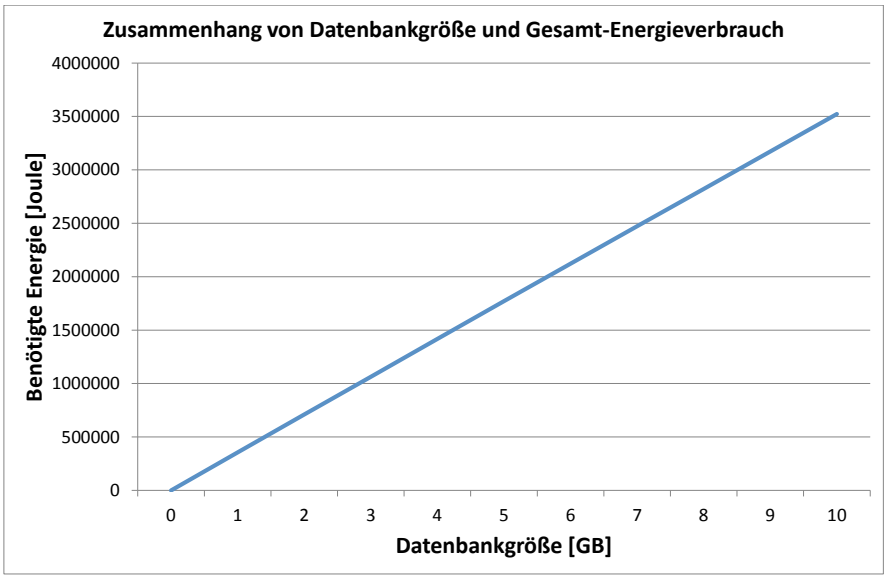



Abbildung 5.11: Zusammenhang von Datenbankgröße und Gesamt-Energieverbrauch.

### 5.3.5 Betrachtung der Befehlsgruppen bezüglich Veränderungen der Datenbankgröße

Die folgenden Untersuchungen sollen zeigen, um welchen Faktor sich die Laufzeit der einzelnen Gruppen verändert, wenn sich die Datenbankgröße um einen bestimmten Faktor verändert. Da die Messungen der 30 SQL-Abfragen auf der 1 GB großen Datenbank, der 5 GB großen Datenbank und der 10 GB großen Datenbank durchgeführt wurden, stehen zur Analyse die zwei Wachstumsfaktoren 5 (1 GB Datenbank auf 5 GB Datenbank) und 2 (5 GB Datenbank auf 10 GB Datenbank) zur Verfügung. Die Tabellen 5.14 und 5.15 zeigen, um welchen Faktor sich die Gesamtlaufzeit bzw. der Gesamt-Energieverbrauch der sieben Gruppen verändert, wenn sich die Datenbankgröße um den Faktor 5 bzw. den Faktor 2 vergrößert. Hierfür wurden die Zeiten und die Joule-Werte alle einer Gruppe zugeordneten Befehle aufsummiert, wobei bei allen Datenbankgrößen wieder nur die SQL-Befehle berücksichtigt wurden, welche auf allen Datenbanken ohne Timeout ausgeführt werden konnten. Die SQL-Befehle 6, 15 bis 8 sowie 25 wurden also nicht berücksichtigt, um eine bessere Vergleichbarkeit zu gewährleisten.

| | 1 GB Datenbank | | 5 GB Datenbank | | Änderungsfaktor | |
|---|---|---|---|---|---|---|
| Gruppe | Laufzeit [s] | Joule | Laufzeit [s] | Joule | Laufzeit | Joule |
| Gruppe 1 | 88,17 | 11.382,66 | 437,18 | 56.751,24 | 5,0 | 5,0 |
| Gruppe 2 | 266,48 | 36.281,33 | 1.326,4 | 181.096,02 | 5,0 | 5,0 |
| Gruppe 3 | 159,08 | 21.820,58 | 837,49 | 112.976,77 | 5,3 | 5,2 |
| Gruppe 4 | 325,58 | 43.054,75 | 948,66 | 126.019,02 | 2,9 | 2,9 |
| Gruppe 5 | 83,57 | 11.122,49 | 422,00 | 56.505,83 | 5,0 | 5,1 |
| Gruppe 6 | 418,84 | 56.654,82 | 2.177,04 | 296.740,38 | 5,2 | 5,2 |
| Gruppe 7 | 1.104,29 | 144.746,84 | 5.572,80 | 731.924,76 | 5,0 | 5,1 |

Tabelle 5.14: Faktor für die Veränderung der Laufzeit und des Energieverbrauchs nach Änderung der Datenbankgröße um Faktor 5.

Die Abbildung 5.12 verdeutlicht, um welchen Faktor die Laufzeit und der Energieverbrauch der verschiedenen Gruppen bei einer Vergrößerung der Datenbank von 1 GB auf 5 GB, also um den Faktor 5, anwachsen. Es wird deutlich, dass der Energieverbrauch und die Laufzeit bei den meisten Gruppen ebenfalls um den Faktor 5 steigen, sich also proportional zur Vergrößerung der Datenbank verhalten. Besonders interessant ist jedoch, dass die Laufzeit von Gruppe 4 nicht um den Faktor 5 ansteigt, sondern nur um den Faktor 3. Selektionen werden demnach sehr effizient durchgeführt und ihre Laufzeit bzw. ihr Energieverbrauch wächst nicht proportional zur Datenbankgröße. Bei der Vergrößerung der Datenbank von 5 GB auf 10 GB, also um den Faktor 2, verdeutlicht die Abbildung 5.13, dass sich die Laufzeit und der Energieverbrauch auch hier ebenfalls bei 6 der 7 Gruppen proportional zur Vergrößerung der Datenbank verhalten. Die Datenbank wurde um den Faktor 2 vergrößert, und die Laufzeiten sämtlicher Gruppen, mit Ausnahme von Gruppe 4 (Selektion),

| | 5 GB Datenbank | | 10 GB Datenbank | | Änderungsfaktor | |
|---|---|---|---|---|---|---|
| Gruppe | Laufzeit [s] | Joule | Laufzeit [s] | Joule | Laufzeit | Joule |
| Gruppe 1 | 437,18 | 56.751,24 | 872,12 | 113.891,96 | 2,0 | 2,0 |
| Gruppe 2 | 1.326,4 | 181.096,02 | 2.652,51 | 363.505,84 | 2,0 | 2,0 |
| Gruppe 3 | 837,49 | 112.976,77 | 1.751,01 | 235.843,31 | 2,1 | 2,1 |
| Gruppe 4 | 948,66 | 126.019,02 | 1.530,19 | 204.694,87 | 1,6 | 1,6 |
| Gruppe 5 | 422,00 | 56.505,83 | 832,21 | 112.109,08 | 2,0 | 2,1 |
| Gruppe 6 | 2.177,04 | 296.740,38 | 4.494,34 | 615.116,28 | 2,1 | 2,1 |
| Gruppe 7 | 5.572,80 | 731.924,76 | 1.970,12 | 260.123,34 | 1,9 | 2,0 |

Tabelle 5.15: Faktor für die Veränderung der Laufzeit und des Energieverbrauchs nach Änderung der Datenbankgröße um Faktor 2.

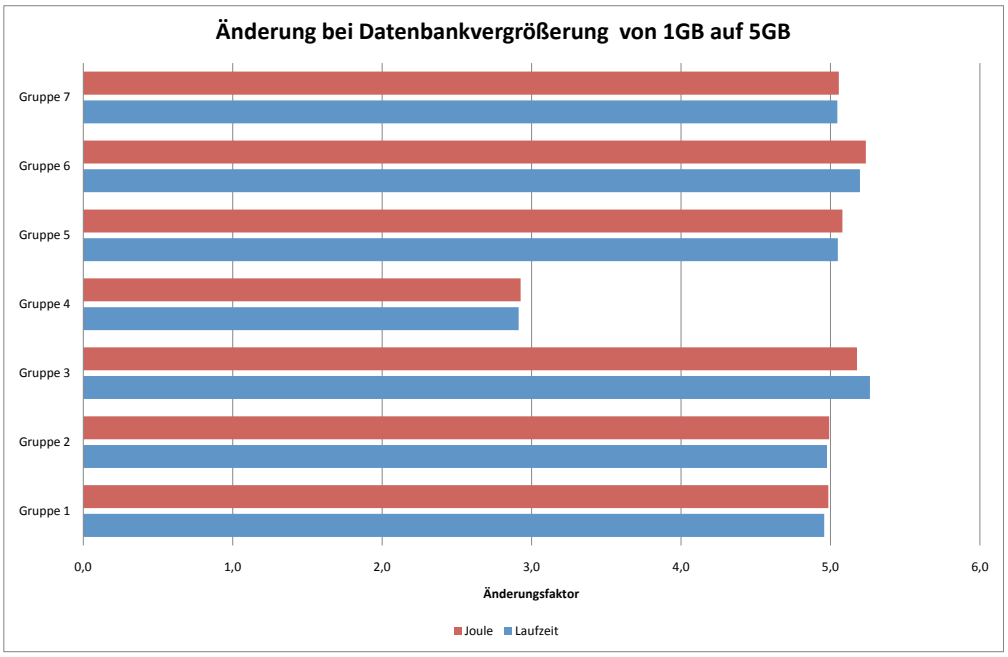



Abbildung 5.12: Änderungsfaktor der durchschnittlichen Laufzeit und des durchschnittlichen Energieverbrauchs aller Gruppen, bei der Vergrößerung der Datenbank von 1 GB auf 5 GB.

haben sich verdoppelt. Der nichtlineare Anstieg des Energieverbrauchs bei Gruppe 4 ist wieder ein Indiz für die effiziente Implementierung der Selektion, die bei dieser Messreihe noch nicht auf einen Index zurückgreifen konnte.

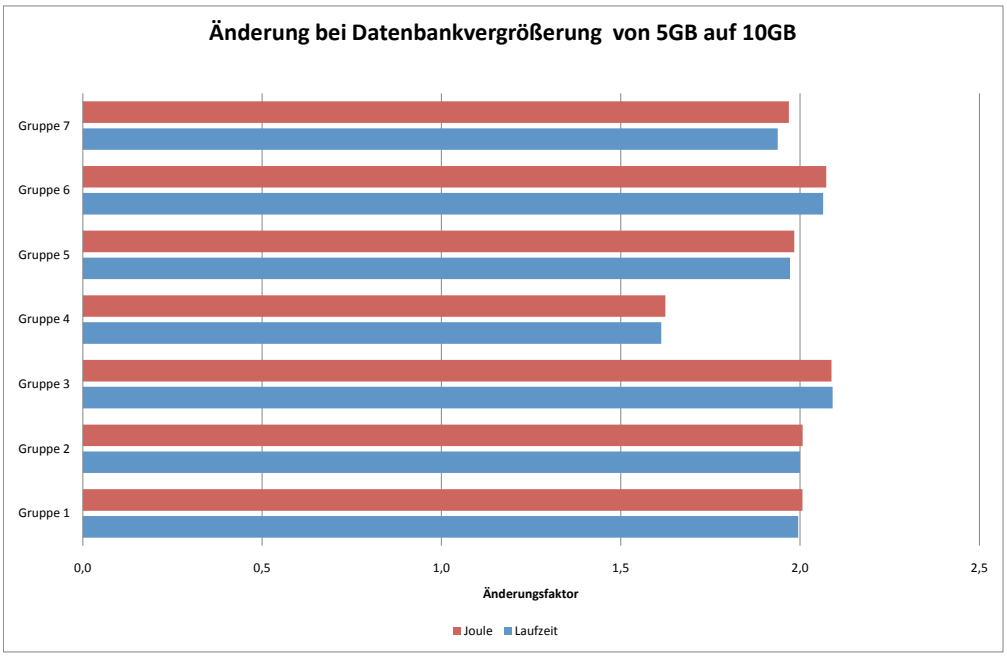



Abbildung 5.13: Änderungsfaktor der durchschnittlichen Laufzeit und des durchschnittlichen Energieverbrauchs aller Gruppen, bei der Vergrößerung der Datenbank von 5 GB auf 10 GB.

**Fazit:** Die Vergrößerung der Datenbank, genauer gesagt der einzelnen Tabellen um denselben Faktor, führt zu einer Erhöhung der Laufzeit und somit zu einer Energieverbrauchssteigerung um genau diesen Faktor. Die einzige Ausnahme bilden die Selektionen, deren Laufzeit um einen geringeren Faktor ansteigt.

## 5.4 Zusammenfassung der Ergebnisse

### Auswirkung der Arbeitsspeichergröße auf den Energieverbrauch der Datenbank Caché

Wurde der dem DBMS Caché zugewiesene Arbeitsspeicher von 256 MB auf 3 GB erhöht, dann nahm der Energieverbrauch des von uns definierten Workloads (siehe Abschnitt 5.2.2) auf der 1 GB-Datenbank um lediglich 0,16 % ab. Die Berechnungen auf der 1 GB-Datenbank benötigten den zusätzlichen Arbeitsspeicher demnach

nicht. Auf der 5 GB-Datenbank betrug die Reduktion des Energieverbrauchs 3,63 % und ist damit am größten. Die Berechnungen auf dieser Datenbankgröße profitierten am meisten von der Arbeitsspeichererweiterung, da das Verhältnis zwischen Größe der Arbeitsspeicher-Erweiterung und der Datenbankgröße für diese Datenbank am größten ist (die 1 GB-Datenbank ausgenommen, die den zusätzlichen Arbeitsspeicher nicht benötigte). Auf der 10 GB-Datenbank betrug die Reduktion 2,08 % (siehe Abschnitt 5.3.2).

Einzelne Typen von SQL-Befehlen können von diesen Gesamtwerten merklich abweichen. So profitieren Sortierungen (*ORDERED BY*) und Verknüpfungen (*JOIN*) in besonderem Maße von einem größeren Arbeitsspeicher. Für die Gruppe der Sortierbefehle (Gruppe 3) beträgt die Energiereduktion je nach Datenbankgröße bis zu 5,63 %. Für die Gruppe der Verknüpfungen (Gruppe 6) wurde eine Reduktion von bis zu 15,23 % festgestellt. Diese Befehls-Typen sind es auch, welche den Gesamtwert für die Energiereduktion maßgeblich beeinflussen. Für Aggregatfunktionen, Selektionen oder Update-Funktionen hatte die Erweiterung des Arbeitsspeichers auf den Energieverbrauch einen geringen oder keinen Einfluss (siehe Abschnitt 5.3.3).

### Auswirkung der Datenbankgröße auf den Energieverbrauch der Datenbank Caché

Der Energieverbrauch unseres Workloads verhält sich insgesamt proportional zu der Datenbankgröße. Bei einer Einzelbetrachtung der SQL-Befehle gibt es Besonderheiten. Bei speicherintensiven Berechnungen (beispielsweise Sortierungen) ist zu berücksichtigen, dass ab einer bestimmten Datenbankgröße die Berechnung nicht mehr im Arbeitsspeicher durchgeführt werden kann und die Laufzeit durch erhöhte I/O-Aktivität diesem Fall deutlich ansteigt. Selektionen mit einfach zu berechnenden WHERE-Bedingungen konnten bei unseren Messungen auch auf großen Datenbanken effizient berechnet werden und die Laufzeit und der Energieverbrauch nahmen weniger stark zu (siehe Abschnitt 5.3.4).

Eine Betrachtung der nach Befehls-Typen klassifizierten Gruppen bestätigt die Gesamtbetrachtung. Auch für die einzelnen Gruppen ist das Verhältnis von Energieverbrauch und Datenbankgröße proportional. Eine Ausnahme bildet die Gruppe der Selektionen (Gruppe 4). Bei einer Vergrößerung der Datenbank von 1 GB auf 5 GB (Faktor 5) nahm der Energieverbrauch nur um den Faktor 2,9 zu. Bei dem Wechsel von der 5 GB- auf die 10 GB-Datenbank (Faktor 2) änderte sich der Energieverbrauch nur um den Faktor 1,6. Daran wird deutlich, dass die Datenbankgröße keinen so großen Einfluss auf den Energieverbrauch von (einfachen) Selektionen hat wie auf andere Befehls-Typen (siehe Abschnitt 5.3.5).

# 6 Energieoptimierungen

## 6.1 TPC-H Benchmark-Schema in Caché

### 6.1.1 Einleitung

*Intersystems Caché* bietet diverse Möglichkeiten, die Daten einer Datenbank zu verwalten und bereitzustellen. Im Folgenden werden wir uns damit beschäftigen, inwieweit die Konfigurationsmöglichkeiten der Datenbank Caché benutzt werden können, um den Workload des TPC-H Benchmarks energieeffizient abzuarbeiten. Der Fokus liegt hierbei bei der Datenhaltung und Organisation sowie der Nutzung von Datenbankkonfigurationen und Schemadefinitionen, wie einem Index. Im Vergleich zum vorangegangenen Kapitel nehmen wir hier die 22 TPC-H Abfragen als Grundlage für die Optimierungen. Die TPC-H Benchmark simulieren einen Workload wie er im Unternehmensumfeld zu finden ist.

### 6.1.2 Datenmodell und Implementierung

Die Daten, die dem TPC-H zugrunde liegen, sind nach der Generierung durch das Programm *DBGEN* als normale Textdateien auf der Festplatte abgespeichert. Für jede Tabelle des TPC-H Schemas existiert eine Textdatei, in der die Daten durch ein Pipe-Symbol getrennt sind. Dies ist hier am Beispiel der Tabelle *nation* dargestellt:

```
0|ALGERIA |0|haggle. carefully final deposits detect slyly ai|
1|ARGENTINA|1|al foxes promise slyly according to the regu...|
2|BRAZIL|1|y alongside of the pending deposits. carefully ...|
3|CANADA|1|eas hang ironic, silent packages. slyly regular...|
4|EGYPT|4|y above the carefully unusual theodolites. final...|
5|ETHIOPIA|0|ven packages wake quickly. regu|
6|FRANCE|3|refully final requests. regular, ironi|
7|GERMANY|3|l platelets. regular accounts x-ray: unusual, ...|
8|INDIA|2|ss excuses cajole slyly across the packages. dep...|
9|INDONESIA|2| slyly express asymptotes. regular deposits ...|
10|IRAN|4|efully alongside of the slyly final dependencies...|
11|IRAQ|4|nic deposits boost atop the quickly final reques...|
12|JAPAN|2|ously. final, express gifts cajole a|
13|JORDAN|4|ic deposits are blithely about the carefully r...|
14|KENYA|0| pending excuses haggle furiously deposits. pen...|
15|MOROCCO|0|rns. blithely bold courts among the closely r...|
16|MOZAMBIQUE|0|s. ironic, unusual asymptotes wake blithely r|
17|PERU|1|platelets. blithely pending dependencies use flu...|
```

```
18|CHINA|2|c dependencies. furiously express notornis slee...|
19|ROMANIA|3|ular asymptotes are about the furious multipl...|
20|SAUDI ARABIA|4|ts. silent requests haggle. closely expr...|
21|VIETNAM|2|hely enticingly express accounts. even, final |
22|RUSSIA|3| requests against the platelets use never acco...|
23|UNITED KINGDOM|3|eans boost carefully special requests ...|
24|UNITED STATES|1|y final packages. slow foxes cajole qui...|
```

Die Rohdaten lassen sich über das Webportal von Caché einlesen. Um die Tabellenstruktur für den TPC-H Benchmark zu erstellen, eignet sich folgender SQL-Befehl. Der durch den TPC-H Benchmark bereitgestellt wird.

```
CREATE TABLE LINEITEM ( L_ORDERKEY    INTEGER NOT NULL,
                        L_PARTKEY     INTEGER NOT NULL,
                        L_SUPPKEY     INTEGER NOT NULL,
                        L_LINENUMBER  INTEGER NOT NULL,
                        L_QUANTITY    DECIMAL(15,2) NOT NULL,
                        L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL,
                        L_DISCOUNT    DECIMAL(15,2) NOT NULL,
                        L_TAX         DECIMAL(15,2) NOT NULL,
                        L_RETURNFLAG  CHAR(1) NOT NULL,
                        L_LINESTATUS  CHAR(1) NOT NULL,
                        L_SHIPDATE    DATE NOT NULL,
                        L_COMMITDATE  DATE NOT NULL,
                        L_RECEIPTDATE DATE NOT NULL,
                        L_SHIPINSTRUCT CHAR(25) NOT NULL,
                        L_SHIPMODE     CHAR(10) NOT NULL,
```

Die SQL-Befehle werden über das Terminal oder das Webportal ausführt. Bevor die Rohdaten eingelesen werden können, müssen Anpassungen an der Konfiguration der Datenbank gemacht werden. Ziel ist es dabei, einen konsistenten Zustand für unsere Versuche herzustellen, um Einflüsse die systembedingt sind, zu minimieren.

**Datenbankgröße und Arbeitsspeicher:** Wir arbeiten in diesem Kapitel mit einer Datenbankgröße von 10 GB und reservieren der Datenbank 3 GB Arbeitsspeicher. Die Auswirkungen der Arbeitsspeicher- und Datenbankgröße wurden bereits im vorherigen Kapitel (siehe Abschnitt 5) untersucht.

Die Konfigurationsänderungen an der Datenbank werden zusätzlichen Speicher benötigen. Deswegen setzten wir vor dem Einlesen der Daten eine Größe von 15 GB für die Datenbankdatei fest. Somit ist sichergestellt, dass Daten, die zusätzlich zu den Datensätzen erzeugt werden, in benachbarten Speicherbereichen auf der Festplatte gespeichert werden. Dadurch wird eine Fragmentierung der Daten auf der Festplatte verhindert.

**Datenbank-Klassen:** Die Datenbank Caché ist eine proprietäre, im Kern hierarchische Datenbank. Daher ist eine objektorientierte und relationale Bearbeitung der Daten möglich. Tabellen sind in der objektorientierten Darstellung Klassen, wie am Beispiel der Tabelle *region* gezeigt wird:

```
Class User.REGION Extends %Persistent [ClassType = persistent,
    DdlAllowed, Owner = UnknownUser,
    ProcedureBlock, SqlRowIdPrivate,
    SqlTableName = REGION,
    StorageStrategy = Default]
{
  Property RREGIONKEY As %Library.Integer(
      MAXVAL = 2147483647, MINVAL = 0)
            [Required, SqlColumnNumber = 2,
             SqlFieldName = R_REGIONKEY];

  Property RNAME As %Library.String(MAXLEN = 25)
      [Required, SqlColumnNumber = 3,
       SqlFieldName = R_NAME];

  Property RCOMMENT As %Library.String(MAXLEN = 152)
      [SqlColumnNumber = 4, SqlFieldName = R_COMMENT];

}
```

In dieser Klassendefinition sind die Parameter für die Tabelle und deren einzelne Spalten festgehalten, alle SQL-relevanten Definitionen lassen sich unmittelbar in der Klassendefinition finden, zusätzlich sind noch Aspekte der objektorientierten Bearbeitung implementiert. Bearbeitungen des Klassenschemas haben unmittelbare Auswirkungen auf die relationale Darstellung der Tabelle.

**Globals:** Globals sind persistente Variablen in denen die Datensätze der Datenbank gespeichert sind - für jeden Datensatz ein Global. Zur eindeutigen Identifikation jedes Datensatzes einer Tabelle übernimmt der Global den Primärschlüssel des Datensatzes als ID. Im Folgenden ist beispielhaft die Menge der Globals angegeben, die der Tabelle *nation* zugeordnet ist.

```
1:^User.NATIOND(0)= $lb("",0,"ALGERIA",0,"haggle. carefu")
2:^User.NATIOND(1)=$lb("",1,"ARGENTINA",1,"al foxes ...")
3:^User.NATIOND(2)=$lb("",2,"BRAZIL",1,"y alongside ...")
4:^User.NATIOND(3)=$lb("",3,"CANADA",1,"eas hang iro...")
5:^User.NATIOND(4)=$lb("",4,"EGYPT",4,"y above the c...")
6:^User.NATIOND(5)=$lb("",5,"ETHIOPIA",0,"ven packag...")
7:^User.NATIOND(6)=$lb("",6,"FRANCE",3,"refully fina...")
8:^User.NATIOND(7)=$lb("",7,"GERMANY",3,"l platelets...")
```

```
9:^User.NATIOND(8)=$lb("",8,"INDIA",2,"ss excuses ca...")
10:^User.NATIOND(9)=$lb("",9,"INDONESIA",2,"slyly exp...")
11:^User.NATIOND(10)=$lb("",10,"IRAN",4,"efully alongs...")
12:^User.NATIOND(11)=$lb("",11,"IRAQ",4,"nic deposits ...")
13:^User.NATIOND(12)=$lb("",12,"JAPAN",2,"ously. final...")
14:^User.NATIOND(13)=$lb("",13,"JORDAN",4,"ic deposits...")
15:^User.NATIOND(14)=$lb("",14,"KENYA",0,"pending excu...")
16:^User.NATIOND(15)=$lb("",15,"MOROCCO",0,"rns. blith...")
17:^User.NATIOND(16)=$lb("",16,"MOZAMBIQUE",0,"s. iron...")
18:^User.NATIOND(17)=$lb("",17,"PERU",1,"platelets. bl...")
19:^User.NATIOND(18)=$lb("",18,"CHINA",2,"c dependenci...")
20:^User.NATIOND(19)=$lb("",19,"ROMANIA",3,"ular asymp...")
21:^User.NATIOND(20)=$lb("",20,"SAUDI ARABIA",4,"ts. s...")
22:^User.NATIOND(21)=$lb("",21,"VIETNAM",2,"hely entic...")
23:^User.NATIOND(22)=$lb("",22,"RUSSIA",3,"requests ag...")
24:^User.NATIOND(23)=$lb("",23,"UNITED KINGDOM",3,"ean...")
25:^User.NATIOND(24)=$lb("",24,"UNITED STATES",1,"y fi...")
```

Der Datensatz mit der sechsten ID (Primärschlüssel) ist beispielsweise wie folgt notiert:

```
^User.NATIOND(5)        = $lb("",5,"ETHIOPIA",0,"ven packag...")
```

**Primärschlüssel, Fremdschlüssel und Typ-Definitionen:** Um das TPC-H Schema direkt abzubilden, sind verschiedene Modifikationen notwendig, nachdem die Klassen erzeugt wurden. Caché fügt automatisch jeder erstellten Tabelle zwei Spalten hinzu („ID“ und „x_classname“), welche beide interne Repräsentanten sind, die Caché zum Verwalten der Daten benötigt. Damit einhergehend ist die Tatsache, dass Caché standardmäßig seine eigene ID-Spalte als Primärschlüssel festlegt. Da die Primärschlüssel durch den TPC-H Benchmark festgelegt sind, müssen wir die Implementierung verändern, so dass die vom TPC-Benchmark festgelegten Spalten als Primärschlüssel benutzt werden. Die Änderung des Primärschlüssels erfolgt in Caché über die Änderung des Klassenschemas im *Caché Studio*. Die Tabelle 6.1 zeigt die Primärschlüssel wie sie im TPC-H Benchmark gefordert sind.

Im TPC-H Benchmark existiert eine Auflistung von Fremdschlüsseln, welche auf immer auf einen Primärschlüssel referenzieren. Fremdschlüssel werden benutzt, um Relationen zwischen Tabellen zu definieren; sie dienen auch der Konsistenzprüfung der Daten beim Einlesen in die Datenbank. Wir definieren im Klassenschema der jeweiligen Tabelle zusätzlich noch die in Tabelle 6.2 aufgeführten Fremdschlüssel.

Die Definition der Primärschlüssel und Fremdschlüssel erfolgt über die Klassendefinitionen, wie beispielhaft oben gezeigt. Das Anlegen eines Primärschlüssels in Caché geschieht über das Definieren eines Index mit speziellem Attribut; die Fremdschlüssel referenzieren dann auf den entsprechend definierten Index. Für die Tabelle Region sieht der hinzugefügte Code folgendermaßen aus:

| Primary Key |
|---|
| P_PARTKEY |
| S_SUPPKEY |
| PS_PARTKEY, PS_SUPPKEY |
| C_CUSTKEY |
| O_ORDERKEY |
| L_ORDERKEY, L_LINENUMBER |
| N_NATIONKEY |
| R_REGIONKEY |

Tabelle 6.1: Primärschlüssel des TPC-H Schemas.

| Foreign Key | Referenz |
|---|---|
| PS_PARTKEY | P_PARTKEY |
| PS_SUPPKEY | S_SUPPKEY |
| C_NATIONKEY | N_NATIONKEY |
| O_CUSTKEY | C_CUSTKEY) |
| L_ORDERKEY | O_ORDERKEY |
| L_PARTKEY | P_PARTKEY |
| L_SUPPKEY | S_SUPPKEY |
| L_PARTKEY, L_SUPPKEY | PS_PARTKEY, PS_SUPPKEY |
| N_REGIONKEY | R_REGIONKEY |

Tabelle 6.2: Fremdschlüssel des TPC-H Schemas.

```
Index REGIONPRIMARYIDX On RREGIONKEY [IdKey, PrimaryKey, Unique];
```

Der Vorteil dieser Datenhaltung ist, dass die Primärschlüssel immer sortiert sind, da die Spalten der Definition eines Index unterliegen. Dies ist für den Workload des TPC-H von Vorteil, da die Abfragen häufig Beziehungen abfragen, die sich auf Primärschlüssel beziehen.
Die Fremdschlüsseldefinitionen sehen wie folgt aus:

```
ForeignKey NATIONIDXFOREIGN(NREGIONKEY) References
    User.REGION(REGIONPRIMARYIDX);
```

Dies ist eine Referenz der Spalte *nregionkey* aus Tabelle *nation* auf den Primärschlüssel der Tabelle *region*.

**Angabe von Wertebereichen:** Um die Möglichkeiten von Caché voll ausnutzen zu können, müssen für die Schlüsselspalten die Wertebereichsdefinition MINVAL und MAXVAL genau definiert werden. Bei der Nutzung von Indizes sind diese Werte für die Datenbank Caché relevant. Unmittelbar nach dem Einlesen der Daten sind die Werte für die Primärschlüssel so gesetzt: MAXVAL = 2147483647, MINVAL = -2147483647. Wir ändern den MINVAL bei allen Primärschlüsseln auf 1; eine Ausnahme bilden die Primärschlüssel der Tabellen *region* und *nation* für diese Tabellen muss der MINVAL 0 sein, da solche Primärschlüssel in Rohdaten vorhanden sind. Ein Beispiel mit festgelegtem Wertebereich:

```
Property RREGIONKEY As
    %Library.Integer(MAXVAL = 2147483647, MINVAL = 0)
    [ Required, SqlColumnNumber = 2,
        SqlFieldName = R_REGIONKEY ];
```

**ExtentSize Selectivity:** Nachdem die Daten nun importiert wurden und TPC-H Schema konform abgebildet wurde, ist es nun nötig, Parameter einzustellen, die für den Abfrageoptimierer von großer Wichtigkeit sind: *Extentsize* und *Selectivity*. Diese Werte werden pro Tabelle festgelegt. Extentsize gibt an wie viele Zeilen in der Tabelle gespeichert sind, und damit die Größe der Tabelle. Die exakte Größe für Extensize für unser Datenmodell mit 10 GB entnehmen wir Tabelle 6.3 aus der Spalte „Zeilenanzahl“.
Der Wert für die Selektivität wird pro Spalte einer Tabelle festgelegt. Die Selektivität ist ein statistischer Wert, der die Werteverteilung einer Spalte wiedergibt. Spalten, die viele verschieden Werte beinhalten, haben eine niedrige Selektivität.
Der Abfrageoptimierer benutzt *Extentsize* und *Selectivity*, um die Kosten für die Ausführung einer Abfrage zu berechnen. Wenn es mehrere Möglichkeiten gibt, eine Abfrage auszuführen, dann entscheidet sich der Abfrageoptimierer für die Möglichkeit, die sich nach seiner Heuristik als die günstigere erweist. Da die berechneten Kosten von der *Extentsize* und *Selectivity* abhängen, ist es notwendig, dass die Werte aktuell sind, um ein effizientes Arbeiten des Abfrageoptimierers zu gewährleisten.

| Tabelle | Zeilenanzahl | Größe (in KB) | Anteil (in %) |
|---|---|---|---|
| lineitem | 59.979.500 | 7.652.065 | 69,2 |
| orders | 15.000.000 | 1.722.847 | 15,6 |
| partsupp | 8.000.000 | 1.184.425 | 10,7 |
| customer | 1.500.000 | 240.574 | 2,2 |
| part | 2.000.000 | 239.594 | 2,2 |
| supplier | 100.000 | 13.942 | $< 1$ |
| nation | 25 | 3 | $< 1$ |
| region | 5 | 1 | $< 1$ |

Tabelle 6.3: Größenverteilung der Tabellen des TPC-H.

### 6.1.3 Messergebnisse der TPC-H Implementierung (Referenzdaten)

In Tabelle 6.4 sind die Messergebnisse der 22 Abfragen des TPC-H Benchmarks auf der 10 GB-Datenbank angegeben. Die Spalte mit der Bezeichnung „Laufzeit [s]“ gibt die Zeit in Sekunden an, welche benötigt wurde um den jeweiligen SQL-Befehl vollständig auszuführen. Die Spalte „Joule“ gibt die geleistete elektrische Arbeit für die vollständige Ausführung eines SQL-Befehls an. Die letzte Spalte „Joule/s“ gibt die durchschnittliche Leistungsaufnahme pro Sekunde in Watt an ($Leistung = \frac{Elektrische\ Arbeit}{Zeit} = \frac{Joule}{Laufzeit[s]}$). Die Laufzeit der Befehle wurde auf maximal eine Stunde begrenzt. Messungen, die diese Grenze überschritten, sind mit dem Wort „Timeout“ markiert. Weitere Details zu der Bedeutung der Spaltenbezeichnungen wurden bereits in Kapitel 5, Abschnitt 5.3.1 erläutert.

Diese Ergebnisse dienen als Referenzdaten für alle weiteren Energieoptimierungen in diesem Kapitel. Unser Ziel ist es, die benötigte Energie für die Ausführung der einzelnen Abfragen zu reduzieren, um so die Energieeffizienz der Datenbank Caché für den TPC-H Workload zu verbessern.

## 6.2 Energieeffizienz und Indizes

### 6.2.1 Einleitung

In diesem Kapitel liegt der Fokus auf der Anwendung von Indizes und die Auswirkungen auf den Energiebedarf für die Verarbeitung der Abfragen des TPC-H Benchmarks. Als Grundlage dient die in Abschnitt 6.1.1 vorgestellte Implementierung der Daten. Ziel ist es, durch das Eingrenzen des Workloads und darauf abgestimmte Indizes den Energiebedarf bei der Ausführung des TPC-H Benchmarks zu senken und darüber hinaus festzustellen, ob Methoden für das Anlegen von Indizes existieren, die die Abfragebearbeitung energieeffizienter machen. Der Workload sind die 22 TPC-H Abfragen.

| | 10 GB Datenbank (TPC-H Benchmark) | | |
|---|---|---|---|
| Abfrage | Laufzeit [s] | Joule | Joule/s |
| 1 | Timeout | N/A | N/A |
| 2 | 59,68 | 6555,91 | 109,85 |
| 3 | 1564,30 | 180717,68 | 115,53 |
| 4 | 247,79 | 30889,91 | 124,66 |
| 5 | 636,63 | 69149,17 | 108,62 |
| 6 | 479,52 | 64405,72 | 134,31 |
| 7 | 2648,42 | 315327,82 | 119,06 |
| 8 | 2653,85 | 308499,13 | 116,25 |
| 9 | Timeout | N/A | N/A |
| 10 | 836,51 | 87374,56 | 104,45 |
| 11 | 372,64 | 50824,42 | 136,39 |
| 12 | 555,65 | 74415,74 | 133,93 |
| 13 | 595,46 | 82157,33 | 137,97 |
| 14 | 1098,41 | 124741,63 | 113,57 |
| 15 | 1022,10 | 137017,96 | 134,05 |
| 16 | 98,69 | 12973,99 | 131,47 |
| 17 | Timeout | N/A | N/A |
| 18 | 2495,52 | 341635,91 | 136,90 |
| 19 | 2953,12 | 331833,36 | 112,37 |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.4: Messergebnisse der 22 Abfragen des TPC-H Benchmarks auf der 10 GB-Datenbank.

### 6.2.2 Indexstrukturen in Caché

Caché bietet wie andere Datenbanken die Möglichkeit, Indizes für die in der Datenbank enthaltenen Daten anzulegen. Diese benötigen zusätzlichen Speicherplatz und sind getrennt von den Daten des TPC-H abgespeichert. Caché bietet zwei Arten von Indizes an.

#### Konventioneller Index

Der Index bei Caché unterscheidet sich von der Funktionsweise nicht von dem einer relationalen Datenbank. Dieser enthält Tupel von Schlüsselattributen, und Referenzen auf die ID und stellt eine Ordnungsrelation für die Schlüsselattribute da. Eine einfache Ordnungsrelation ist das Sortieren nach einer Randbedingung, und kann z.B. bei Zahlenwerten auf- oder absteigend sein. Der Index kann bei Abfragen, die auf der referenzierten Spalte operieren, effizienter die abgefragten Datensätze bereitstellen, da durch die vorhandene Ordnungsrelation keine sequentielle Suche auf der Spalte erfolgen muss. Der erste Index in unserem Datenmodell ist jede Spalte, die einen Primärschlüssel enthält. Die Struktur, in der der Index abgespeichert, ist ein $B^+$-Baum [Wer07].

Ein zusätzlich angelegter Index benötigt immer zusätzlichen Speicherplatz. Caché speichert die Indizes wie die Daten in Globals, wobei der Inhalt des Globals für einen Index nur eine Referenz auf den indizierten Datensatz enthält. Bei einer Aktualisierung der Daten muss zusätzlich der Index aktualisiert werden, falls eine indizierte Spalte von der Aktualisierung betroffen ist. Dies bedeutet einen höheren Aufwand beim Ändern der Daten. Darum ist es von Vorteil, wenn die Anzahl der Indizes nicht hoch ist, um bei Aktualisierungen, unnötige Kosten zu vermeiden.

#### Bitmap-Index

Caché bietet neben dem konventionellen Index noch den Bitmap-Index zur Indizierung von Daten an. Dieser eignet sich für Spalten mit einer hohen Selektivität. Das sind die Spalten, bei denen ein konventioneller Index keine Steigerung der Effizienz bringt. Bei einem Bitmap-Index werden die indizierten Attribute einem Schlüssel in Form von einem Bitvektor zugeordnet. Der Vorteil ist, dass man bei einer multidimensionalen Suche nur Bitvektoren miteinander vergleichen muss. Tabelle 6.5 enthält Datensätze die eine hohe Selektivität haben (Geschlecht und Familienstand). Tabelle 6.6 zeigt den resultierenden Bitmap Index. Wenn nun eine Abfrage alle Elemente der Tabelle sucht die ledig und männlich sind ist die Suchmaske 10010. Die Abfrage kann mit einem Bitmap-Index schnell verarbeitet werden, da es sich bei der Abfrage im Kern um Binäroperationen handelt, die sehr effizient auf dem Prozessor ausgeführt werden können. Der Bitmap-Index benötigt durch seine Struktur wesentlich mehr Speicherplatz als ein konventioneller Index, und es ist daher aufwändiger ihn bei Veränderung der Daten zu aktualisieren. Aus diesen Gründen wird der Bitmap-Index häufig bei Datenbanken eingesetzt, bei denen überwiegend lesende Transkationen

| Name | Geschlecht | Familienstand |
|---|---|---|
| Max Mustermann | männlich | ledig |
| Birgit Muster | weiblich | verheiratet |
| Ludwig Musterer | männlich | geschieden |
| Arne Musterherr | männlich | ledig |

Tabelle 6.5: Beispieldaten für hohe Selektivität (Spalte „Geschlecht“ und „Familienstand“).

| Name | männlich | weiblich | verheiratet | ledig | geschieden |
|---|---|---|---|---|---|
| Birgit Muster | 0 | 1 | 1 | 0 | 0 |
| Ludwig Musterer | 1 | 0 | 0 | 0 | 1 |
| Max Mustermann | 1 | 0 | 0 | 1 | 0 |
| Arne Musterherr | 1 | 0 | 0 | 1 | 0 |

Tabelle 6.6: Bitmap für Tabelle hohe Selektivität.

ausgeführt werden. Wird ein Bitmap-Index eingesetzt, tauscht man höheren Speicherplatzbedarf gegen effizientere Berechnungen. Der Bitmap-Index wird, wie auch der konventionelle Index, in der Klassendefinition implementiert. Der folgende Code zeigt eine Bitmap-Definition.

```
Index Bitmap On GESCHLECHT [ Type = bitmap ];
```

### 6.2.3 Index Auswahl

Mit Indizes kann die Menge von Daten, die bei einer Abfrage von der Festplatte gelesen werden müssen, reduziert werden. Wir suchen in den 22 TPC-H Abfragen nach Bedingungen, die es uns ermöglichen, nicht benötigte Datensätze durch einen Index herauszufiltern. Wir können hierbei nur auf den konventionellen Index zurückgreifen, da der Bitmap-Index Rahmenbedingungen erfordert, die durch das TPC-H Schema nicht erfüllt sind. Zum einen ist der Bitmap-Index ist nur sinnvoll einsetzbar, wenn die Selektivität der zu indizierenden Spalte sehr hoch ist zum anderen beinhaltet der TPC-H Benchmark nur wenige Abfragen, die auf Spalten mit hoher Selektivität operieren. Alle diese Spalten gehören zur Tabelle *lineitem* die einen zusammengesetzten Primärschlüssel (vgl.6.1) haben. Laut Herstellerangabe ist es nicht möglich, auf Tabellen mit zusammengesetzten Primärschlüssel einen Bitmap-Index anzulegen.

Wir analysieren bei den Abfragen die WHERE-Bedingungen, da dies der Teil einer Abfrage ist, bei dem ein Index eine effizientere Bearbeitung ermöglicht. Diese Vorgehen wird von Algorithmen genutzt, die der Suche nach optimalen Indizes dienen [Küh09]. Zur Verdeutlichung dient Abfrage 12:

```
select
  l_shipmode,
```

```
  sum(case
    when o_orderpriority = '1-URGENT' or o_orderpriority = '2-HIGH'
      then 1 else 0
  end) as high_line_count,
  sum(case
    when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH'
      then 1 else 0
  end) as low_line_count
from
  orders, lineitem
where
  o_orderkey = l_orderkey
  and l_shipmode in ('MAIL', 'SHIP')
  and l_commitdate < l_receiptdate
  and l_shipdate < l_commitdate
  and l_receiptdate >= ToDate ('1994-01-01','YYYY-MM-DD')
  and l_receiptdate < DateAdd('y',1,'1994-01-01')
group by
  l_shipmode
order by
  l_shipmode
```

Wie zu sehen ist, operiert die Abfrage auf den zwei Tabellen *orders* und *lineitem*. Es handelt sich hierbei um die größten Tabellen unseres Schemas (vgl. Tabelle 6.3). In der WHERE-Bedingung werden verschiedenen Spalten der beiden Tabellen abgefragt. Da wir noch keinen zusätzlichen Index haben, müssen alle Elemente von *lineitem* nach den abgefragten Bedingungen untersucht werden. Mit Hilfe des Abfrageplans, den der Abfrageoptimierer bereitstellt, können wir sehen, wie die Verarbeitung der Abfrage durchgeführt würde. Abbildung 6.1 zeigt den Abfrageplan für Abfrage 12. Die Abfolge der Bearbeitung beginnt mit Modul B, das eine temporäre Datei C generiert.
Modul B beginnt mit einer Iteration über den Primärschlüssel der Tabelle *lineitem*.

```
Read master map SQLUser.LINEITEM.LINEITEMPRIMARYIDX,
looping on L_ORDERKEY and L_LINENUMBER.
```

In der Iteration werden für jede Zeile die Daten nach den Werten für o_orderkey und l_shipmode überprüft. o_orderkey ist der Primärschlüssel der Tabelle *orders*.

```
For each row:
Read master map SQLUser.ORDERS.ORDERSPRIMARYIDX,
using the given idkey value.
Check distinct values for %SQLUPPER(L_SHIPMODE) using temp-file B,
subscripted by a hashing of these values.
```

Der Abfrageoptimierer berechnet die Funktion aus der SELECT-Bedingung.

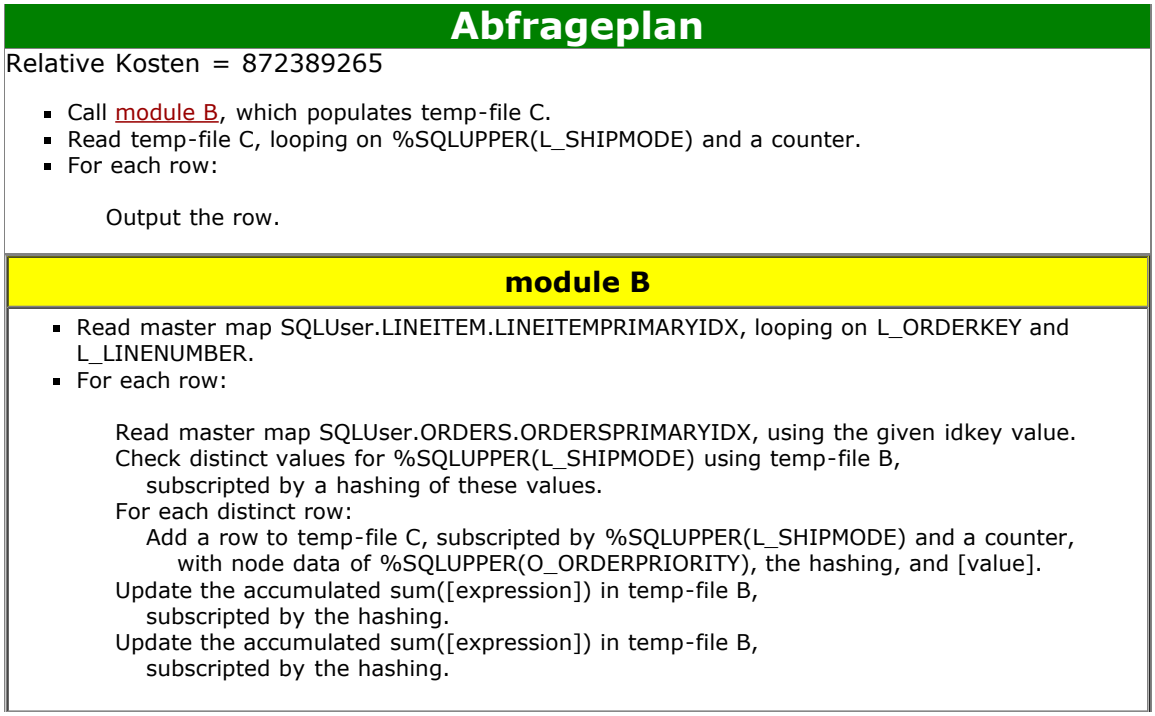



Abbildung 6.1: Abfrageplan TPC-H Abfrage 12 ohne zusätzlichen Index.

```
For each distinct row:
Add a row to temp-file C, subscripted by %SQLUPPER(L_SHIPMODE)
and a counter,
with node data of %SQLUPPER(O_ORDERPRIORITY),the hashing,and [value].
Update the accumulated sum([expression]) in temp-file B,
subscripted by the hashing.
Update the accumulated sum([expression]) in temp-file B,
subscripted by the hashing.
```

Als letzter Schritt der Abfrageausführung wird die group-by und order-by Bedingung berechnet.

```
Read temp-file C, looping on %SQLUPPER(L_SHIPMODE) and a counter.
For each row:
Output the row.
```

Es fällt sofort auf, dass der Abfrageplan nicht detailliert ist, denn die Überprüfung von l_commitdate, l_receiptdate und l_shipdate wird nicht aufgeführt. Es ist davon auszugehen, dass der Abfrageoptimierer intelligent genug ist, diese Überprüfung während der Iteration auf *lineitem* auszuführen, bevor er die weiteren Bedingungen überprüft.

Im Abfrageplan ist ersichtlich, dass alle Datensätze der Tabelle *lineitem* untersucht werden müssen, um das Ergebnis zu berechnen. Die Iteration am Beginn der Ausführung identifizieren wir als den teuersten Faktor bei der Ausführung.
Der Index bietet die Möglichkeit, Datensätze, die nicht zu den abgefragten Bedingungen passen, effizient aus der Berechnung auszuschließen. Die Spalte l_receiptdate eignet sich bei Abfrage 12 für einen zusätzlichen Index. Der Wertebereich ist durch die WHERE-Bedingung eingegrenzt

```
l_receiptdate >= ToDate ('1994-01-01','YYYY-MM-DD')
and l_receiptdate < DateAdd('y',1,'1994-01-01')
```

und die Spalte hat eine niedrige Selektivität von 0.04 %. Wir legen einen zusätzlichen Index an, der die Spalte l_receiptdate nach dem Datum indiziert. Der Abfrageoptimierer arbeitet mit Heuristiken, um die kostengünstigste Abfragevariante zu ermitteln. Nachdem der Index für l_receiptdate erstellt wurde, besteht die Möglichkeit, sich den Abfrageplan anzuzeigen, ohne dass die Abfrage ausgeführt werden muss. Die Abbildung 6.2 zeigt den Abfrageplan, nachdem der zusätzliche Index für die l_receiptdate angelegt wurde. Im Vergleich zum Abfrageplan ohne zusätzliche Indizes haben sich Veränderungen ergeben.
Der Abfrageoptimierer beginnt mit der Ausführung von Modul D, dieses Modul ruft Modul B auf, das eine temporäre Datei erzeugt. Modul B liest den Index *LRECIEPTDATEIDX*; dies ist der Index, den wir für l_receiptdate angelegt haben. Es wird eine Iteration über dem Index ausgeführt, die eine eingrenzende Auswahl beinhaltet.

```
Read index map SQLUser.LINEITEM.LRECIEPTDATEIDX,
looping on L_RECEIPTDATE (with a range condition)
, L_ORDERKEY, and L_LINENUMBER.
```

Jede Zeile, die der Bedingung genügt, wird in eine temporäre Datei A geschrieben, wobei die ID der Spalte gespeichert wird (in diesem Fall der Primärschlüssel des Datensatzes aus der Tabelle *lineitem*).

```
For each row:
                Add a row to temp-file A, subscripted by ID,
                with no node data.
```

Die temporäre Datei A wird an Modul B übergeben. Modul B ruft Modul C auf, das eine Iteration auf den gespeicherten IDs aus Datei A ausführt.

```
Read temp-file A, looping on ID.
```

In Modul C wird für jede ID ein Lesezugriff auf dem entsprechenden Datensatz mit dem jeweiligen Primärschlüssel der Tabelle *lineitem* ausgeführt.

```
For each row:
Read master map SQLUser.LINEITEM.LINEITEMPRIMARYIDX,
using the given idkey value.
```

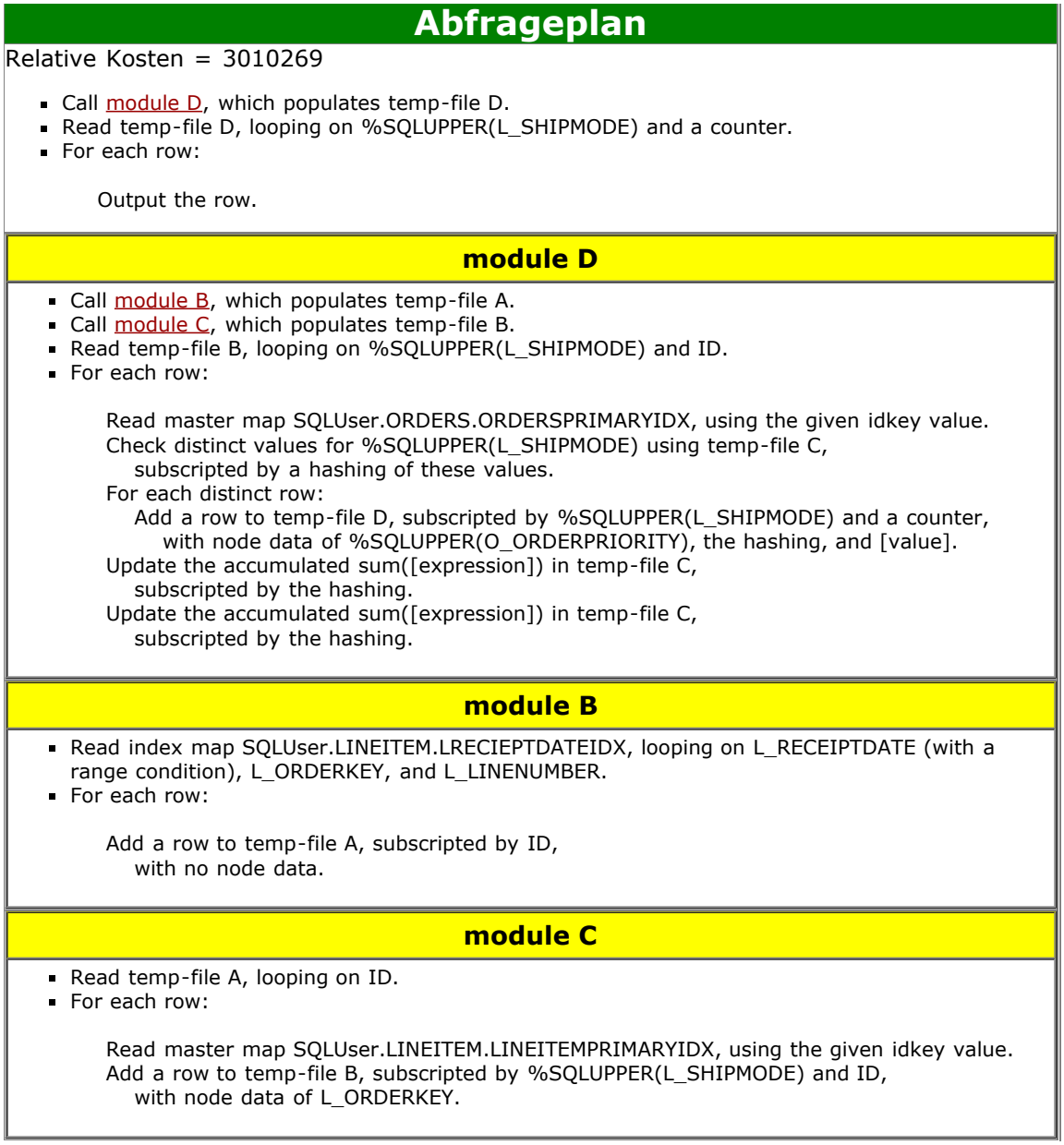



Abbildung 6.2: Abfrageplan Abfrage 12 mit zusätzlichem Index auf Spalte lreceiptdate.

Für jede ID wird eine Zeile in eine temporäre Datei B geschrieben, die nach l_shipmode und dem Primärschlüssel indiziert ist und die den Datensatz für l_orderkey beinhaltet.

```
Add a row to temp-file B, subscripted by %SQLUPPER(L_SHIPMODE) and ID,
with node data of L_ORDERKEY.
```

Die Datei B wird an Modul D übergeben, dass eine Iteration über l_shipmode und der ID ausführt.

```
Read temp-file B, looping on %SQLUPPER(L_SHIPMODE) and ID.
```

In der Iteration wird in jedem Schritt mit Hilfe des Primärschlüssels von *orders* überprüft, ob o_orderkey gleich l_orderkey ist.

```
Read master map SQLUser.ORDERS.ORDERSPRIMARYIDX,
using the given idkey value.
```

Die folgenden Operationen sind identisch zu denen, die im Abfrageplan 6.1 ohne zusätzlichen Index ausgeführt werden. Es ist ersichtlich, dass die Iteration über der Tabelle *lineitem* aus dem Abfrageplan ohne Index 6.1 nicht mehr vorhanden ist, falls ein Index auf l_receiptdate angelegt wird. Die Operation, die wir als kostenintensiv klassifiziert haben, wurde durch eine Iteration auf dem zusätzlich angelegten Index ersetzt.

Alle Abfragen wurden nach diesem Muster analysiert. Der Fokus lag auf WHERE-Bedingungen, die eingegrenzte Werte beinhalten. Spalten, die in dieses Muster passen, haben wir als mögliche Kandidaten für einen zusätzlichen Index angesehen. Diese Spalten wurden auf ihre Selektivität hin untersucht, bei Spalten mit niedriger Selektivität haben wir einen zusätzlichen Index für die Spalte angelegt.
Im Folgenden ist ein Auszug von eingrenzenden Bedingungen bei den TPC-H Abfragen:

```
l_shipdate <= DateAdd('d',-90,'1998-12-01')
p_size = 15
p_type like '\%BRASS'
o_orderdate < ToDate('1995-03-15','YYYY-MM-DD')
l_shipdate > ToDate('1995-03-15','YYYY-MM-DD')
o_orderdate >= ToDate('1993-07-01','YYYY-MM-DD')
o_orderdate < DateAdd('m',3,'1993-07-01')
o_orderdate >= ToDate('1994-01-01','YYYY-MM-DD')
o_orderdate < DateAdd('yyyy',1,'1994-01-01')
l_shipdate >= ToDate('1994-01-01','YYYY-MM-DD')
l_shipdate < DateAdd('yyyy',1,'1994-01-01')
l_shipdate between ToDate('1995-01-01','YYYY-MM-DD')
ToDate('1996-12-31','YYYY-MM-DD')
```

```
o_orderdate between ToDate('1995-01-01','YYYY-MM-DD')
ToDate('1996-12-31','YYYY-MM-DD')
p_type = 'ECONOMY ANODIZED STEEL')
```

Durch die Analyse der Abfragen, wie Anhand von Abfrage 12 erklärt, ergeben sich Indizes für den TPC-H Workload, die in Tabelle 6.7 dargestellt sind. Wir haben bei

| Tabelle des TPC-H Schemas | indiziert Spalte | Selektivität der indizierten Spalte |
|---|---|---|
| lineitem | l_shipdate | 0.0437% |
| lineitem | l_receiptdate | 0.0441% |
| lineitem | l_shipmode | 14.28% |
| part | p_type | 0.6667% |
| part | p_size | 2% |
| part | p_container | 2.5% |
| orders | o_orderdate | 0.0678% |
| nation | n_name | 4.0000% |

Tabelle 6.7: Tabellen, indizierte Spalten und deren Selektivität.

der Analyse der Abfragen zusätzlich darauf geachtet, dass die Indizes von möglichst vielen Abfragen benutzt werden können. Dabei wurde untersucht, ob es eine Schnittmenge bei den abgefragten Spalten gibt, die für einen Index in Betracht kommen. Wir haben für die Abfragen nur 8 zusätzliche Indizes angelegt um eine Steigerung der Energieeffizienz zu erreichen. Tabelle 6.8 zeigt, welche Index von welcher Abfrage benutzt wird. Bei den Abfragen wo keine Inhalte zu sehen sind, gab es keine Bedingung im WHERE-Abschnitt, die dem gesuchten Muster entspricht. Wir haben aus diesem Grund auf zusätzliche Indizes für diese Abfragen verzichtet.

### 6.2.4 TPC-H Benchmark mit Indizes

Der TPC-H Benchmark wurde mit Indizes ausgeführt. Dabei wurden alle 22 Abfragen hintereinander ausgeführt. Nach jeder Abfrage wurde die Datenbank neu gestartet, damit die folgenden Abfragen nicht von Daten, die zwischengespeichert wurden, profitieren können. In Tabelle 6.9 sind die Ergebnisse des Benchmarks festgehalten. Die Werte in Klammern geben die prozentuale Veränderung zu den Referenzdaten 6.4 an. Bei 13 Abfragen haben wir durch zusätzliche Indizes Energie eingespart. Sieben Abfragen wurde nach unserem festgelegten Timeout von einer Stunde abgebrochen. Dies ist ein Abbruch mehr als in den Referenzdaten 6.4. Abfrage 13 blieb unverändert, da für diese kein zusätzlicher Index angelegt wurde. Abfrage 16 hat eine höhere Laufzeit und einen höheren Energiebedarf als in den Referenzdaten.
Die Energieeinsparungen befinden sich in einem Bereich von 9 % bis 82 %.

#### Analyse der Abfragen mit niedrigerem Energiebedarf

Wir sehen an den Messergebnissen, dass durch Indizes Energieeinsparungen erreicht werden können. Wir haben für alle Abfragen Indizes nach gleichem Muster angelegt,

| Abfrage | Index auf Spalte |
|---|---|
| 1 | L_SHIPDATE |
| 2 | P_TYPE, P_SIZE |
| 3 | O_ORDERDATE, L_SHIPDATE |
| 4 | O_ORDERDATE |
| 5 | O_ORDERDATE |
| 6 | L_SHIPDATE |
| 7 | L_SHIPDATE, N_NAME |
| 8 | O_ORDERDATE, P_TYPE |
| 9 | - |
| 10 | O_ORDERDATE |
| 11 | N_NAME |
| 12 | L_RECIEPTDATE |
| 13 | - |
| 14 | L_SHIPDATE |
| 15 | L_SHIPDATE |
| 16 | P_SIZE |
| 17 | P_CONTAINER |
| 18 | O_ORDERDATE |
| 19 | P_SIZE, L_SHIPMODE |
| 20 | L_SHIPDATE, N_NAME |
| 21 | L_RECIEPTDATE, N_NAME |
| 22 | - |

Tabelle 6.8: Die 22 TPC-H Abfragen und indizierte Spalte.

| | 10 GB Datenbank (TPC-H Benchmark) mit zusätzlichen Indizes | | |
|---|---|---|---|
| Nr. | Laufzeit [s] | Joule | Joule/s |
| 1 | Timeout | N/A | N/A |
| 2 | 54,81 (-8,16 %) | 5350,89 (-18,38 %) | 97,63 (-11,13 %) |
| 3 | 1208 (-22,78 %) | 145477,7 (-19,50 %) | 120,43 (+4,24 %) |
| 4 | 203,51 (-17,87 %) | 22817,59 (-26,13 %) | 112,12 (-10,06 %) |
| 5 | 527,54 (-17,14 %) | 55848,21 (-19,24 %) | 105,86 (-2,53 %) |
| 6 | 333,52 (-30,45 %) | 42706,59 (-33,69 %) | 128,05 (-4,67 %) |
| 7 | 2353,35 (-11,14 %) | 286929,06 (-9,01 %) | 121,92 (+2,40 %) |
| 8 | 1238,23 (-53,34 %) | 135992,889 (-55,92 %) | 109,83 (-5,52) |
| 9 | Timeout | N/A | N/A |
| 10 | 592,63 (-29,16 %) | 60352,92 (-30,93 %) | 101,84 (-2,50 %) |
| 11 | 332,93 (-10,66 %) | 45866,10 (-9,76 %) | 137,77 (+1,01 %) |
| 12 | 137,96 (-75,17 %) | 12887,89 (-82,68 %) | 93,42 (-30,25 %) |
| 13 | 589,31 (-1,03 %) | 81339,62 (-1,00 %) | 138,02 (+0,04 %) |
| 14 | 487,69 (-61,93 %) | 47486,84 (-55,60 %) | 97,37 (-14,26 %) |
| 15 | 409,35 (-59,95 %) | 44015,13 (-67,88 %) | 107,53 (-19,79 %) |
| 16 | 288,87 (+192,72 %) | 29881,93 (+130,32 %) | 103,44 (-21,32 %) |
| 17 | Timeout | N/A | N/A |
| 18 | Timeout | N/A | N/A |
| 19 | 1886,20 (-36,13 %) | 205502,07 (-38,07 %) | 108,95 (-3,04 %) |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.9: Messergebnisse der 22 Abfragen des TPC-H Benchmarks auf der 10 GB-Datenbank mit zusätzlichen Indizes.

wie in Abschnitt 6.2.3 erklärt. Dennoch gibt es Unterschiede bei den Einsparungen. Dies kann einerseits an der Benutzung des Index liegen oder an der Komplexität der Abfrage. Wir haben uns beim Erstellen der Indizes auf einen Aspekt konzentriert, die Suche nach eingrenzenden Wertebereichen in der WHERE-Bedingung. Die Abfrage des TPC-H sind teilweise sehr vielschichtig und komplex. Wir wollen nun analysieren, wie die einzelnen Einsparungen zustande kommen.

Abfragen 6, 8 und 12 zeigen unterschiedliche Einsparungen der Energie bei der Ausführung mit Index. Wir wollen analysieren, ob es bei diesen Abfragen einen Zusammenhang zwischen der Anwendung des Index und dem Grad der Energieeinsparungen gibt. Hierbei helfen uns die Abfragepläne.
Abfrage 6 des TPC-H sieht wie folgt aus:

```
select
  sum(l_extendedprice * l_discount) as revenue
from
  lineitem
where
  l_shipdate >= ToDate('1994-01-01','YYYY-MM-DD')
  and l_shipdate < DateAdd('yyyy',1,'1994-01-01')
  and l_discount between .06 - 0.01 and .06 + 0.01
  and l_quantity < 24
```

Die Abfrage hat eine geringe Komplexität, da die Abfrage nur auf der Tabelle *lineitem* operiert. Die Abbildung 6.3 zeigt die Auslastung für die Ausführung ohne Index. In dieser Abbildung ist ersichtlich, dass der Prozessor über die gesamte Abfragebearbeitung hinweg nahezu zu 100 % ausgelastet ist. Die Leserate der Festplatte ist auf einem konstanten Niveau von 20 %, der Energieverbrauch liegt fast konstant bei 130 Watt.
Bei der Ausführung mit zusätzlichem Index benutzt die Abfrage den Index für die Spalte l_shipdate. Die Abbildung 6.4 zeigt die Auslastung bei der Ausführung mit zusätzlichem Index. Hier ist zu erkennen, dass die Bearbeitung von Abfrage 6 anders ausgeführt wird. Die CPU-Auslastung ist bis zur Sekunde 110 auf 100 %, die Leserate auf einem Niveau von unter 5 %. Danach sinkt die CPU Auslastung auf 50 % und die Leserate steigt stark an auf über 50 %. Sie schwankt zwar im weiteren Verlauf, bleibt jedoch über 35 %. Die Prozessorauslastung schwankt zwischen 50 % und 65 %. Der Energiebedarf beträgt in dieser Phase zwischen 110 und 120 Watt. Aus dem Abfrageplan leiten wir die Phasen, die sich in den Auslastungsgrafiken zeigen, ab. Abbildung 6.5 zeigt den Abfrageplan für Abfrage 6 ohne zusätzlichen Index. Dieser zeigt, was wir in Abbildung 6.4 für die Ausführung ohne Index erkennen. Der Abfrageplan besteht nur aus einer Phase; diese führt eine Iteration über alle Elemente der Tabelle *lineitem* aus.

```
Read master map SQLUser.LINEITEM.LINEITEMPRIMARYIDX,
looping on L_ORDERKEY and L_LINENUMBER.
```

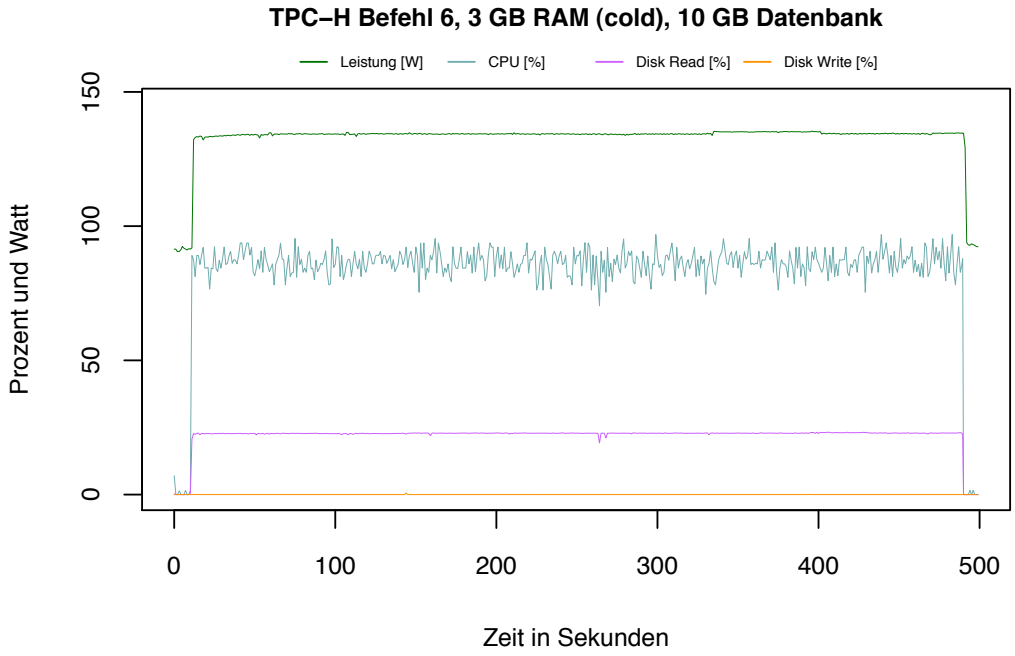



Abbildung 6.3: TPC-H Befehl 6, 3 GB RAM (cold ), 10GB Datenbank.

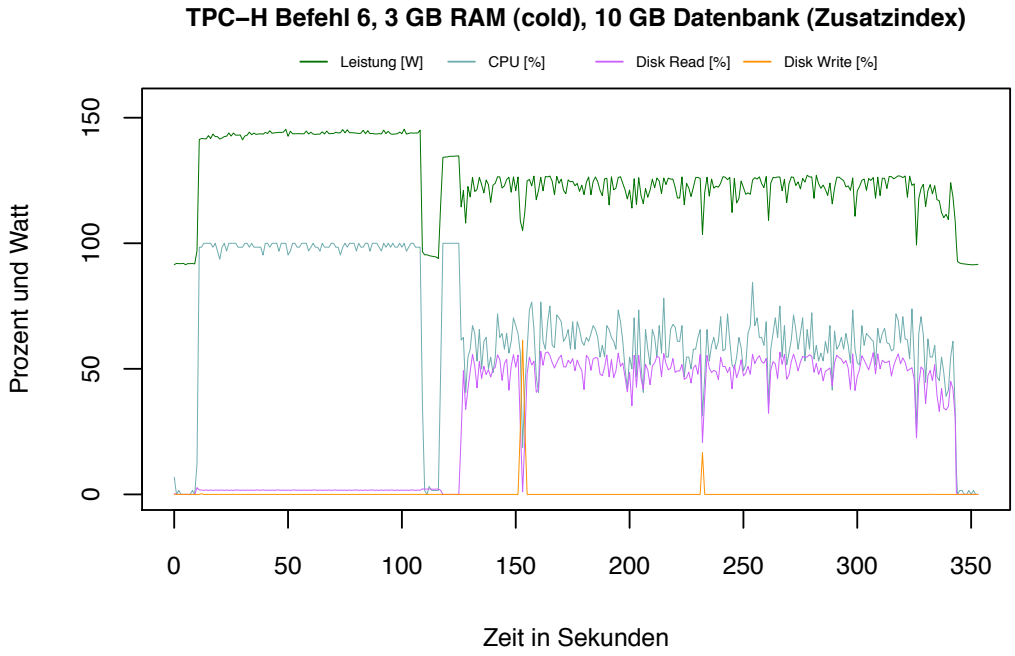



Abbildung 6.4: TPC-H Befehl 6, 3 GB RAM (cold ), 10GB Datenbank mit zusätzlichem Index.

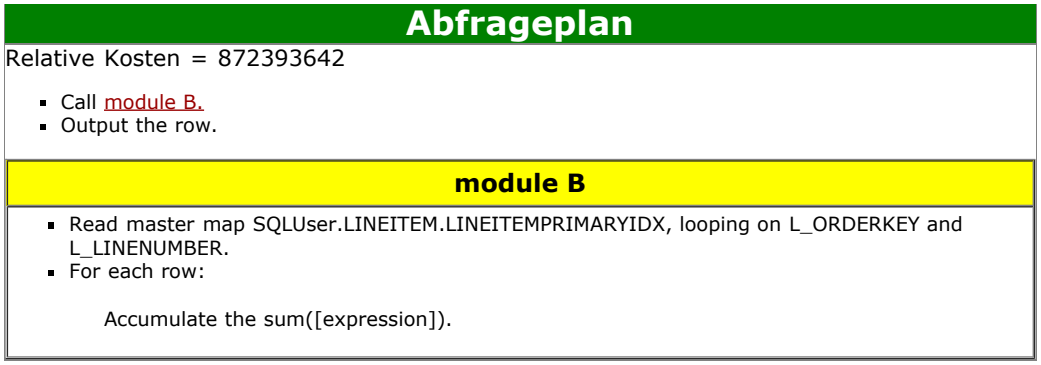



Abbildung 6.5: Abfrageplan der TPC-H Abfrage 6 ohne zusätzlichen Index.

Während dieser werden die abgefragten Bedingungen für die Spalten l_shipdate, l_discount und l_quantity und aus der Tabelle *lineitem* überprüft. Diese Details werden im Abfrageplan nicht angezeigt. Für jede Iteration wird die angefragte Summe aus der Select-Bedingung berechnet.

```
sum(l_extendedprice * l_discount) as revenue
```

korrespondiert zu dem Folgenden Teil des Abfrageplans:

```
For each row:
                          Accumulate the sum([expression]).
```

Diese Art der Ausführung erzeugt eine hohe CPU-Last und damit einen hohen Energieverbrauch über die ganze Abfrage hinweg.

Der Abfrageplan für die Ausführung mit zusätzlichem Index ist in Abbildung 6.6 dargestellt. Im Unterschied zum Abfrageplan 6.5 besteht der Abfrageplan 6.6 aus zwei Modulen. Die Ausführung beginnt mit Modul C welches Modul B aufruft. Modul B führt eine Iteration mit eingrenzender Auswahl über dem Index der Spalte l_shipdate aus.

```
Read index map SQLUser.LINEITEM.LSHIPDATEIDX,
looping on L_SHIPDATE (with a range condition),
L_ORDERKEY, and L_LINENUMBER.
```

Dies entspricht dem folgenden WHERE-Teil der Abfrage:

```
l_shipdate >= ToDate('1994-01-01','YYYY-MM-DD')
and l_shipdate < DateAdd('yyyy',1,'1994-01-01')
```

Modul B generiert eine temporäre Datei A, indiziert nach der ID für die Datensätze, die zu der abgefragten Bedingung passen.

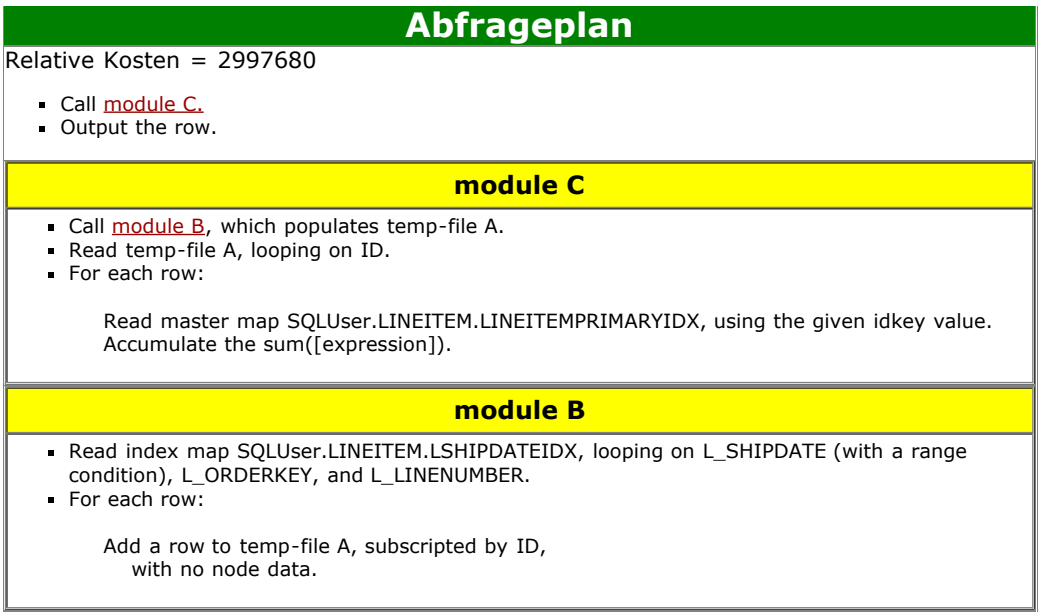



Abbildung 6.6: Abfrageplan Abfrage 6 mit zusätzlichem Index.

```
For each row:
Add a row to temp-file A, subscripted by ID,
with no node data.
```

Diese temporäre Datei wird in Modul C für die Iteration über dem Primärschlüssel genutzt.

```
Read temp-file A, looping on ID.
```

In dieser Iteration werden die weiteren Berechnungen der Abfrage ausgeführt.

```
For each row:
Read master map SQLUser.LINEITEM.LINEITEMPRIMARYIDX,
using the given idkey value.
```

Dies korrespondiert zu der folgenden SQL Befehlen aus Abfrage 6:

```
l_discount between .06 - 0.01 and .06 + 0.01
l_quantity < 24
```

Am Ende der Abfrage wird die Berechnung aus der Select-Bedingung durchgeführt.

```
Accumulate the sum([expression]).
```

Die Phase bis 110 Sekunden in Abbildung 6.4 entspricht Modul C in Abfrageplan 6.6. Die Phase ab 110 Sekunden entspricht Modul B. Es ist aus der Auslastungsgrafik ersichtlich, dass die Auswertung der WHERE-Bedingung anhand des Index und das Schreiben der temporären Datei sehr CPU-intensiv ist. Die Verarbeitung von Modul C hat eine niedrige CPU-Intensität und eine hohe Auslastung der Festplatte zufolge. Für den Energiebedarf ist dies ein Optimum, da die CPU im Vergleich zur Festplatte ein Vielfaches an Strom verbraucht. Wir sehen an Abfrage 6 mit Index, das die Nutzung des selbigen Folgen für die Auslastung des Gesamtsystems und damit für den Energieverbrauch hat.

Es gibt dafür zwei Gründe. Der Erste ist, das durch den Index effizient Datensätze von der Berechnung ausgeschlossen werden können. Die weitere Bearbeitung der Abfrage erfolgt auf einer Untermenge von Datensätzen aus der Tabelle *lineitem*. Somit kann die Überprüfung der Datensätze schneller abgeschlossen werden. Dies hat Auswirkungen auf die Bearbeitungszeit. Der zweite Grund ist die Reduzierung der Berechnungen. Nach der Nutzung des Index muss für die Bearbeitung der Abfrage eine Bedingung weniger ausgewertet werden, wodurch die die Gesamtzahl der Berechnungen reduziert wird. Daraus folgt ein Sinken der CPU-Auslastung und ein Anstieg der Leserate, wie in Abbildung 6.4 zu sehen ist.

Die Energieersparnis ergibt sich hier durch Reduzierung der Berechnungen bei gleichzeitiger Reduzierung der Datensätze.Wir führen den folgenden SQL Befehl aus:

```
SELECT count(l_shipdate) FROM lineitem WHERE
l_shipdate >= ToDate('1994-01-01','YYYY-MM-DD')
and l_shipdate < DateAdd('yyyy',1,'1994-01-01')
```

Das Ergebnis liefert 9.099.165 Datensätze zurück. Somit fallen 15 % der Datensätze der Tabelle *lineitem* in den abgefragten Wertebereich. Das heißt ca. 85 % der Datensätze wurden aus der Berechnung ausgeschlossen.

TPC-H Abfrage 8 hat einen komplexeren Aufbau :

```
select
  o_year,
  sum(case
    when nation = 'BRAZIL' then volume
    else 0
  end) / sum(volume) as mkt_share
from (
  select
    DatePart('yyyy',o_orderdate) as o_year,
    l_extendedprice * (1 - l_discount) as volume,
    n2.n_name as nation
  from
```

```
    part, supplier, lineitem, orders,
    customer, nation n1, nation n2, region
  where
    p_partkey = l_partkey
    and s_suppkey = l_suppkey
    and l_orderkey = o_orderkey
    and o_custkey = c_custkey
    and c_nationkey = n1.n_nationkey
    and n1.n_regionkey = r_regionkey
    and r_name = 'AMERICA'
    and s_nationkey = n2.n_nationkey
    and o_orderdate between ToDate('1995-01-01','YYYY-MM-DD')
      and ToDate('1996-12-31','YYYY-MM-DD')
    and p_type = 'ECONOMY ANODIZED STEEL') as all_nations
group by
  o_year
order by
  o_year
```

und operiert auf 7 Tabellen des TPC-H Schemas. Bei der Ausführung ohne zusätzlichen Index wird die Bearbeitung nach Abfrageplan 6.7 ausgeführt. Es ist ersichtlich, dass die Abfrage mit einer Iteration über der Tabelle *lineitem* beginnt.

```
Read master map SQLUser.LINEITEM.LINEITEMPRIMARYIDX,
looping on L_ORDERKEY and L_LINENUMBER.
```

Es wird für jede Iteration auf die anderen abgefragten Tabellen zugegriffen, um die gegebenen Bedingungen zu überprüfen.

```
For each row:
Read master map SQLUser.PART.PARTPRIMARYIDX,
 using the given idkey value.
Read master map SQLUser.SUPPLIER.SUPPLIERPRIMARYIDX,
 using the given idkey value.
Read master map SQLUser.NATION.NATIONPRIMARYIDX,
 using the given idkey value.
Read master map SQLUser.ORDERS.ORDERSPRIMARYIDX,
 using the given idkey value.
Read master map SQLUser.CUSTOMER.CUSTORMERPRIMARYIDX,
 using the given idkey value.
Read master map SQLUser.NATION.NATIONPRIMARYIDX,
 using the given idkey value.
Read master map SQLUser.REGION.REGIONPRIMARYIDX,
 using the given idkey value.
```

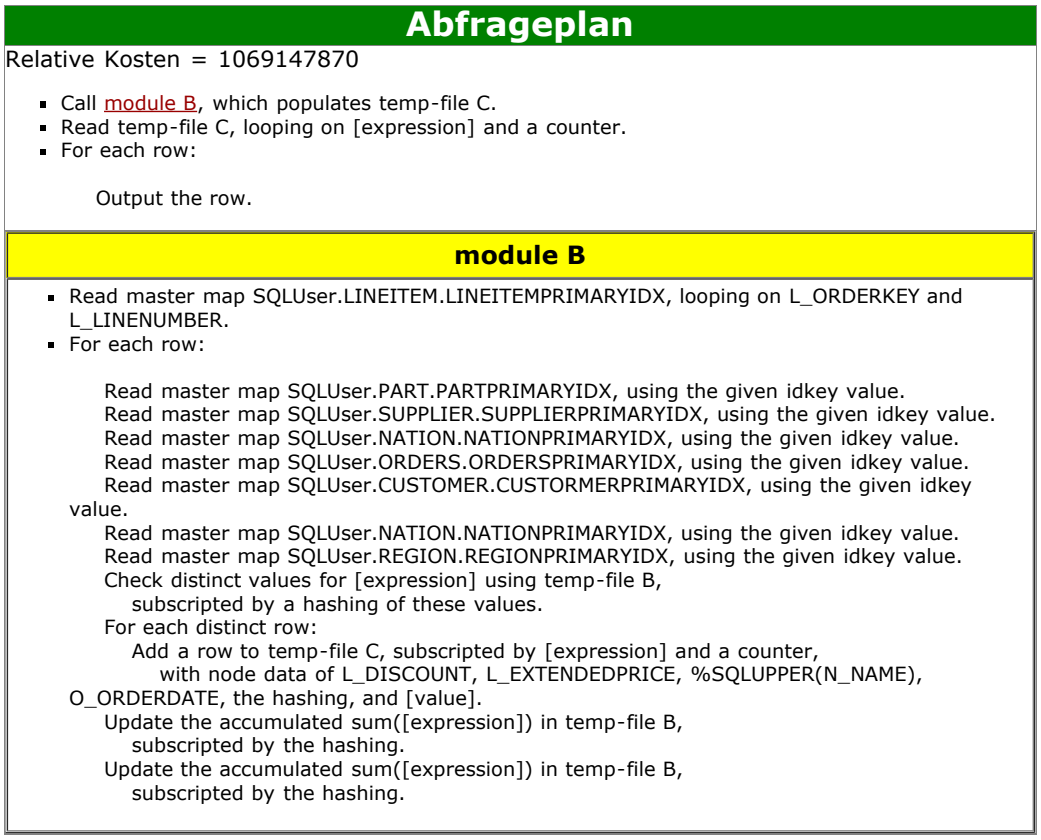



Abbildung 6.7: Abfrageplan TPC-H Abfrage 8 ohne zusätzlichen Index.

Der restliche Teil des Abfrageplans spiegelt den oberen Select-Teil wieder und ist für uns an dieser Stelle nicht von Relevanz. Wie aus Tabelle 6.8 ersichtlich, benutzt Abfrage 8 den Index für die Spalte o_orderdate und p_type.
Abbildung 6.8 zeigt den Abfrageplan für die Ausführung mit zusätzlichem Index. In

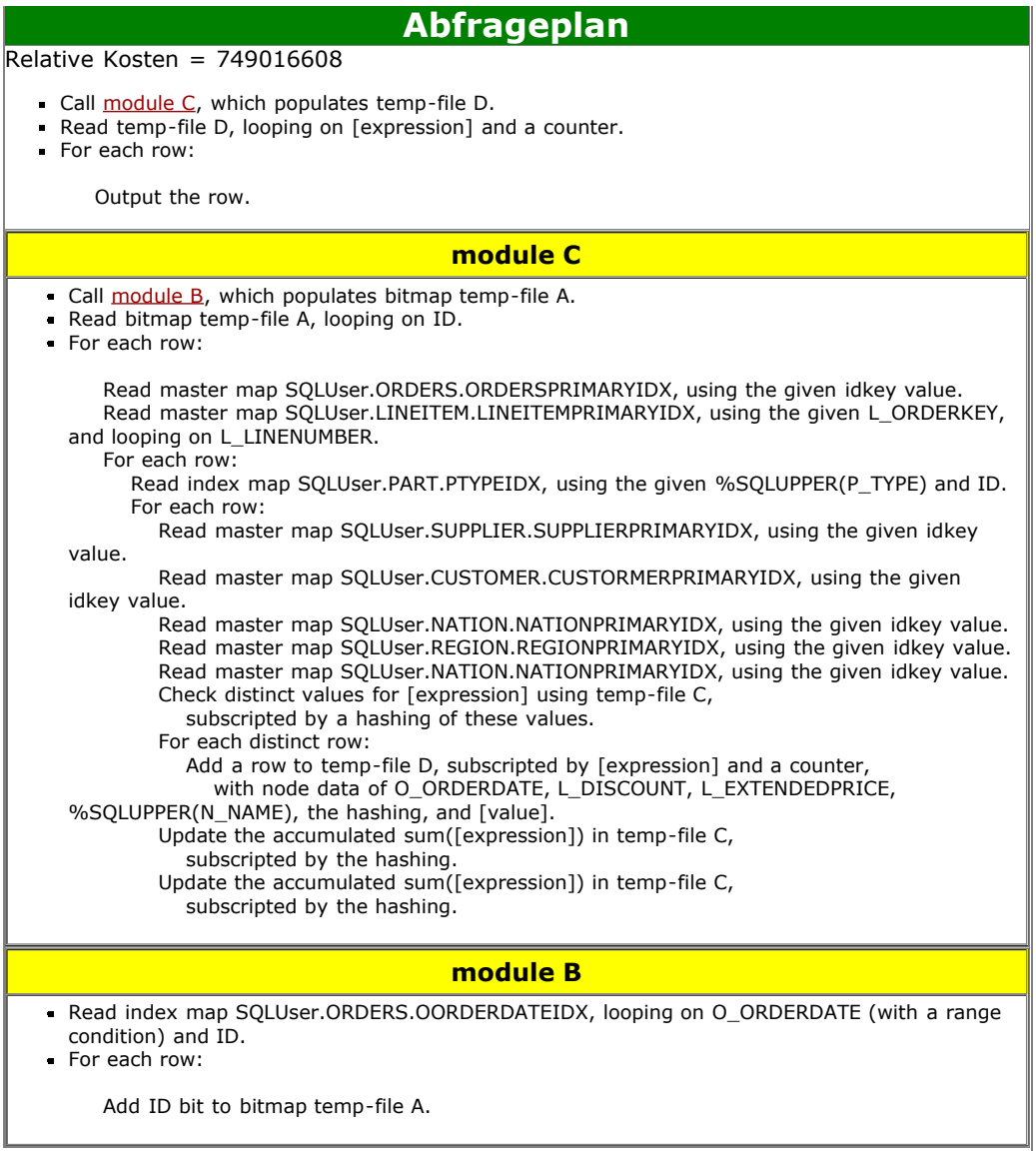



Abbildung 6.8: Abfrageplan TPC-H Abfrage 8 mit zusätzlichem Index.

Veränderung zu Abfrageplan 6.7 werden zwei Module abgearbeitet.
Modul C ruft Modul B auf, welches eine temporäre Datei A erzeugt.

```
Call module B, which populates bitmap temp-file A.
```

Modul B führt eine Iteration auf dem Index über der Spalte o_orderdate aus, diese Iteration hat eine eingrenzende Auswahl, das abgefragte o_orderdate aus dem WHERE-Teil der Abfrage.

```
Read index map SQLUser.ORDERS.OORDERDATEIDX, looping on O_ORDERDATE
(with a range condition) and ID.
```

Die temporäre Datei A wird nach ihrer Erstellung für die Iteration in Modul C benutzt

```
Read bitmap temp-file A, looping on ID.
```

*(das Bitmap bezieht sich nicht auf einen Bitmap-Index, sondern nur auf eine Speicherstrategie für die temporäre Datei)* Die weiteren Schritte des Abfrageplans dienen der Überprüfung und Berechnung der weiteren Bedingungen der Abfrage. Für die Optimierung ist an dieser Stelle von Relevanz, dass die Iteration über die Tabelle *lineitem* aus Abfrageplan 6.7 nicht mehr für die Berechnung notwendig ist.
Abbildung 6.9 zeigt die Auslastung und den Energieverbrauch von Abfrage 8 ohne zusätzlichen Index. Auffällig ist, dass in den ersten 400 Sekunden die CPU-Auslastung

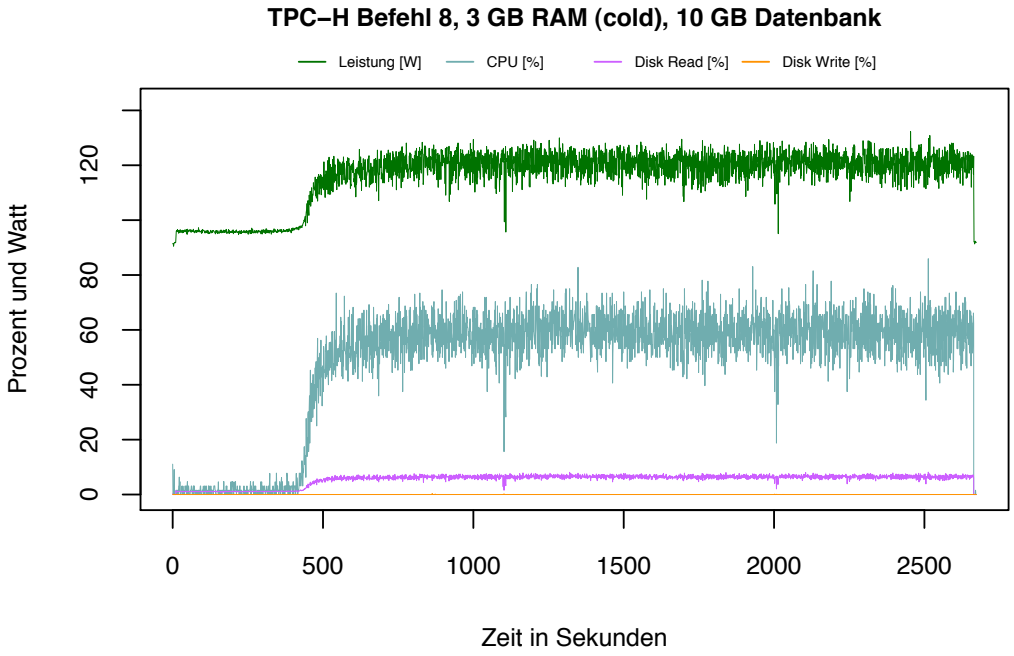



Abbildung 6.9: TPC-H Befehl 8, 3 GB RAM (cold), 10GB Datenbank.

und die I/O-Aktivität sehr niedrig sind. In dieser Phase ist der Energieverbrauch des Gesamtsystems nur minimal über dem Wert des Leerlaufs. Ab 400 Sekunden steigt die CPU-Auslastung deutlich an, und schwankt im weiteren Verlauf der Abfrage zwischen 50 % und 70 %, die Leserate der Festplatte steigt nur minimal an und bleibt bis zum Ende der Abfrage auf niedrigem Niveau. Mit dem Anstieg der CPU-Auslastung steigt der Energiebedarf auf 120 Watt und verbleibt auf diesem um diesen Wert bis die Verarbeitung der Abfrage beendet ist. Die niedrige Leserate der Festplatte ist mit

der Verarbeitung von mehreren Tabellen zu begründen. Da wir nicht wie in Abfrage 6 auf einer Tabelle Operieren ‚sondern mehrere Tabellen lesen müssen. Dies hat für die Festplatte zu folge das ein Lesen in wechselnden, nicht zusammenhängenden Speicherbereichen nötig ist. Für die Festplatte ist es bei dieser Abfrage nicht möglich die Daten sequentiell auszulesen.
Abbildung 6.10 zeigt die Auslastung während der Verarbeitung mit zusätzlichem Index. Im Unterschied zu Abbildung 6.3 ist zu erkennen, dass die Anfangsphase der

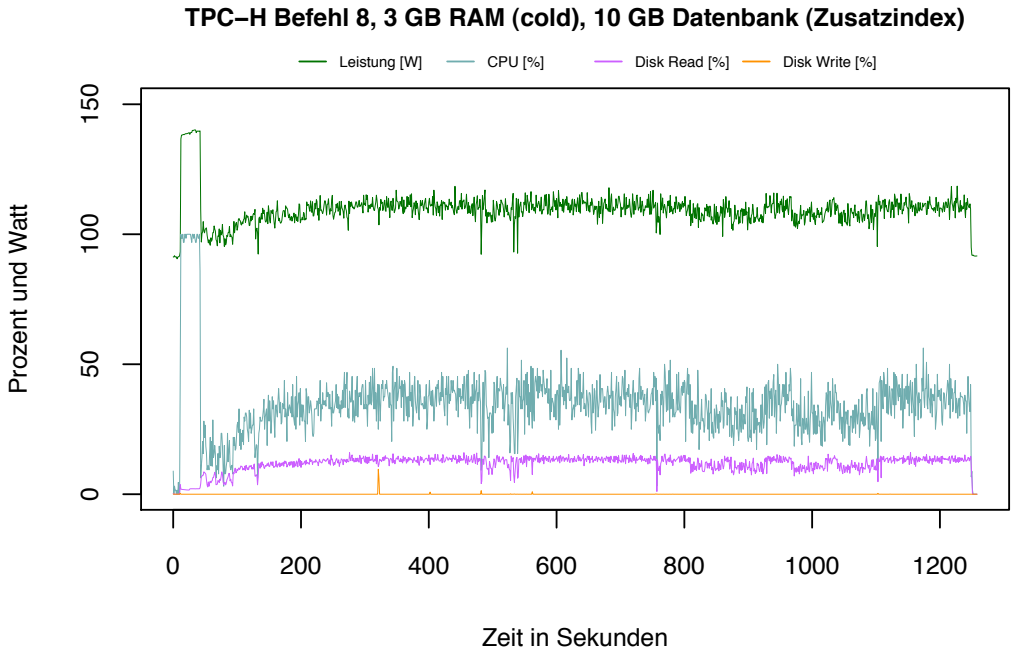



Abbildung 6.10: TPC-H Befehl 8, 3 GB RAM (cold ), 10GB Datenbank mit zusätzlichem Index.

Ausführung von einer hohen Prozessor Auslastung dominiert wird, die CPU ist hierbei zu 100 % ausgelastet, und die I/O Aktivität ist sehr niedrig. Der Energieverbrauch ist in dieser Phase bei 140 Watt was dem Maximum des Energieverbrauchs des Gesamtsystems entspricht. Ab der 45 Sekunde sinkt die CPU Auslastung auf 25 % ab und die Leserate der Festplatte steigt an; dies führt zu einer Reduzierung des Energiebedarfs auf ca. 110 Watt. Dieser Zustand bleibt bis zur Beendigung der Bearbeitung erhalten, schwankt allerdings deutlich.
Im Unterschied zu der Auslastung ohne Index ist die Anfangsphase nicht von niedriger CPU-Aktivität geprägt sondern von einer sehr hohen. Analog zu Abfrage 6 die Phase in der die Abfrage auf dem Index operiert und die entsprechenden Werte in eine temporäre Datei übertragen werden.
Die Auslastung für Abfrage 6 und 8 korrelieren in Bezug auf die Charakteristik der Verarbeitung. Auf beiden Abbildungen 6.4 6.10 ist zu erkennen, dass die Phasen, in denen der Index benutzt wird, von hoher CPU Auslastung und einem hohen Energiebedarf dominiert werden. Jedoch folgt in beiden Fällen eine zweite Phase mit

reduzierter CPU Aktivität und erhöhter Leserate. Dies ist damit zu erklären, dass hier wie bei Abfrage 6 nach der Nutzung des Index weniger Datensätze für die Berechnung verbleiben. Die Festplatte muss seltener die Speicherbereiche wechseln, und kann somit eine höhere Leserate erreichen.
Der Unterschied in der Länge der Anfangsphase mit 100 % CPU Auslastung liegt an der Anzahl der Datensätze, die aus dem Index geladen werden und dem Index selbst. Die Analyse wie, viele Datensätze in dem abgefragten Wertebereich von Abfrage 8 fallen, führen wir nach dem Muster wie bei Abfrage 6 aus. Der folgenden SQL Befehl gibt an, wie viele Datensätze zu der Bedingung

```
o_orderdate between ToDate('1995-01-01','YYYY-MM-DD')
and ToDate('1996-12-31','YYYY-MM-DD')
```

passen.

```
SELECT count(o_orderdate) FROM orders WHERE
o_orderdate between ToDate('1995-01-01','YYYY-MM-DD')
and ToDate('1996-12-31','YYYY-MM-DD')
```

Das Ergebnis der Abfrage sind 4.557.513 Datensätze. Das sind im Vergleich zu Abfrage 6 (9.099.165) ungefähr 50 % weniger. Daher dauert die Phase mit hoher CPU Auslastung bei Abfrage 8 nur 45 Sekunden, bei Abfrage 6 jedoch 110 Sekunden. Die Zeit hat sich nicht genau halbiert; das liegt an der Größe des Index. Der von Spalte o_orderdate ist kleiner als der von Spalte l_shipdate, dies ist gut an der Verteilung der Daten auf die einzelnen Tabellen 6.3 zu sehen.

Wir sehen uns nun die Abfrage 12 an, die mit zusätzlichem Index die größten Energieeinsparungen erzielt hat. Anhand derer wir in Abschnitt 6.2.3 erklärt haben, welche Spalten wir zusätzlich indexieren möchten, um die Energieeffizienz zu steigern.
Abbildung 6.11 zeigt die Auslastung des Systems bei der Ausführung ohne zusätzlichen Index. Die Abbildung zeigt, dass bis auf wenige Sekunden am Beginn der Ausführung die CPU Auslastung über die ganze Bearbeitung der Abfrage auf einem Niveau von über 80 % liegt. Die Leserate der Festplatte liegt bei konstant 25 % was höher ist als in Abfrage 8, da hier nur 2 Tabellen verarbeitet werden müssen. Der Energieverbrauch hat ein Niveau von über 130 Watt. Wie im Abschnitt 6.2.3 erklärt wird ist bei der Bearbeitung der Abfrage ohne zusätzlichen Index eine Iteration über die ganze Tabelle *lineitem* notwendig. Daraus ergibt sich das Auslastungsverhalten des Systems. Abbildung 6.12 zeigt die Auslastung während der Ausführung mit zusätzlichem Index. Es ist ersichtlich, dass die CPU-Auslastung am Beginn der Verarbeitung kurzzeitig auf 100 % ansteigt. Diese Verhalten bei der Nutzung eines zusätzlichen Index ist bei Abfrage 6 und 8 so zu beobachten(vgl.: 6.10 und 6.4). Jedoch ist diese Phase bei Abfrage 12 sehr kurz. Die restliche Bearbeitung der Abfrage ist geprägt von einer minimalen CPU Auslastung und einer minimalen Leserate der Festplatte. Daraus resultiert ein durchschnittlicher Energieverbrauch von 93 Watt, welcher nur minimal über dem Niveau des Leerlaufs liegt. Der Unterschied zu Abfrage 6 und 8

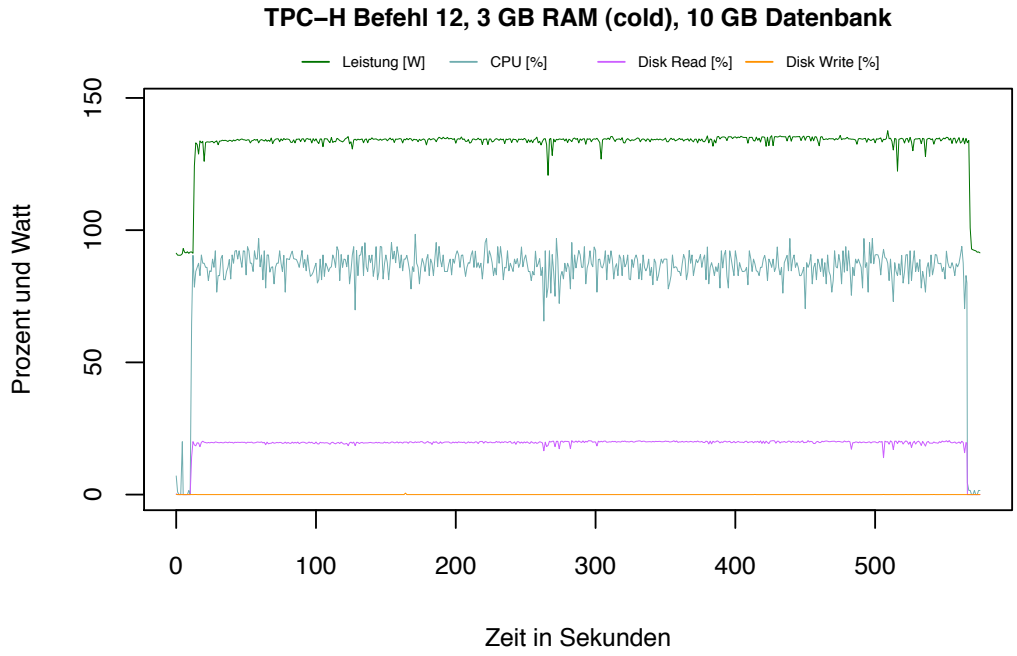



Abbildung 6.11: TPC-H Befehl 8, 3 GB RAM (cold ), 10GB Datenbank.

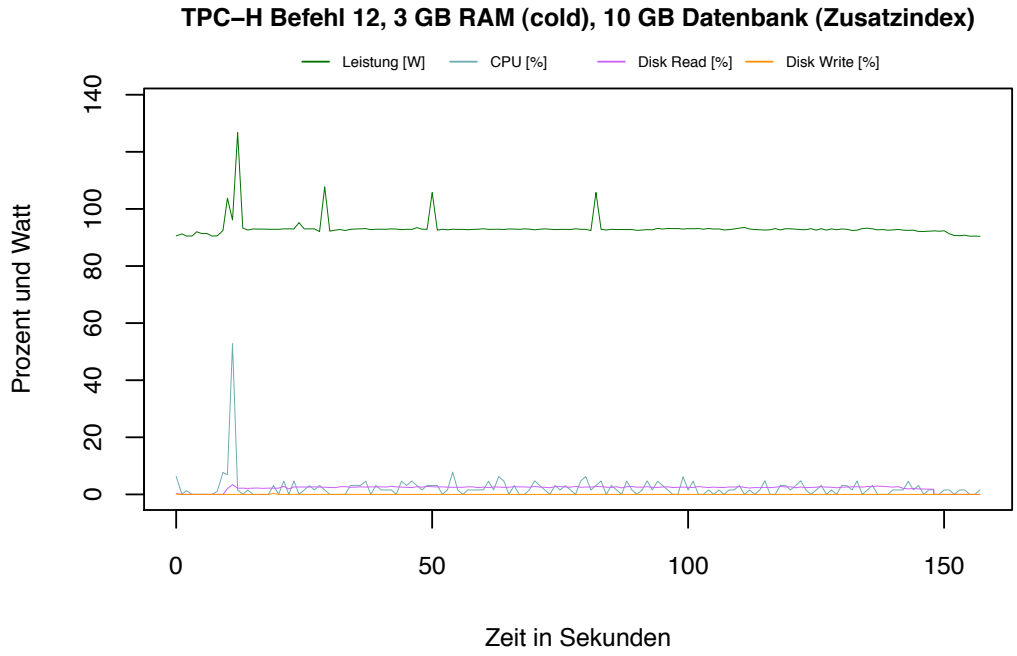



Abbildung 6.12: TPC-H Befehl 8, 3 GB RAM (cold ), 10GB Datenbank mit zusätzlichem Index.

liegt in der Absenkung der CPU-Auslastung und der Leserate der Festplatte, während bei Abfrage 6 und 8 die CPU Auslastung auf ein Niveau von 30% gefallen ist, fällt die CPU Auslastung bei Abfrage 12 mit Index auf nahezu 0 %. Die Leserate bleibt im Vergleich zu Abfrage 6 & 8 (vgl.:6.10 und 6.4) auch auf niedrigem Niveau, während sie in den anderen Beispielen ansteigt.
Das abweichende Verhalten bei Abfrage 12 ist mit dem Index zu erklären.
Wir überprüfen wie viele Datensätze in den abgefragten Wertebereich

```
l_receiptdate >= ToDate ('1994-01-01','YYYY-MM-DD')
and l_receiptdate < DateAdd('y',1,'1994-01-01')
```

fallen. Wir berechnen dies mit der SQL Abfrage:

```
SELECT count(l_receiptdate) FROM lineitem WHERE
l_receiptdate >= ToDate ('1994-01-01','YYYY-MM-DD')
and l_receiptdate < DateAdd('y',1,'1994-01-01')
```

Das Ergebnis ist 24803 Datensätze. Dies ist im Vergleich zu Abfrage 6 und 8 nur ein Bruchteil an Datensätzen, die für die weitere Bearbeitung in Frage kommen. Da durch die Nutzung des Index 99 % der Datensätze von der Berechnung ausgeschlossen werden ist die Energieersparnis entsprechend hoch.
Da nur wenige Datensätze verarbeitet werden müssen, und diese nicht zusammenhängend auf der Festplatte gespeichert sind, kann die Festplatte nicht sequentiell lesen, sondern muss von einem Speicherbereich in den nächsten wechseln. Dadurch ist die Leserate der HDD so niedrig.
Bei der Analyse der Abfragen 6, 8 und 12 fällt auf, dass die Anzahl der Werte, die nach der Auswertung des Index zur Berechnung verbleiben mit der Energieersparnis der Abfragen korrelieren.

Die Tabellen 6.10 gibt an, wie viele Elemente nach Auswertung des Index noch zu verarbeiten sind. Die Anzahl der Elemente ist in Spalte „Anzahl Datensätze“ aufgeführt. Die Spalte „Joule Ersparnis“ gibt an, wie viel Prozent an Energie im Vergleich zu den Referenzdaten 6.4 eingespart wurde. Die Spalte „Tabelle“ gibt an, auf welcher Tabelle des Schemas ohne zusätzlichen Index eine Iteration über alle Elemente ausgeführt wurde. Die Tabelle ist nicht vollständig absteigend sortiert, da die Komplexität der Abfrage bei dem Grad der Einsparung eine Rolle spielt. Jedoch ist zu erkennen, das je mehr Datensätze von der Berechnung ausgeschlossen werden, die Energieeinsparungen umso größer ist.

Zusammenfassend ist zu sagen, dass das Ausschließen von Datensätzen aus der Berechnung zu einer Steigerung der Energieeffizienz führt. Der Grad der Energieeinsparung hängt von der Anzahl der ausgeschlossenen Datensätze ab.

### Höherer Energiebedarf aufgrund von Indizes

Bei Abfrage 16 und 18 kam es durch die Nutzung eines zusätzlichen Index zu einer höheren Laufzeit und einem höheren Energiebedarf. Abfrage 18 wurde durch einen

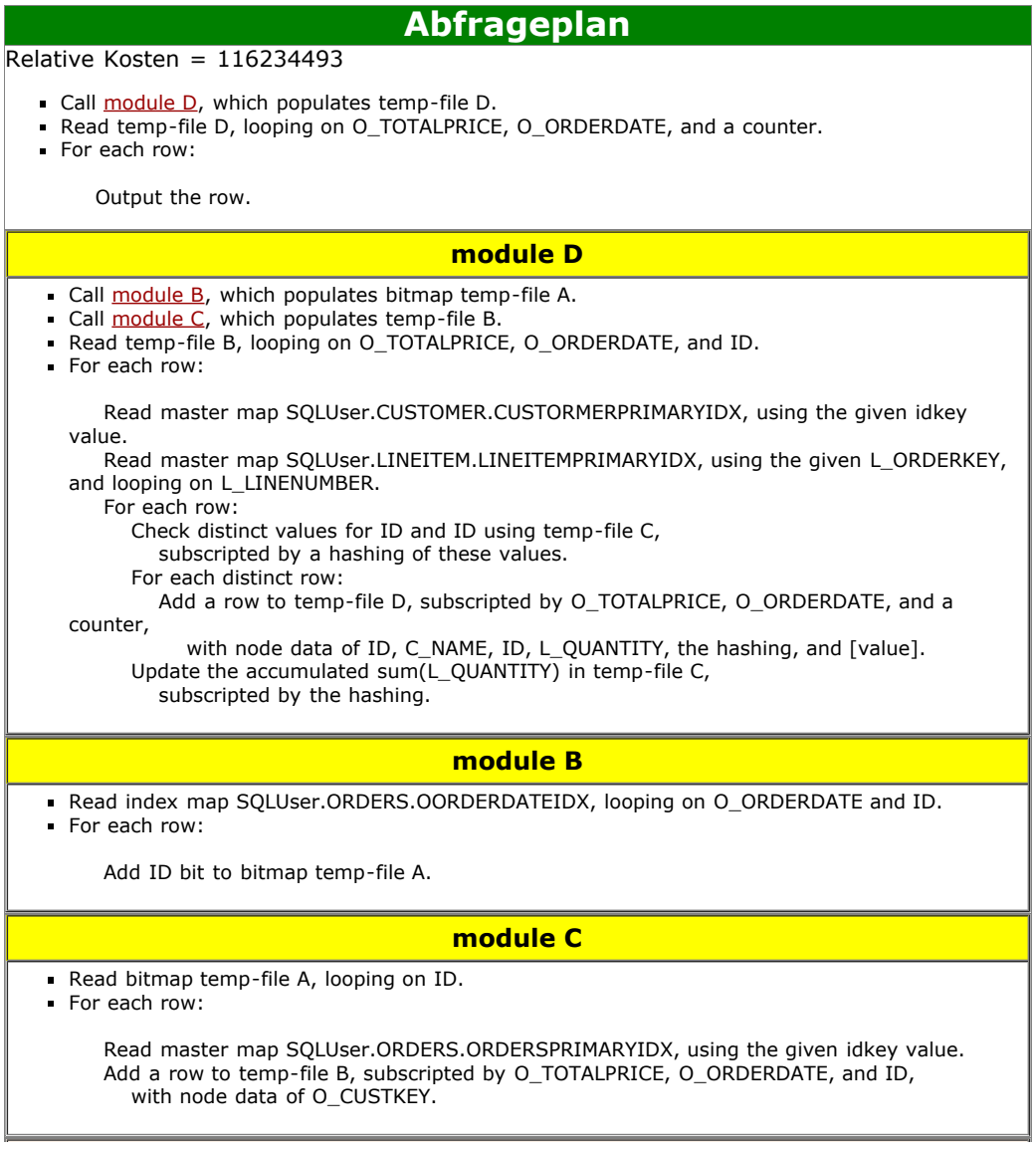
| Abfrage | Tabelle | Anzahl Datensätze | Joule Ersparnis [%] |
|---|---|---|---|
| 12 | Lineitem | 24.803 | -82,68 |
| 15 | Lineitem | 771.875 | -67,88 |
| 8 | Lineitem | 4.557.513 | -55,92 |
| 14 | Lineitem | 749.223 | -55,60 |
| 6 | Lineitem | 9.099.165 | -33,69 |
| 19 | Lineitem | 8.566.164 | -30,26 |
| 3 | Lineitem | 7.289.442 | -19,50 |
| 7 | Lineitem | 18.230.325 | -9,01 |
| 10 | Orders | 573.157 | -30,93 |
| 4 | Orders | 573.671 | -26,13 |
| 5 | Orders | 2.275.919 | -19,24 |
| 2 | Part | 399.587 | -18,38 |

Tabelle 6.10: Anzahl der Elemente, die nach Auswertung des Index noch zu verarbeiten sind.

Timeout nach einer Stunde abgebrochen im Gegensatz zu den Referenzdaten. Der Grund für die Verlängerung der Laufzeit muss am benutzten Index liegen. Dieser liegt auf der Spalte o_orderdate. Im Folgenden Abfrage 18:

```
select
  c_name, c_custkey, o_orderkey,
  o_orderdate, o_totalprice, sum(l_quantity)
from
  customer, orders, lineitem
where
  o_orderkey in (
    select
      l_orderkey
    from
      lineitem
    group by
      l_orderkey having sum(l_quantity) > 300)
  and c_custkey = o_custkey
  and o_orderkey = l_orderkey
group by
  c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by
  o_totalprice desc, o_orderdate
```

Es ist ersichtlich, dass es keine Bedingung im WHERE-Teil der Abfrage gibt, die o_orderdate eingrenzt. Der Abfrageplan 6.13 für die Ausführung mit Index zeigt,



**Abfrageplan**

Relative Kosten = 116234493

- Call module D, which populates temp-file D.
- Read temp-file D, looping on O_TOTALPRICE, O_ORDERDATE, and a counter.
- For each row:

  Output the row.

**module D**

- Call module B, which populates bitmap temp-file A.
- Call module C, which populates temp-file B.
- Read temp-file B, looping on O_TOTALPRICE, O_ORDERDATE, and ID.
- For each row:

  Read master map SQLUser.CUSTOMER.CUSTORMERPRIMARYIDX, using the given idkey value.
  Read master map SQLUser.LINEITEM.LINEITEMPRIMARYIDX, using the given L_ORDERKEY, and looping on L_LINENUMBER.
  For each row:
  Check distinct values for ID and ID using temp-file C,
  subscripted by a hashing of these values.
  For each distinct row:
  Add a row to temp-file D, subscripted by O_TOTALPRICE, O_ORDERDATE, and a counter,
  with node data of ID, C_NAME, ID, L_QUANTITY, the hashing, and [value].
  Update the accumulated sum(L_QUANTITY) in temp-file C,
  subscripted by the hashing.

**module B**

- Read index map SQLUser.ORDERS.OORDERDATEIDX, looping on O_ORDERDATE and ID.
- For each row:

  Add ID bit to bitmap temp-file A.

**module C**

- Read bitmap temp-file A, looping on ID.
- For each row:

  Read master map SQLUser.ORDERS.ORDERSPRIMARYIDX, using the given idkey value.
  Add a row to temp-file B, subscripted by O_TOTALPRICE, O_ORDERDATE, and ID,
  with node data of O_CUSTKEY.

Abbildung 6.13: Abfrageplan TPC-H Abfrage 18 mit zusätzlichem Index.

dass der Index auf der Spalte o_orderdate dennoch verwendet wird. Für die Verlängerung der Laufzeit ist die Ausführung von Modul B verantwortlich, welches als erstes ausgeführt wird.

```
Read index map SQLUser.ORDERS.OORDERDATEIDX,
 looping on O_ORDERDATE and ID.
For each row:
Add ID bit to bitmap temp-file A.
```

Modul B schreibt jeden Eintrag aus der indizierten Spalte o_orderdate in eine temporäre Datei, mit der die weiteren Berechnungen für die Abfrage ausgeführt werden. Dies ist kostenintensiver als ein sequentielles Durchlaufen der Tabelle *orders*, wie es bei der Abfragebearbeitung ohne Index vorkommt (siehe dazu Abbildung 6.14). Hier

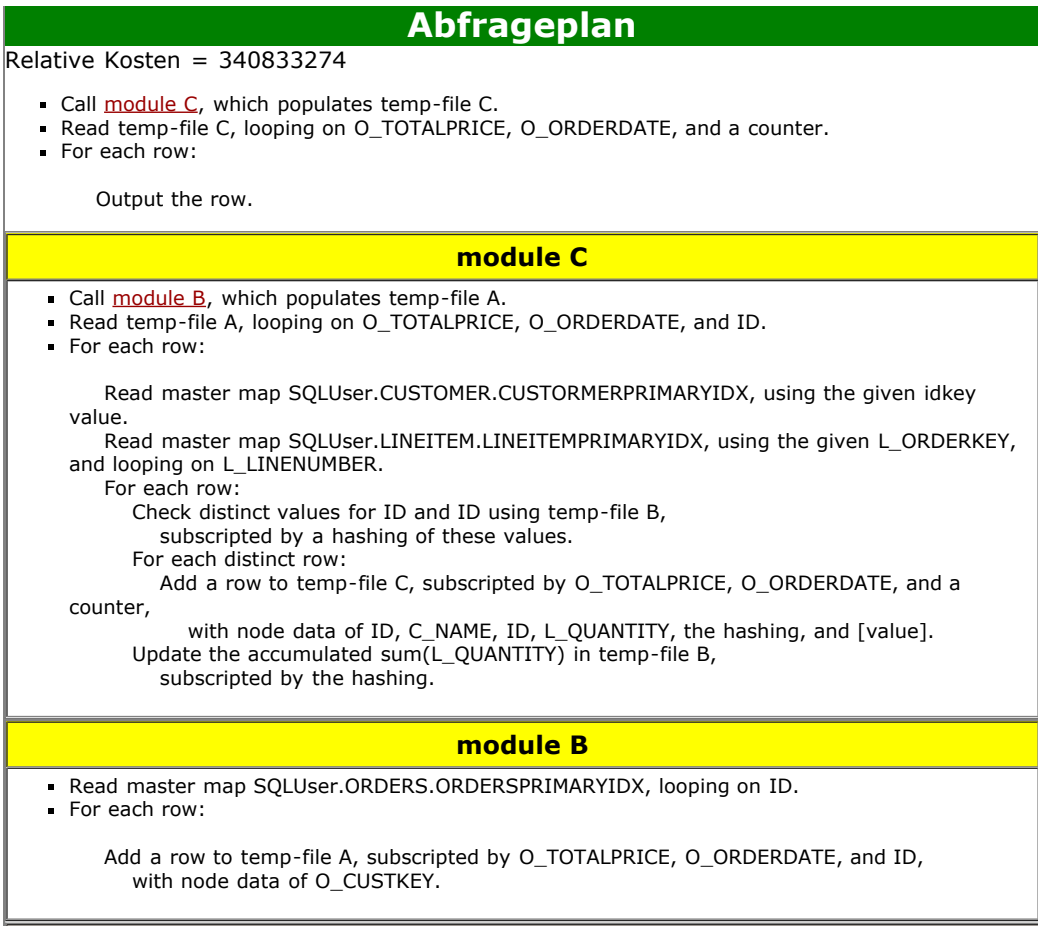



Abbildung 6.14: Abfrageplan TPC-H Abfrage 18 ohne zusätzlichen Index.

werden in Modul B die Bearbeitungsschritte ausgeführt, die im Abfrageplan mit zusätzlichem Index 6.13 in Modul B und C ausgeführt werden. Beide Module führen eine Iteration über alle Elemente von o_orderdate aus.

```
Modul C
Read bitmap temp-file A, looping on ID.
For each row:
Read master map SQLUser.ORDERS.ORDERSPRIMARYIDX,
 using the given idkey value.
Add a row to temp-file B,
 subscripted by O_TOTALPRICE, O_ORDERDATE, and ID,
with node data of O_CUSTKEY.
```

Daraus resultiert die höhere Laufzeit und der höhere Energieverbrauch bei der Ausführung mit zusätzlichem Index.

Ein ähnliches Bild ergibt sich für Abfrage 16:

```
select
  p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt
from
  partsupp, part
where
  p_partkey = ps_partkey
  and p_brand <> 'Brand#45'
  and p_type not like 'MEDIUM POLISHED%'
  and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
  and ps_suppkey not in (
    select
      s_suppkey
    from
      supplier
    where
      s_comment like '%Customer%Complaints%')
group by
  p_brand, p_type, p_size
order by
  supplier_cnt desc, p_brand,p_type, p_size
```

Diese Abfrage benutzt bei der Ausführung mit zusätzlichem Index, denjenigen der für die Spalte p_size angelegt wurde (siehe dazu den Abfrageplan 6.15). Das ist nach unserer Forderung für die Nutzung von Indizes korrekt, denn p_size ist in der WHERE-Bedingung der Abfrage eingeschränkt.

```
p_size in (49, 14, 23, 45, 19, 3, 36, 9)
```

Der Abfrageplan 6.15 zeigt, dass in Modul B eine Iteration mit eingrenzendem Wertebereich über dem Index auf der Spalte p_size ausgeführt wird.

**Abfrageplan**

Relative Kosten = 107132418

- Call module B, which populates temp-file A.
- Call module F, which populates temp-file C.
- Read temp-file C, looping on count(PS_SUPPKEY), %SQLUPPER(P_BRAND), %SQLUPPER(P_TYPE), P_SIZE, and a counter.
- For each row:

  Output the row.

**module B**

- Read index map SQLUser.PART.PSIZEIDX, looping on P_SIZE (with a given set of values) and ID.
- For each row:

  Read master map SQLUser.PART.PARTPRIMARYIDX, using the given idkey value.
  Read master map SQLUser.PARTSUPP.PARTSUPPPRIMARYIDX, using the given PS_PARTKEY, and looping on PS_SUPPKEY.
  For each row:
  Check distinct values for %SQLUPPER(P_BRAND), P_SIZE, and %SQLUPPER(P_TYPE) using temp-file A,
  subscripted by a hashing of these values.
  For each distinct row:
  Add a row to temp-file A, subscripted by the hashing,
  with node data of P_SIZE, %SQLUPPER(P_TYPE), %SQLUPPER(P_BRAND), and PS_SUPPKEY.
  Check distinct values for %SQLUPPER(P_BRAND), P_SIZE, %SQLUPPER(P_TYPE), and PS_SUPPKEY using a temp-file.
  For each distinct row:
  Update the accumulated count(PS_SUPPKEY) in temp-file A,
  subscripted by the hashing.

**module F**

- Read temp-file A, looping on the hashing subscript.
- For each row:

  Add a row to temp-file C, subscripted by count(PS_SUPPKEY), %SQLUPPER(P_BRAND), %SQLUPPER(P_TYPE), P_SIZE, and a counter,
  with node data of [expression] and [expression].

**subquery**

- Read master map SQLUser.SUPPLIER.SUPPLIERPRIMARYIDX, using the given idkey value

Abbildung 6.15: Abfrageplan TPC-H Abfrage 16 mit zusätzlichem Index.

```
Read index map SQLUser.PART.PSIZEIDX,
looping on P_SIZE (with a given set of values) and ID.
```

Bei der Ausführung ohne zusätzlichen Index. Zeigt der Abfrageplan 6.16, dass in

**Abfrageplan**

Relative Kosten = 116187574

- Call module B, which populates temp-file A.
- Call module F, which populates temp-file C.
- Read temp-file C, looping on count(PS_SUPPKEY), %SQLUPPER(P_BRAND), %SQLUPPER(P_TYPE), P_SIZE, and a counter.
- For each row:

  Output the row.

**module B**

- Read master map SQLUser.PART.PARTPRIMARYIDX, looping on ID.
- For each row:

  Read master map SQLUser.PARTSUPP.PARTSUPPPRIMARYIDX, using the given PS_PARTKEY, and looping on PS_SUPPKEY.
  For each row:
  Check distinct values for %SQLUPPER(P_BRAND), P_SIZE, and %SQLUPPER(P_TYPE) using temp-file A,
  subscripted by a hashing of these values.
  For each distinct row:
  Add a row to temp-file A, subscripted by the hashing,
  with node data of P_SIZE, %SQLUPPER(P_TYPE), %SQLUPPER(P_BRAND), and PS_SUPPKEY.
  Check distinct values for %SQLUPPER(P_BRAND), P_SIZE, %SQLUPPER(P_TYPE), and PS_SUPPKEY using a temp-file.
  For each distinct row:
  Update the accumulated count(PS_SUPPKEY) in temp-file A,
  subscripted by the hashing.

**module F**

- Read temp-file A, looping on the hashing subscript.
- For each row:

  Add a row to temp-file C, subscripted by count(PS_SUPPKEY), %SQLUPPER(P_BRAND), %SQLUPPER(P_TYPE), P_SIZE, and a counter,
  with node data of [expression] and [expression].

**subquery**

- Read master map SQLUser.SUPPLIER.SUPPLIERPRIMARYIDX, using the given idkey value.
- Determine subquery result.

Abbildung 6.16: Abfrageplan TPC-H Abfrage 16 ohne zusätzlichen Index.

Modul B eine Iteration über die Tabelle *part* ausgeführt wird.

```
Read master map SQLUser.PART.PARTPRIMARYIDX, looping on ID.
```

Dies ist der der einzige Unterschied bei der Ausführung beider Abfragepläne. Die Anzahl der Datensätze, die die Bedingung für p_size erfüllen, ist 319.686 von 2 Millionen.

Abbildung 6.17 zeigt die Auslastung bei der Ausführung ohne Index, Abbildung 6.18 mit zusätzlichem Index. Bei der Auslastung für die Ausführung ohne Index ist im

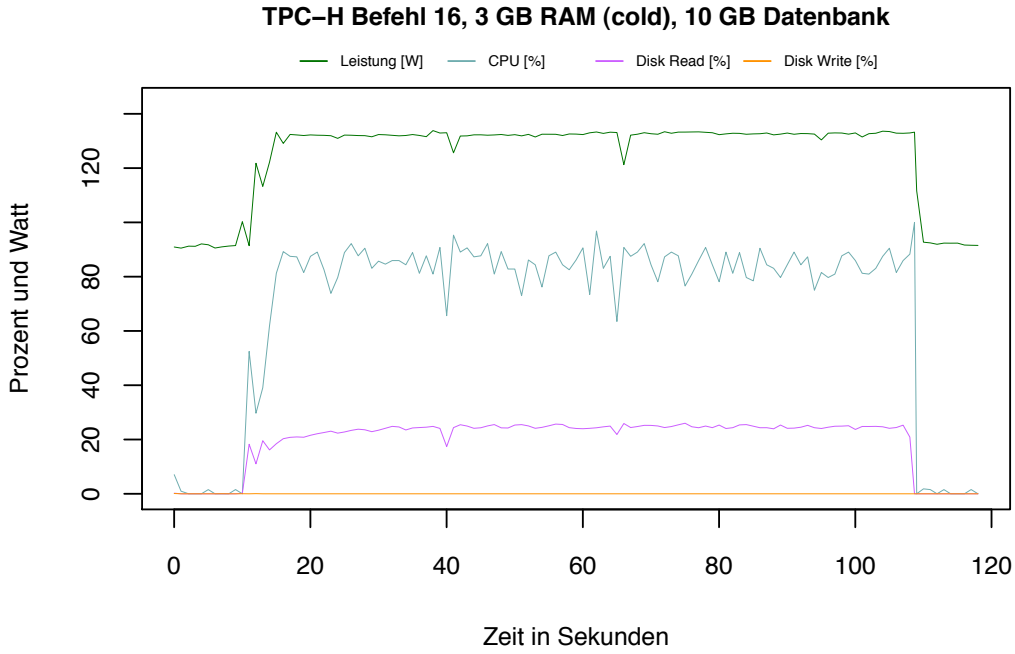



Abbildung 6.17: TPC-H Befehl 8, 3 GB RAM (cold ), 10GB Datenbank.

Vergleich zu der Ausführung mit Index zu sehen, dass die CPU-Auslastung bei über 80 % ist und die Leserate der Festplatte sehr hoch ist. Bei der Ausführung ohne zusätzlichen Index ist CPU-Auslastung niedrig, die Leserate der Festplatte auch.
Das ist damit zu begründen, dass für die Datensätze, die im Wertebereich für p_size liegen, keine temporäre Datei für die Auswertung angelegt wird, sondern die Referenzen bei der Ausführung ausgewertet werden (vgl. Modul B in 6.8. Da der Index in einem anderen Speicherbereich der Festplatte liegt. ist ein sequentielles Lesen von der Festplatte nicht möglich, da immer zwischen dem Speicherbereich des Index und dem der Daten gewechselt wird, wie an der niedrigen Leseaktivität der Festplatte in Abbildung 6.17 im Vergleich zu Abbildung 6.18 zu sehen ist. Dies ist der Grund für den höheren Energiebedarf und die längere Laufzeit bei Ausführung mit zusätzlichem Index.

Abfrage 16 und 18 haben aus unterschiedlichen Gründen einen höhere Energiebedarf und eine längere Laufzeit bei der Nutzung eines zusätzlichen Index. Der Abfrageoptimierer misst bei diesen beiden Abfragen den vorhandenen Indizes eine zu große Kostenersparnis bei. Die benutzte Heuristik führt zu einer ineffizienteren Ausführung der Abfrage. Wir haben in Caché nicht die Möglichkeit uns alle Ausführungsvarianten für eine Abfrage anzusehen und selbst auszuwählen. Caché bietet jedoch die Möglichkeit, Indizes für die Verarbeitung von Abfragen auszuschließen. Um das zu tun, muss

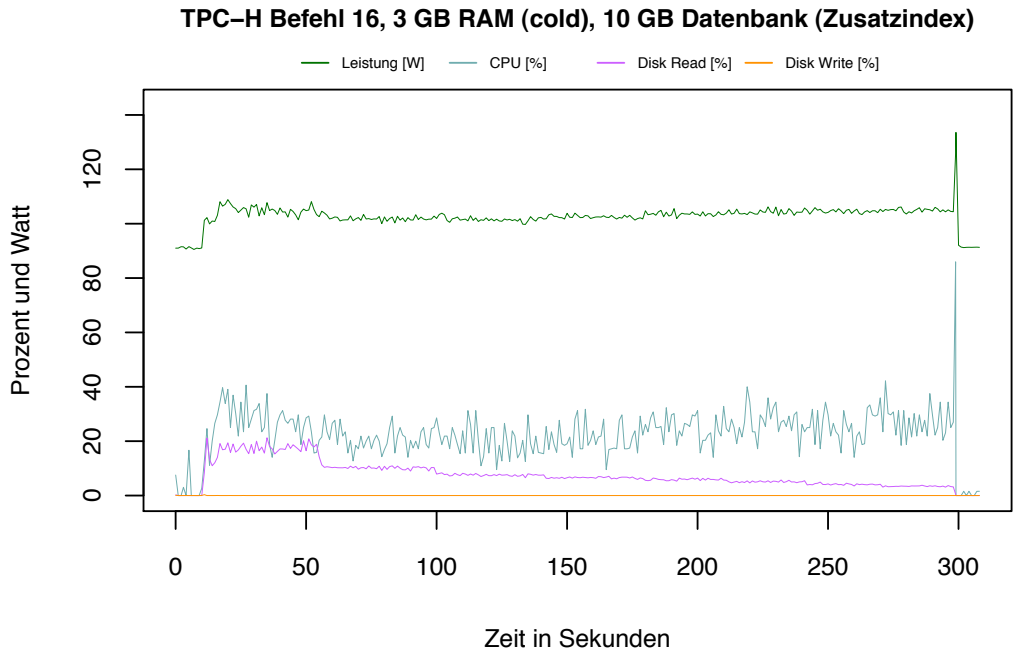



Abbildung 6.18: TPC-H Befehl 8, 3 GB RAM (cold ), 10GB Datenbank mit zusätzlichem Index.

nicht das Datenmodel verändert werden, sondern nur die Abfrage selbst. Die Syntax für das Ignorieren von Indizes ist wie folgt:

```
%IGNOREINDICES"'&STRING"'
```

Das *„STRING“* Argument ist ein Platzhalter für den Namen des Index, der ignoriert werden soll. Wir haben die SQL Syntax für Abfrage 16 und 18 mit diesem Befehl modifiziert, um den Energiebedarf beider Abfragen auf das Niveau der Referenzdaten 6.4 zurückzusetzten. Folgende Zeilen wurden der SQL Syntax hinzugefügt. In Abfrage 16 direkt nach der FROM Bedingung:

```
%IGNOREINDICES"'PSIZEIDX"'
```

und in Abfrage 18 auch direkt nach der FROM Bedingung:

```
%IGNOREINDICES"'OORDERDATEIDX"'
```

Durch diese Änderung erhalten wir die Laufzeit und den Energiebedarf aus den Referenzdaten.

#### Abfragen mit Timeout in der Referenz und mit zusätzlichem Index

Die Abfragen, die in den Referenzdaten einen Timeout hatte, und bei der Ausführung mit zusätzlichem Index. Haben einen hohen Grad an Komplexität. Wobei hier zwischen Berechnungskomplexität wie bei Abfrage 1 zu unterscheiden ist und Komplexität bei der Auswahl der Abgefragten Daten wie bei Abfrage 22. Die zusätzlich angelegten Indizes haben bei diesen Abfragen nicht dazu geführt, dass sie innerhalb einer Stunde zu Ende berechnet werden konnten. Der limitierende Faktor ist hierbei die Konfiguration unseres Testsystems, dass ab einem gewissen Grad von Komplexität der Abfrage nicht genug Kapazitäten hat um die Abfrage innerhalb einer Stunde zu beantworten.

### 6.2.5 Fazit Indizes

Wir haben in den vergangenen Abschnitten festgestellt, welche Auswirkungen Indizes auf den Energiebedarf einer Datenbank haben können. Die Indizes auf abgefragte Wertebereichen haben sich als guter Ansatz herausgestellt, um Energie bei der Abfragebearbeitung zu sparen. Abfrage 16 und 18 konnten nicht von einem zusätzlichen Index nach unserem Muster profitieren, genauso wenig wie die Abfragen aus den Referenzdaten, die nach einer Stunde abgebrochen wurden. Tabelle 6.11 zeigt das Gesamtergebnis, nachdem die SQL Syntax für Abfrage 16 und 18 wie in Abschnitt 6.2.4 angepasst wurde. Wir sehen in der Tabelle, dass durch zusätzliche Indizes 30 % der Energie im Vergleich zu den Referenzdaten eingespart wurden. Es ist an Abfrage 16 und 18 zu sehen, dass die Benutzung eines Index bei der Bearbeitung einer Abfrage nicht zwangsläufig zu einer Energieersparnis führt.
Es ist auch festzuhalten, dass der generische Ansatz, den wir für das Anlegen zusätzlicher Indizes gewählt haben, bei der Ausführung des TPC-H Workloads hohe Energieersparnisse erzielt hat. Jedoch ist die positive Wirkung der Indizes auf den Energieverbrauch von 13 Abfragen limitiert. Die Einsparungen, die durch Indizes erzielt werden können, sind von der Abfragenart abhängig und damit vom Nutzerverhalten.
Die Einschränkung des Workloads auf die 22 TPC-H Abfragen ist ein entscheidender Faktor für Energieeinsparungen mit zusätzlichen Indizes. Es ist anzunehmen, dass Abfragen, die über diese 22 Abfragen hinausgehen nicht unbedingt von den angelegten Indizes profitieren. Das Nutzerverhalten trägt einen entscheidenden Faktor zur Energieeffizienz bei. Die Einschränkung auf eine endliche Menge von Abfragen und deren Analyse sind der Schlüssel für die Energieeffizienz bei der Nutzung zusätzlicher Indizes.

| | 10 GB Datenbank (TPC-H Benchmark) mit zusätzlichen Indizes | | |
|---|---|---|---|
| Nr. | Laufzeit [s] | Joule | Joule/s |
| 1 | Timeout | N/A | N/A |
| 2 | 54,81 (-8,16 %) | 5350,89 (-18,38 %) | 97,63 (-11,13 %) |
| 3 | 1208 (-22,78 %) | 145477,7 (-19,50 %) | 120,43 (+4,24 %) |
| 4 | 203,51 (-17,87 %) | 22817,59 (-26,13 %) | 112,12(-10,06 %) |
| 5 | 527,54 (-17,14 %) | 55848,21 (-19,24 %) | 105,86 (-2,53 %) |
| 6 | 333,52 (-30,45 %) | 42706,59 (-33,69 %) | 128,05 (-4,67 %) |
| 7 | 2353,35 (-11,14 %) | 286929,06 (-9,01 %) | 121,92 (+2,40 %) |
| 8 | 1238,23 (-53,34 %) | 135992,889 (-55,92 %) | 109,83 (-5,52 %) |
| 9 | Timeout | N/A | N/A |
| 10 | 592,63 (-29,16 %) | 60352,92 (-30,93 %) | 101,84 (-2,50 %) |
| 11 | 332,93 (-10,66 %) | 45866,10 (-9,76 %) | 137,77 (+1,01 %) |
| 12 | 137,96 (-75,17 %) | 12887,89 (-82,68 %) | 93,42 (-30,25 %) |
| 13 | 589,31 (-1,03 %) | 81339,62 (-1,00 %) | 138,02 (+0,04 %) |
| 14 | 487,69 (-61,93 %) | 47486,84 (-55,60 %) | 97,37 (-14,26 %) |
| 15 | 409,35 (-59,95 %) | 44015,13 (-67,88 %) | 107,53 (-19,79 %) |
| 16 | 97,79 (N/A) | 12873,99 (N/A) | 131,49 (N/A) |
| 17 | Timeout | N/A | N/A |
| 18 | 2485,52 (N/A) | 341665,91 (N/A) | 136,90 (N/A) |
| 19 | 1886,20 (-36,13 %) | 205502,07 (-38,07 %) | 108,95 (-3,04 %) |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |
| Gesamt | 12938,33 (-29,37) | 1547113,42 (-30,26) | - |

Tabelle 6.11: Messergebnisse der 22 Abfragen des TPC-H Benchmarks auf der 10 GB-Datenbank mit zusätzlichen Indizes.

## 6.3 SSD als Optimierungsmöglichkeit

### 6.3.1 Einleitung

Obwohl *Solid State Discs (SSDs)*, die frei mit "Festkörper-Laufwerk"übersetzt werden können, schon vor Jahrzenten entwickelt wurden, sind deren Preise erst in den letzten Jahren so weit gesunken, dass ein Einsatz in Rechenzentren oder auch im Heimgebrauch überhaupt in Betracht gezogen werden kann. Auch die Speicherkapazitäten bewegen sich inzwischen in Bereichen, die für Rechenzentren interessant sind, und die Kapazitäten werden in den nächsten Jahren noch weiter wachsen.
Im Vergleich zu herkömmlichen Festplatten (HDD) befinden sich in SSDs keinerlei mechanische Bauteile, sondern viele Flash-Speicherchips. Jede dieser Flash-Zellen besteht aus einem Floating-Gate-Transistor, wie er in Abbildung 6.19 dargestellt ist. Das Anheben von Elektronen in das „Floating Gate“ wird als Schreiben bezeichnet, das Entfernen der Elektronen als Löschen. Generell wird für einen Schreibvorgang etwas mehr Energie benötigt als für das Lesen. Jede dieser Speicherzellen kann elek-

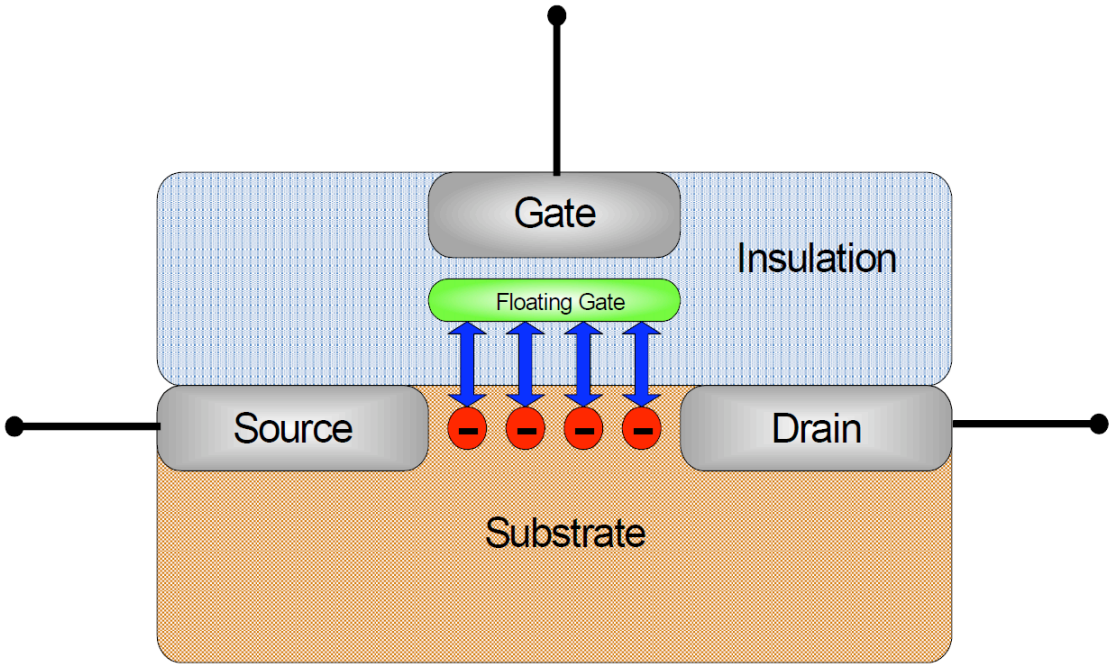



Abbildung 6.19: Aufbau einer Flashzelle [BD09].

trische Ladungen etwa zehn Jahre speichern, ohne dass die SSD mit Strom versorgt werden muss [BD09]. Der Speicherbereich einer SSD wird in Blöcke gleicher Größe aufgeteilt, die wiederum in Seiten unterteilt sind. Ein Block enthält normalerweise 32 bis 128 Seiten, und eine Seite ist 512 Byte bis 2 KB groß. Die kleinste Einheit, die gelesen werden kann, ist eine Seite, die größte ein Block.

Durch den Einsatz dieser Flash-Technologie müssen in SSDs keine Motoren mehr

zum Rotieren der Discs oder zum Bewegen der Lese-/Schreibköpfe betrieben werden. Hierdurch sinkt der Energieverbrauch von SSDs gegenüber herkömmlichen HDDs. Die fehlende Mechanik hat zusätzlich noch den Vorteil, dass die bei HDDs als Seektime bekannte Wartezeit, die durch Rotation der Discs und die Positionierung der Lese-/Schreibköpfe entsteht, entfällt. Ein weiterer Vorteil ist, dass SSDs weniger empfindlich gegenüber Erschütterungen und Temperaturschwankungen sind. Generell sind SSDs wegen der fehlenden Mechanik ausfallsicherer als HDDs [BD09].

SSDs haben aber den entscheidenden Nachteil gegenüber HDDs, dass einzelne Bits einer Seite nicht beliebig geändert werden können. Änderungen können nur von 1 (gelöscht) auf 0 (geschrieben) und blockweise durchgeführt werden. Wenn ein Bit in einem Block geändert werden soll, muss der gesamte Block samt der Änderung in einen neuen, leeren Block kopiert werden. Der ursprüngliche Block wird daraufhin gelöscht. Die Speicherung der Informationen in neuen Blöcken wird dabei durch den Controller der SSD so gesteuert, dass möglichst alle Blöcke gleich oft beschrieben werden. Dieser als *Wear Leveling* bezeichnete Prozess erhöht die Lebensdauer der SSD, da eine einzelne Speicherzelle nur begrenzt oft gelöscht und wieder beschrieben werden kann. Durch diesen aufwendigen Vorgang dauert das Schreiben von Daten auf eine SSD zwischen 1,5 und 2,0 ms [BD09], wohingegen Leseoperationen in wenigen $\mu$-Sekunden durchgeführt werden können. Besonders in diesen kurzen Zugriffszeiten bei wahlfreiem Zugriff (*Random Access*) liegen die Stärken der SSDs gegenüber HDDs. Untersuchungen haben aber gezeigt, dass HDDs beim Lesen nicht immer im Nachteil gegenüber SSDs sind. Dies gilt im Besonderen dann, wenn die zu lesenden Seiten dicht beieinander liegen. Je mehr Seiten hintereinander gelesen werden, desto geringer wird der Geschwindigkeitsvorteil von SSDs [Cor09].

In Datenbankanwendungen werden SSDs bisher selten eingesetzt, da in der Regel sehr große Datenmengen verwaltet werden und die Preise für SSDs momentan noch sehr hoch sind. Ein GB Speicherplatz kostet zwischen 1,50 und 3,00 EUR, wohingegen die Preise für HDDs zwischen 0,03 und 1,00 EUR liegen.[1] Daher muss je nach Anwendungsbereich eine Balance zwischen dem Geschwindigkeitsvorteil und den Anschaffungskosten der SSDs gefunden werden.

Daten, die schnell verfügbar sein müssen, sollten auf SSDs abgelegt werden, wohingegen Archive mit geringer Zugriffsrate durchaus auf herkömmlichen Festplatten gespeichert werden können [LD09]. Genau diesen Ansatz greifen moderne Serverlösungen auf, indem sie die verschiedenen Vor-und Nachteile der unterschiedlichen Speichermedien berücksichtigen. Solche Hybridlösungen wie z.B. die IBM System Storage DS8000-Serie [IBM10] bestehen aus verschiedenen Speichertypen und ermitteln selbstständig, welche Daten auf welchem Medium abgelegt werden.

Zusammenfassend kann gesagt werden, dass der Einsatz von SSDs eine vielversprechende Optimierungsmethode für Datenbanken ist, da dieser Speicher viele Vorteile

[1] Preise am 28.04.2011 aus Onlineshop eines Hardware-Versandhandels entnommen.

(besonders in Bezug auf den Energieverbrauch) mit sich bringt. Aus diesem Grund haben wir die Laufzeiten und den Energieverbrauch von Caché unter Verwendung einer SSD getestet. Wir vermuteten, dass die Dauer der Abfragen verkürzt würde, wodurch auch der Energieverbrauch sinkt.

### 6.3.2 Verbesserungen für den Flash-Speicher-Einsatz

Solid State Discs bzw. Flash Speicher haben aufgrund ihrer mechanikfreien Bauart, wie bereits erwähnt, einige Vorteile, wie eben auch einen geringen Energieverbrauch und hohe Performance bei wahlfreiem Zugriff. Wir konnten mit unseren Versuchen auch zeigen, dass sich der Einsatz von Flash-Speichern aus energetischer Sicht lohnt. Um das volle Potenzial von Flash-Speichern in Datenbankservern zu nutzen, reicht es allerdings nicht aus, die herkömmlichen Festplatten nur durch Flash-Speicher zu ersetzen.

Neben den positiven Eigenschaften, wie z.B. der hohen Leserate, haben Flash-Speicher den entscheidenden Nachteil, dass das Schreiben von Daten um ein Vielfaches länger dauert als das Lesen von Daten. Da eine derartige Einschränkung aber für die konventionellen Festplatten nicht gilt, wurde diese Tatsache bei der Entwicklung bisheriger Datenbankmanagementsysteme (DBMS) nicht berücksichtigt. Diese verwenden das seit Jahren für HDDs bewährte Inplace-Update, bei dem einzelne Änderungen an Datenseiten direkt in die Datenbasis auf der Festplatte übertragen werden. Für Flash-Speicher bedeutet dieser Ansatz, dass unnötig oft teure Schreibvorgänge durchgeführt werden müssen. Ein möglicher Ansatz, Flash-Speicher optimal in Datenbankanwendungen einsetzen zu können, besteht darin, das DBMS so zu verändern, dass es die Lese-/Schreibasymmetrie von Flash-Speichern berücksichtigt.

Ein ebenfalls zu beachtender Aspekt ist, dass aktuelle Datenbankmanagementsysteme berücksichtigen, dass auf HDDs unterschiedliche Zugriffsarten auch unterschiedliche Zugriffszeiten benötigen. Die drei häufigsten Zugriffsarten für eine Datenbank auf eine Datei sind:

- Sequentieller Zugriff: Hierbei werden alle Seiten einer Datei hintereinander gelesen.
- Wahlfreier Zugriff: Bei diesem Zugriff werden nur einzelne Seiten der Datei benötigt. Der Lesearm muss in der Regel viel bewegt werden, was die Lesezeit in die Höhe treibt.
- Wahlfrei sortierter Zugriff: Hierbei werden ähnlich wie bei dem wahlfreien Zugriff nur einzelne Seiten benötigt. Allerdings werden diese Seiten entweder in aufsteigender oder absteigender Reihenfolge gelesen, wodurch die Bewegungen des Lesearmes reduziert werden.

Am schnellsten können auf HDDs sequentielle Zugriffe durchgeführt werden. Da aber häufig Datensätze aus verschiedenen Seiten benötigt werden, ist diese optimale Zugriffsart nicht immer möglich. Aktuelle Datenbankmanagementsysteme überführen

daher häufig die Seitenanfragen durch Sortierung so, dass ein schneller wahlfreier sortierter Zugriff möglich ist.

Für Flash-Speicher bringen derartige Optimierungen keinen Vorteil mehr, da diese sowohl bei wahlfreiem als auch wahlfrei-sortiertem Zugriff ähnliche Lesezeiten aufweisen. Es ist also sinnvoll, bei der Entwicklung neuer Datenbankmanagementsysteme genau darauf zu achten, welche Funktionen nicht mehr, und welche unbedingt für die optimale Ausnutzung von Flash-Speichern benötigt werden. Auch wenn man den Aspekt der Energieeinsparung berücksichtigt, macht es für eine SSD keinen Unterschied, mit welcher Zugriffsart die Daten gelesen werden, wie Abbildung 6.20 zeigt.

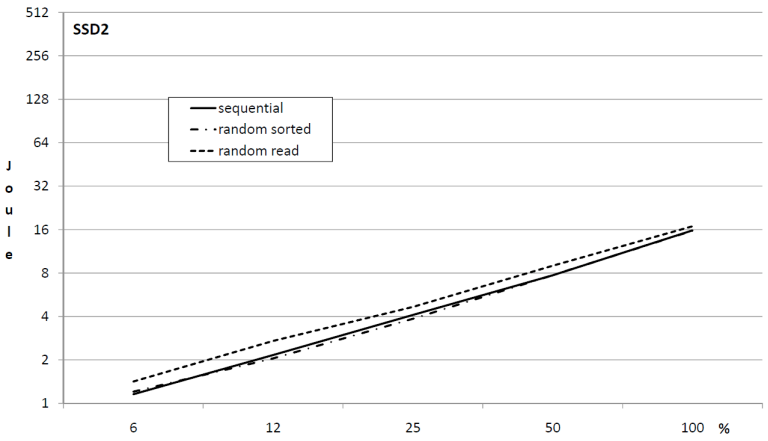



Abbildung 6.20: Energieverbrauch der verschiedenen Lesezugriffe auf einer SSD.[SHH10].

Eine dritte wichtige Beobachtung ist die, dass es große Unterschiede zwischen den SSDs gibt. Bei HDDS kann die Aussage getroffen werden, dass die Performance mit der Rotationsgeschwindigkeit steigt und alle Zugriffsarten im gleichen Verhältnis zueinander stehen. Dies bedeutet, dass eine Festplatte A, die beim sequentiellen Lesen schneller ist als eine Festplatte B, auch bei wahlfreiem Zugriff schneller ist als Festplatte B. Die Performance einer Festplatte kann daher gut vorausgesagt werden, wenn man die Spezifikation der Festplatte kennt.

Für Flash-Speicher kann eine derartige Aussage nicht getroffen werden, wie Abbildung 6.21 zeigt. Hier wird deutlich, dass die SSD3 zwar bei sequentiellem Zugriff schneller ist als die SSD2, diese bei wahlfreiem Zugriff aber schneller ist als SSD2.

Wie die obigen Tatsachen zeigen, gibt es viele Aspekte, die bei dem Einsatz von Flash-Speichern in Datenbankservern beachtet werden sollten. Sicherlich kann der Energieverbrauch allein durch die Bauart der SSD gesenkt werden, wie unsere Messungen gezeigt haben. Aber eine optimale Ausnutzung des vollen Potenzials von

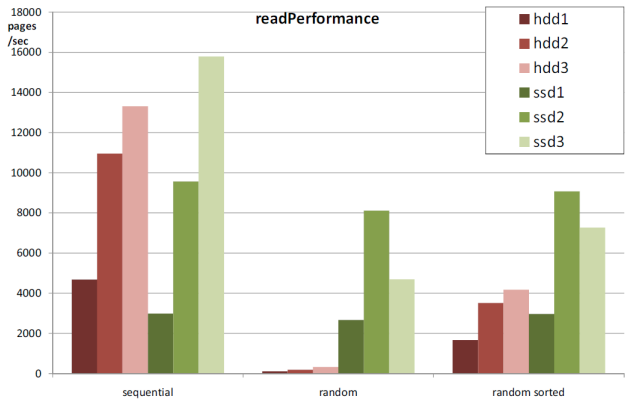



Abbildung 6.21: Vergleich der Leseperformance von drei HDDs und drei SSDs.[SHH10].

Flash-Speichern ist nur durch entsprechende Anpassungen des Datenbankmanagementsystems möglich.

So sollten neue Konzepte von Datenbankmanagementsystemen die Write-Read-Asymmetrie von Flash-Speichern berücksichtigen, um die Vorteile von Flash-Speichern optimal zu nutzen. Unnötige Schreibzugriffe, besonders aber die damit verbunden Löschoperationen sollten vermieden werden, wenn nötig auch auf Kosten von mehr Leseoperationen, denn während die Zugriffszeiten für Lese- und Schreibzugriffe im $\mu$-Sekundenbereich liegen, dauert das Löschen eines Blockes etwa 1,5 ms. Im Folgenden sollen einige Ansätze zur Optimierung von Datenbankmanagementsystemen gezeigt werden, die dies berücksichtigen.

### Die Idee des *In-Page-Loggings*

Aktuelle Datenbanken verwenden *Inplace-Updates*, bei denen Änderungen an Datensätzen direkt auf den entsprechenden Seiten der Festplatte geändert werden. Für herkömmliche HDDs ist dieses Verfahren optimal, da bei diesen Festplatten einzelne Bits einer Seite geändert werden können. Flash-Speicher hingegen können nur ganze Blöcke, in denen die Seiten enthalten sind, ändern. Das selektive Ändern nur eines einzelnen Bits ist nicht möglich, da immer der ganze Block samt der Änderung in einen leeren Block geschrieben werden muss.

**Inplace-Update:** Diese Strategie weist jeder Seite im Hauptspeicher eine Seite im Sekundärspeicher zu. Wird die Seite aus dem Datenbankmanagementsystem-Puffer verdrängt, wird deren Inhalt, sofern er verändert wurde, auf die zugeordnete Seite im Sekundärspeicher kopiert, wodurch dort der bisherige Inhalt der Seite überschrieben wird [KE09]. Da die Datenbasis auf dem Sekundärspeicher meist nicht den aktuellen Zustand der Daten enthält, da dieser nur im Puffer bearbeitet wurde, müssen alle Änderungen an den Seiten im Puffer in einer Log-Datei gespeichert werden. Nur so

kann die Konsistenz der Daten gesichert werden.

Durch die häufigen Schreibzugriffe sind die Inplace-Updates der aktuellen Datenbankmanagement-Systeme für Flash-Speicher insofern von Nachteil. Würde man die nötigen Schreibzugriffe durch andere Ansätze im Datenbankmanagementsystem auf ein Minimum reduzieren, könnten Flash-Speicher ihre Vorteile gegenüber herkömmlichen Festplatten besser ausspielen.

Die Idee, unnötige Schreibzugriffe auf ein Minimum zu reduzieren, wird beim *In-Page-Logging (IPL)* von Sang-Won Lee und Bongki Moon [Lee07] durch nur geringe Änderungen an der Datenbankarchitektur umgesetzt. Dieser Ansatz berücksichtigt ebenfalls, dass es für Flash-Speicher möglich ist, Daten überall auf dem Speicher abzulegen bzw. zu lesen, ohne dadurch an Geschwindigkeit zu verlieren, und erzielt dadurch eine für Flash-Speicher optimale Performance.

Ein weiteres Problem aktueller Datenbanken in Bezug auf SSDs zeigt sich bei der Verwaltung der Log-Einträge. Bei herkömmlichen Festplatten werden die bei der Änderung von Daten erzeugten Logdaten sequentiell auf der Festplatte abgelegt, wodurch sich die Suchzeit innerhalb der Log-Einträge reduzieren lässt. Durch diese Art der Logeintrag-Speicherung können Log-Einträge zwar schnell hintereinander geschrieben werden, aber sie werden nicht in der Nähe der eigentlichen Änderung gespeichert. Sie liegen dadurch getrennt von den Daten.

Ein Problem dieses Ansatzes ist, dass immer wenn eine Datenseite aus der Datenbank gelesen werden soll, die aktuelle Version der Seite aus der ursprünglichen Seite mit Hilfe der Log-Einträge wiederhergestellt werden muss. Da die für die Rekonstruktion einer Seite benötigten Log-Einträge aber auf der Festplatte verstreut liegen können (innerhalb des Logfiles), kann es sehr aufwendig sein, die benötigten Einträge zu finden.

Da es für SSDs keinen Vorteil bringt, die Daten sequentiell zu schreiben, ist dies nicht länger nötig, weil hier der wahlfreie Zugriff nicht schneller ist als der sequentielle Zugriff. Das hat den Vorteil, dass man die Log-Einträge auf Flash-Speichern zusammen mit einer Datenseite in einen Block des Speichers ablegen kann und diese ohne lange Suchzeit direkt zur Verfügung stehen. Hieraus resultiert auch der Name In-Page-Logging. Zwar wird auch bei diesem Ansatz die zu lesende Datenmenge durch die Log-Einträge erhöht. Dies ist aber akzeptabel, da dafür das aufwendige Suchen der Log-Einträge in anderen Speicherbereichen entfällt und die Lesegeschwindigkeit bei Flash-Speichern sehr hoch ist.

### Funktionsweise des In-Page-Loggings (IPL)

Wird der Inhalt einer sich im Speicher befindlichen Seite geändert, während die Datenbasis auf dem Sekundärspeicher noch nicht aktualisiert wurde, wird diese Seite im Arbeitsspeicher als "dirty"bezeichnet und entsprechend markiert. Wird eine Daten-

seite im Arbeitsspeicher als "dirty"markiert, kann sofort ein In-Memory-Logsektor im Arbeitsspeicher zur Verfügung gestellt werden. Dieser Logsektor kann dann auf den Flash-Speicher geschrieben werden, wenn der Logsektor im Arbeitsspeicher voll ist oder die Seite aus dem Arbeitsspeicher entfernt wird. Um diese Idee umsetzen zu können, wird ein Block des Flash-Speichers in zwei Teile unterteilt. Der eine Teil enthält die Datenseiten und der andere die Logsektoren. Die Größe des In-Memory-Logsektors muss genauso groß sein wie die des Flash-Speichers.

Wenn ein Block keine ausreichend freien Speichermöglichkeiten für die Log-Einträge mehr hat, werden die Seiten und Log-Einträge in einem neuen Block zusammengeführt. Hierbei wird aus der Vorgängerversion einer Seite und den entsprechenden Log-Einträgen die aktuelle Version einer Seite erzeugt und in dem neuen Block gespeichert. Dies geschieht für alle Seiten des vollen Blockes gleichzeitig. Da sich in dem neuen Block nun bereits die aktuellen Versionen der Seiten befinden, sind die alten Log-Einträge obsolet und müssen nicht mehr gespeichert werden. Somit ist wieder ausreichend Platz für neue Log-Einträge. Der Inhalt des ursprünglichen Blockes wird anschließend gelöscht.

Der IPL-Ansatz benötigt neben diesem den Schreibzugriff minimierenden Verfahren auch einen anderen Leseansatz. Wenn eine defekte Seite neu gelesen werden muss, geschieht dies durch das Lesen der ursprünglichen Version der Seite vom Flash-Speicher. Gleichzeitig wird der korrekte Zustand der Seite mit Hilfe der Log-Einträge, welche sich im selben Block befinden, wieder hergestellt. Zwar ist die Rekonstruktion der Seite beim Lesen teuer, aber es werden dadurch viele Schreiboperationen gespart, da das Zusammenführen nur dann ausgeführt werden muss, wenn der Logsektor voll ist. Simulationen haben gezeigt, dass die IPL-Strategie dabei helfen kann, die Nachteile von Flash-Speichern zu überwinden, und eine bessere Performance erzielt [Lee07].

### FlashDB

FlashDB ist eine für Flash-Geräte optimierte, von Suman Nath und Aman Kansal [Nat07] vorgestellte, Datenbank. Die Datenbank wird mit den Lese- und Schreibkosten des verwendeten Flash-Speichers konfiguriert und passt daraufhin ihre Speicherstruktur so an, dass der Energieverbrauch gesenkt und die Latenzzeit für den Workload optimiert wird. FlashDB besteht aus zwei Einheiten, zum einen dem Datenbank-Manager, der für die eigentlichen Datenbankoperationen wie Indexerstellung und Query-Compilation verantwortlich ist, und zum anderen dem Storage-Manager, der für die effiziente Ansteuerung des Flash-Speichers verantwortlich ist. Der Storage-Manager kontrolliert den Puffer und die Garbage-Collection. Die Architektur von FlashDB ist in Abbildung 6.22 dargestellt. Die zentrale Idee hinter FlashDB ist ein selbstanpassender $B^+$-Baum für die Indexstruktur. Für Flashspeicher gibt es zwei verschiedenen $B^+$-Bäume:

1. $B^+$-Baum(Disk): Hierbei werden die Konten des Baumes je nach Größe über mehrere hintereinander liegende Seiten auf dem Flash-Speicher abgelegt. Diese

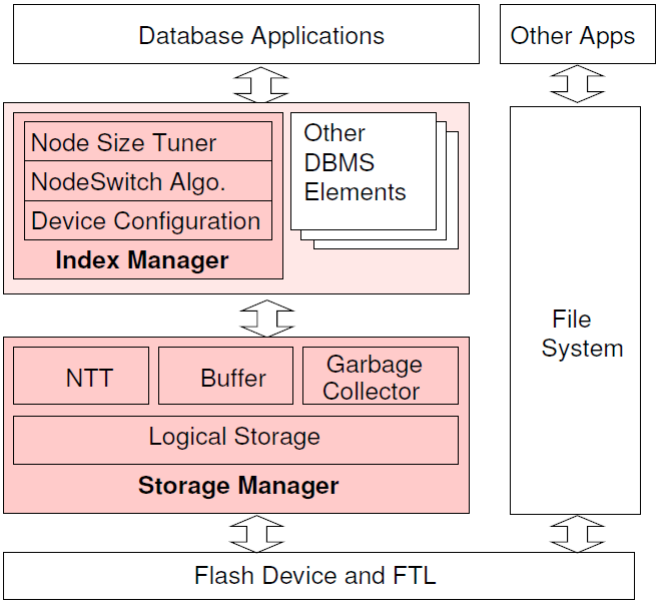



Abbildung 6.22: Die Architektur von FlashDB.[Nat07].

werden bei Bedarf eingelesen. Bei einem Update eines Knotens muss dieser in den Arbeitsspeicher gelesen und dort verändert werden. Anschließend muss der Knoten wieder auf den Flash-Speicher geschrieben werden. Dieser Vorgang ist, wie bereits erwähnt, für Flash-Speicher sehr ungünstig. $B^+$-Bäume(Disk) sind demnach für schreibintensive Workloads ungeeignet.

2. $B^+$-Baum(Log): Bei diesem Ansatz werden Änderungen an Knoten nur als Logeintrag in einem im Arbeitsspeicher gehaltenen Puffer vermerkt. Wenn dieser Puffer gefüllt ist, wird er auf den Flash-Speicher geschrieben. Wird nun ein modifizierter Knoten eingelesen, muss dieser erst mit Hilfe der Logeinträge rekonstruiert werden, da die Änderungen im Knoten selbst nicht gespeichert wurden. Diese Baumstruktur ist für leseintensive Workloads ungeeignet.

Der Vorteil von FlashDB ist, dass diese Datenbank sich selbstständig an den verwendeten Flash-Speicher und den Workload anpasst, um flexibel zu entscheiden, ob ein Indexknoten im Log-Modus oder im Disk-Modus gespeichert wird. Auf diese Weise kann die Latenzzeit verkürzt und Energie gespart werden. Diese Bäume werden als $B^+$-Bäume(ST) bezeichnet, wobei ST für „Self-Tuning“ steht.

Der Baum kann zu jedem Zeitpunkt Knoten beider Baumtypen enthalten, und ein Knoten kann seinen Typ bei Bedarf ändern. Hierbei muss allerdings berücksichtigt werden, dass das Überführen von einem Knotentyp in einen anderen ebenfalls Energie benötigt und dass diese Änderungen sich nur lohnen, wenn die Kosten durch zukünftige Operationen amortisiert werden. Das Kernstück des $B^+$-Baumes(ST) ist ein Algorithmus, welcher entscheidet, wann ein Knotentyp in einen anderen Kno-

tentyp umgewandelt werden soll. Zur Umwandlung eines Log-Knotens in einen Disk-Knoten muss dieser mit seinen Log-Einträgen rekonstruiert und anschließend auf dem Flash-Speicher gespeichert werden. Zur Umwandlung eines Disk-Knotens in einen Log-Knoten wird der Knoten in den Speicher gelesen und in eine Menge von Logeinträgen zerlegt, die den Knoten repräsentieren. Diese Logeinträge werden dann im Puffer gespeichert. Da beim Umwandeln Kosten entstehen, sollten Knoten nicht unnötig umgewandelt werden. Der Algorithmus vergleicht die Kosten für einen Knoten eines bestimmten Typs mit den Kosten, die der andere Knotentyp verursacht hätte, und überprüft dann, ob sich die Kosten für einen Wechsel des Knotentyps lohnen. Außerdem berechnet der Algorithmus über ein Nutzen-Kosten-Verhältnis, unter Berücksichtigung der Lese- und Schreibkosten des Flash-Speichers, welche Knotengröße für den Baum zu wählen ist. Dies ist wichtig, da große Knoten die Höhe des Baumes reduzieren, aber pro Knoten mehr Informationen gelesen werden müssen und dadurch wieder höhere Kosten entstehen. Zusätzlich werden die Logeinträge des Baumes so verwaltet, dass unnötige Einträge verhindert werden, indem semantisch entgegengesetzte Befehle (z.B. ADD_KEY k und DELETE_KEY k) entfernt werden (Semantic Compaction) und alte Logeinträge durch Garbage-Collection gelöscht werden, um Speicherplatz zu schaffen.

Untersuchungen mit verschiedenen Flash-Speichern und unterschiedlichen Workloads haben gezeigt, dass sich der Energieverbrauch mit den $B^+$-Bäumen(ST) von FlashDB gegenüber reinen $B^+$-Bäumen(Disk) oder $B^+$-Bäumen(log) senken lässt. Abbildung 6.23 zeigt die Ergebnisse für drei unterschiedliche Workloads (SEQ, RND und LAB), einmal nur für Insert-Operationen und einmal für Abfragen, in denen jedes Datenelement 10 Mal nach der Erzeugung des Index angefordert wird. Hier wird deutlich, dass der Energieverbrauch für alle Zugriffe auf $B^+$-Bäumen(ST) niedriger oder maximal gleich groß wie der Energieverbrauch der anderen Baumtypen ist.

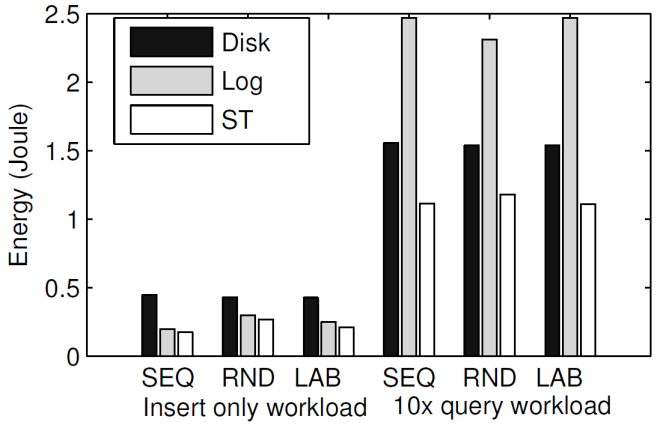



Abbildung 6.23: Vergleich der Leseperformance von drei HDDs und 3 SSDs.[Nat07].

### 6.3.3 Leistungsaufnahme des Servers im Leerlauf

Um die Leistungsaufnahme des Servers im Leerlauf (Idle) unter Verwendung eines Festkörperlaufwerks (Solid State Drive, SDD) zu ermitteln, wurden vier zehnstündige Messungen (Tabelle 6.12) durchgeführt. Die Messergebnisse unter Verwendung eines Festplattenlaufwerks (Hard Disk Drive, HDD) wurden bereits in Abschnitt 6.1.3, Tabelle 6.4) angegeben. Wie bereits bei den Messungen mit der HDD, wurden neben der Leistungsaufnahme in Watt gleichzeitig die Prozessornutzung und die Festplattenaktivität aufgezeichnet. Wieder war bei zwei der Messungen die Datenbank-Software Caché deaktiviert und bei zwei Messungen aktiviert, jedoch ohne durch SQL-Anfragen belastet zu werden. Der Zeitraum jeder der vier Messungen betrug jeweils zehn Stunden. Das Messintervall betrug wieder eine Sekunde, was zu 36.000 Datensätzen für einen Testlauf führte. Bei aktivierter Datenbanksoftware liegen die Durchschnittswer-

| Testlauf | Min. | 0,25-Quantil | Median | ∅ | 0,75-Quantil | Max. |
|---|---|---|---|---|---|---|
| mit Caché | 79,04 | 79,74 | 79,89 | 79,90 | 79,96 | 94,16 |
| mit Caché | 78,78 | 79,65 | 79,72 | 79,76 | 79,81 | 99,98 |
| ohne Caché | 78,93 | 79,62 | 79,67 | 79,66 | 79,71 | 97,32 |
| ohne Caché | 78,80 | 79,50 | 79,58 | 79,58 | 79,66 | 98,14 |

Tabelle 6.12: Leistungsaufnahme (in Watt) des Servers im Leerlauf bei Verwendung einer SSD.

te bei Verwendung einer SSD bei 79,91 Watt und 79,76 Watt. Demgegenüber stehen 90,60 Watt und 90,39 Watt bei Verwendung einer HDD. Dies ergibt im Durchschnitt eine Ersparnis von 10,48 Watt bis 10,84 Watt durch den Wechsel von einer HDD zu einer SSD.

Der bei allen Testläufen kleinste gemessene Wert liegt nahe am Durchschnittswert. Demzufolge gibt es keine Ausreißer nach unten. Die meisten Werte liegen sehr dicht beieinander, und vor allem die Werte im Bereich des unteren Quantils bis zum oberen Quantil liegen in keinem Fall um einen Wert größer als 0,22 Watt auseinander. Bei einer HDD betrug dieser Unterschied 0,65 Watt. Im Bereich des unterem Quantils bis zum oberen Quantil gibt es demnach bei Verwendung einer SSD eine etwas geringere Schwankung als bei Verwendung einer HDD.

Der Durchschnittswert ist gegenüber dem Median nur leicht nach oben verschoben, was darauf hindeutet, dass es nur vereinzelte Ausreißer nach oben hin gibt, welche zwar den Durchschnitt nach oben hin beeinflussen, nicht jedoch den Median in vergleichbarem Maße. Als Ausreißer werden hier Werte über 100 Watt bezeichnet. Die Anzahl der Ausreißer ist im Falle einer SSD geringer als bei einer HDD. Während es im Falle einer HDD bei jeweils 36.000 Werten in 4 Testläufen bis zu 76 Ausreißer nach oben hin gab, so sind es bei einer SSD maximal 4 Ausreißer. Auch bei durch das Betriebssystem bedingter plötzlicher Festplatten- und Prozessoraktivität unterliegt die SSD kaum Schwankungen im Energieverbrauch.

### 6.3.4 Ersetzung der HDD durch eine SSD

Um das Einsparpotenzial von Solid State Discs auf den Energieverbrauch bzw. die Laufzeit von TPC-H-Abfragen auf unserem Datenbankserver zu untersuchen, wurde in einem ersten Versuchsaufbau die HDD durch eine SSD ersetzt. Das Betriebssystem und die Datenbank wurden hierbei genau so auf der SSD installiert und konfiguriert wie zuvor auf der HDD. Anschließend wurden die 1 GB-Datenbank, die 5 GB-Datenbank und die 10 GB-Datenbank auf die SSD kopiert und in Caché eingebunden. Es wurden hierbei keine Versuche unternommen, die Datenbank auf irgendeine Art für den Einsatz auf einer SSD zu optimieren. Außerdem wurden bei diesen Messungen keine zusätzlichen Indizes benutzt. Lediglich die Primary-Keys und die Foreign-Keys aus dem TPC-H-Schema wurden, genau wie bei den Vergleichsmessungen auf der HDD, verwendet. Die dann durchgeführten Messreihen mit den 22 TPC-H-Abfragen lieferten die folgenden Messergebnisse.

#### Messergebnissse

Die Tabelle 6.13 zeigt die Laufzeiten, den Energieverbrauch in Joule und die durchschnittliche Leistungsaufnahme (Joule/s) der einzelnen SQL-Befehle des TPC-H-Workloads, wobei in Klammern die prozentualen Abweichungen zu den Messergebnissen auf der herkömmlichen Festplatte (siehe Abschnitt 6.1.3, Tabelle 6.4) angegeben sind.

Die Ergebnisse zeigen deutlich, dass viele SQL-Befehle durch den Einsatz der SSD sowohl kürzere Laufzeiten als auch einen niedrigeren Energieverbrauch aufweisen. So konnte der Energieverbrauch für Abfrage 10 um 54,83 % gesenkt werden und die Laufzeit sogar um 55,56 %. Um den positiven Einfluss der SSD auf die Laufzeit und den Energieverbrauch genauer zu untersuchen, sind in Abbildung 6.24 der Energieverbrauch, die Festplattenaktivität und die Prozessorauslastung des Datenbankservers für den Befehl 10 mit der herkömmlichen Festplatte dargestellt. Abbildung 6.25 zeigt im Vergleich hierzu dieselben Parameter für Befehl 10 auf der SSD.

In Abbildung 6.24 ist deutlich erkennbar, dass die Festplattenaktivität während der ersten 450 Sekunden nur sehr gering ist und wenige Daten gelesen werden. Auch die Prozessorauslastung ist ebenfalls sehr niedrig, wodurch der Energieverbrauch bei etwa 90 Watt liegt. Erst nach etwa 450 Sekunden steigt die Leserate der Festplatte leicht von weniger als 10 % auf 15 % bis 20 % an. Im Gegensatz dazu zeigt Abbildung 6.25, dass die Leserate der SSD direkt zu Beginn der Abfrage bei etwa 30 % liegt und dieser Wert während der gesamten Abfrage nicht mehr unterschritten wird. Hier profitiert die SSD von ihrem bauartbedingten Vorteil, sehr schnell Daten lesen zu können. Durch diesen permanent hohen Lesezugriff kann die Abfrage im Vergleich zu der HDD in etwa der Hälfte der Zeit abgearbeitet werden.
Ein weiterer Unterschied zwischen den beiden Kurven für die Leseoperationen ist der, dass die Kurve der HDD durchgängig kleineren Schwankungen unterliegt, die sich als „Zacken“ in der Kurve darstellen. Auch dies ist ein Indiz dafür, dass die

| Nr | Laufzeit [s] | Joule | Joule/s |
|---|---|---|---|
| 1 | Timeout | N/A | N/A |
| 2 | 30,21 (-49,37 %) | 3.418,83 (-47,85 %) | 113,15 (+3,00 %) |
| 3 | 998,22 (-36,19 %) | 115.767,78 (-35,94 %) | 115,97 (+0,39 %) |
| 4 | 326,53 (+31,78 %) | 34.099,89 (+10,39 %) | 104,43 (-16,23 %) |
| 5 | 368,76 (-42,08 %) | 39.839,37 (-42,39 %) | 108,03 (-0,54 %) |
| 6 | 465,74 (-2,87 %) | 56.589,47 (-12,14 %) | 121,50 (-9,54 %) |
| 7 | 1.909,42 (-27,90 %) | 223.524,18 (-29,11 %) | 117,06 (-1,68 %) |
| 8 | 1.511,25 (-43,05 %) | 183.359,28 (-40,56 %) | 121,33 (+4,37 %) |
| 9 | 2.024,79 (-43,76 %) | 246.335,26 (-38,01 %) | 121,66 (+10,22 %) |
| 10 | 371,77 (-55,56 %) | 39.468,43 (-54,83 %) | 106,16 (+1,64 %) |
| 11 | 375,91 (+0,88 %) | 47.070,86 (-7,39 %) | 125,22 (-8,19 %) |
| 12 | 510,97 (-8,04 %) | 62.296,16 (-16,29 %) | 121,92 (-8,97 %) |
| 13 | 592,68 (-0,47 %) | 75.163,88 (-8,51 %) | 126,82 (-8,08 %) |
| 14 | 516,59 (-52,97 %) | 61.834,66 (-50,43 %) | 119,70 (+5,40 %) |
| 15 | 1.000,39 (-2,12 %) | 121.524,15 (-11,31 %) | 121,48 (-9,38 %) |
| 16 | 120,31 (+21,91 %) | 13.530,17 (+4,29 %) | 112,46 (-14,46 %) |
| 17 | Timeout | N/A | N/A |
| 18 | 2.624,69 (+5,18 %) | 324.977,03 (-4,88 %) | 123,82 (-9,56 %) |
| 19 | 1.397,94 (-52,66 %) | 166.721,25 (-49,76 %) | 119,26 (+6,14 %) |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.13: Laufzeiten und Energieverbrauch der SSD bei 8 KB großen Blöcken für die 22 TPC-H-Abfragen. Die Werte in Klammern sind die prozentuale Abweichung zu den Werten der HDD mit 8 KB.

SSD weniger anfällig für die Anordnung der Daten auf dem Speicher ist. Die Kurve der SSD hingegen ist annähernd „zackenfrei“. Generell liegt die durchschnittliche Leistungsaufnahme des Servers unter Verwendung der SSD etwa 10 Watt unter der durchschnittlichen Leistungsaufnahme des Servers unter Verwendung der HDD.

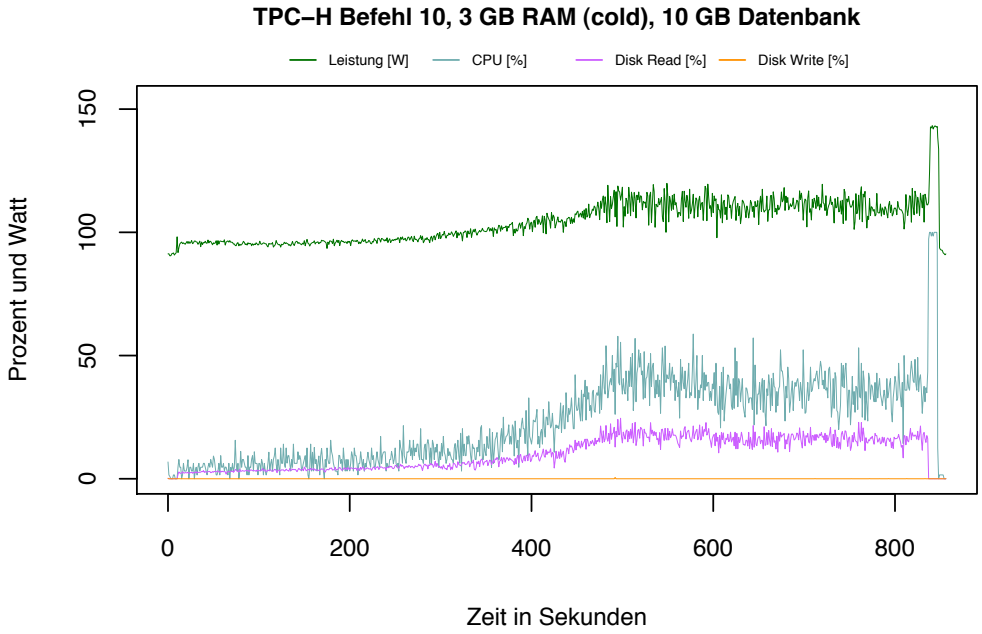



Abbildung 6.24: TPC-H Befehl 10, 3 GB RAM (cold), 10 GB Datenbank.

Interessanterweise gibt es aber auch Abfragen, die nicht durch den Einsatz der SSD profitiert haben, sondern ganz im Gegenteil sogar längere Laufzeiten und einen höheren Energieverbrauch aufzeigen. Beispiele für solche Abfragen sind die Befehle 4 und 16. Die Abbildungen 6.26 und 6.27 zeigen wieder die CPU-Auslastung sowie die Festplattenaktivität und die Leistung. Hier wird deutlich, dass die Prozessorauslastung auf der HDD etwa bei 60 % liegt und die Leserate bei etwa 30 %. Auf der SSD hingegen liegt die Prozessorauslastung nur bei 50 % und die Leserate bei knapp über 20 %. Bei Abfrage 16 zeigt sich ein ähnliches, jedoch nicht so stark ausgeprägtes Muster. Warum die Leserate der SSD in diesen Fällen unter denen der HDD liegt, konnten wir mit unseren technischen Mitteln nicht untersuchen. Auch mit Hilfe des Supports ließ sich keine Erklärung finden.

In der Abbildung 6.28 sind die prozentualen Abweichungen zwischen HDD und SSD für jede der 22 TPC-H-Abfragen eingezeichnet. Werden die 5 Abfragen, die weder auf der HDD noch auf der SSD innerhalb von 60 Minuten ausgeführt werden konnten, außer Acht gelassen, haben 9 von 17 verbleibende Abfragen deutlich vom Einbau der SSD profitiert. Ihre Laufzeiten bzw. ihr Energieverbrauch konnte pro Abfrage um

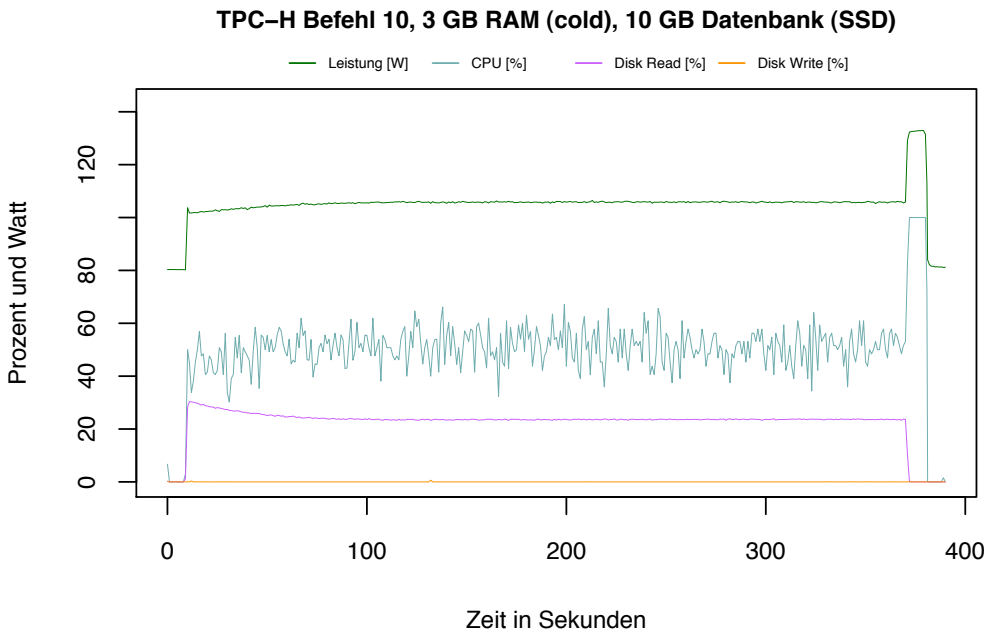



Abbildung 6.25: TPC-H Befehl 10, 3 GB RAM (cold), 10 GB Datenbank (SSD).

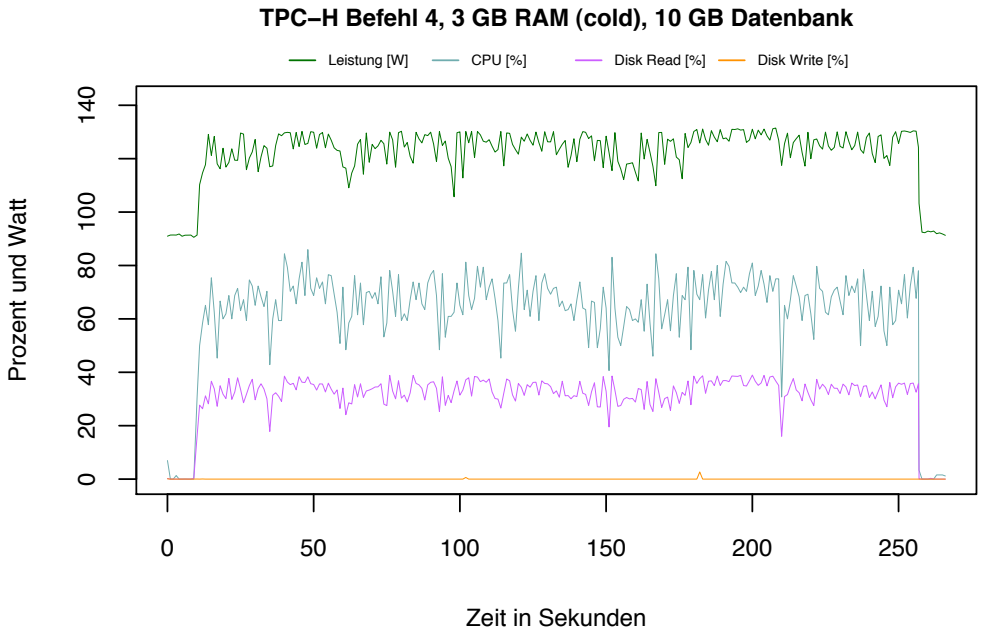



Abbildung 6.26: TPC-H Befehl 4, 3 GB RAM (cold), 10 GB Datenbank.

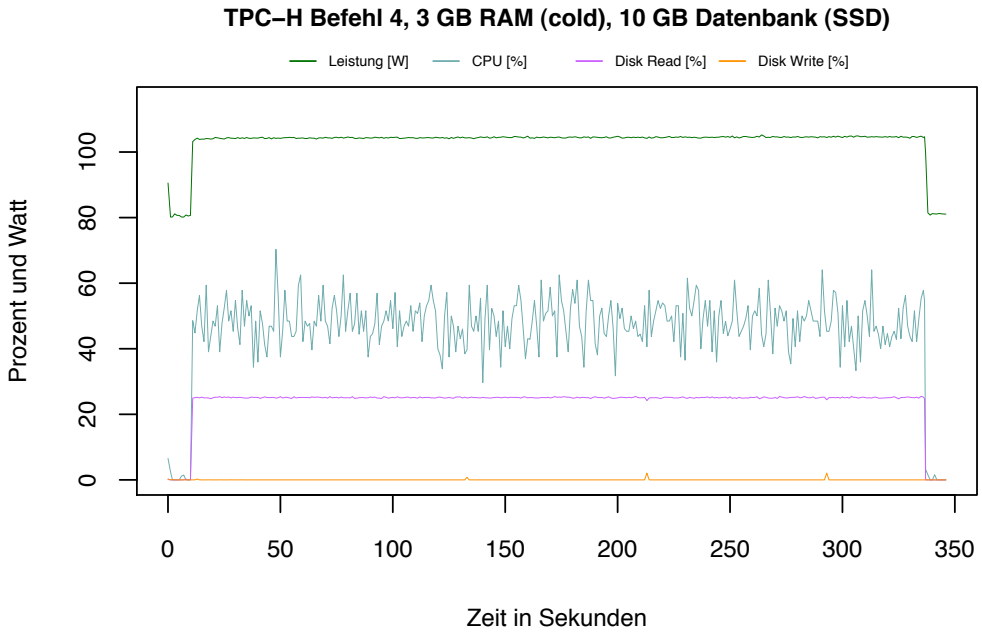



Abbildung 6.27: TPC-H Befehl 4, 3 GB RAM (cold), 10 GB Datenbank (SSD).

mindestens 25 % gesenkt werden. Insgesamt konnte durch den Einsatz der SSD in unseren Versuchen ohne weitere Optimierungsmaßnahmen die Gesamtlaufzeit aller 16 berücksichtigten TPC-H Befehle um 17,32 % und der Energieverbrauch um 18,17 % gesenkt werden.

### 6.3.5 Auswirkungen der Blockgröße

In der Literatur wird für OLAP-Systeme (**O**n**l**ine **A**nalytical **P**rocessing), die der TPC-H Benchmark simuliert, empfohlen, eine möglichst große Datenbank-Blockgröße zu verwenden [Ada10]. Durch diese Einstellungen können pro Lesevorgang mehr Daten in den Arbeitsspeicher gelesen und anschließend verarbeitet werden. Diesen Ansatz haben wir unter dem Aspekt des Energieverbrauchs aufgegriffen und den Energieverbrauch für drei unterschiedliche Blockgrößen untersucht. Der erste Test wurde mit 2 KB großen Blöcken durchgeführt, der zweite Test mit 8 KB großen Blöcken und der dritte Test mit 64 KB großen Blöcken.

#### Messergebnisse

Die Tabellen 6.14, 6.15 und 6.16 zeigen die Messergebnisse der Testreihen mit unterschiedlichen Blockgrößen von 2 KB, 8 KB und 64 KB. Die Werte der Testreihe für 8 KB große Blöcke wurden aus Tabelle 6.13 übernommen, da die Standardblockgröße von Caché auf 8 KB eingestellt ist und daher alle bisher durchgeführten Messungen, sei es auf der HDD oder der SSD, mit 8 KB großen Blöcken durchgeführt wurden. Die

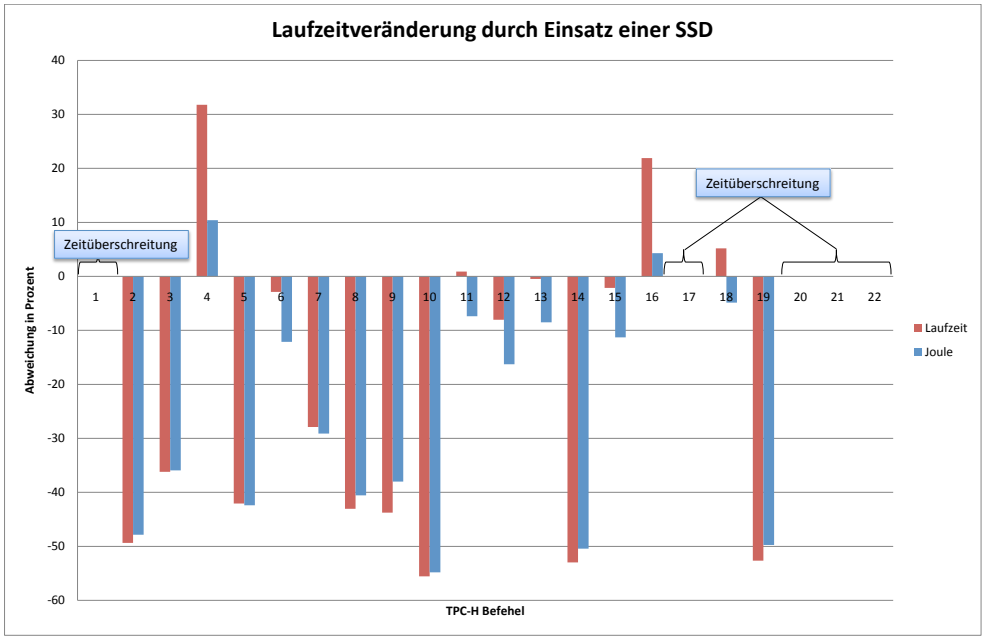



Abbildung 6.28: Veränderung der Laufzeit und des Energieverbrauches durch Einsatz einer SSD im Vergleich zur HDD.

Messergebnisse aus den Versuchen in Abschnitt 6.3.4 konnten daher einfach übertragen werden. In den Tabellen mit den Messergebnissen sind die Abfragen, die nicht innerhalb von einer Stunde ausgeführt werden konnten, wieder mit „Timeout“ markiert. Die Werte in Klammern geben die prozentuale Abweichung zu den auf der HDD gemessenen Werten an.

### Analyse der Messergebnisse (SSD Datenbank-Blockgröße 2 KB)

Die Ergebnisse in Tabelle 6.14 für die Messungen mit der SSD bei einer Blockgröße von 2 KB, zeigen, dass sich die meisten Laufzeiten im Vergleich zur HDD mit 8 KB großen Datenbank-Blöcken verlängert haben. Auch die Summe der Laufzeiten hat sich im Gegensatz zu den Laufzeiten der HDD (siehe Abschnitt 6.1.3, Tabelle 6.4) von 18.318,28 Sekunden auf 23.909,12 Sekunden verlängert, wodurch auch der Energieverbrauch bei den meisten Abfragen ebenfalls gestiegen ist. Der geringste Laufzeitanstieg beträgt 6,02 % und der größte 572,04 %.

Insgesamt haben die 16 berücksichtigten Abfragen auf der SSD mit 2 KB großen Blöcken eine Laufzeit von 23.909,12 Sekunden, also von ungefähr 6,6 Stunden. Die

| Nr | Laufzeit [s] | Joule | Joule/s |
|---|---|---|---|
| 1 | Timeout | N/A | N/A |
| 2 | 42,44 (-28,88 %) | 4710,76 (-28,14 %) | 110,99 (1,03 %) |
| 3 | 1736,22 (10,99 %) | 185201,20 (2,48 %) | 106,67 (-7,67 %) |
| 4 | 1665,26 (572,04 %) | 180521,43 (484,40 %) | 108,45 (-13,04 %) |
| 5 | 561,65 (-11,78 %) | 58108,65 (-15,97 %) | 103,46 (-4,75 %) |
| 6 | 731,66 (52,58 %) | 85834,27 (33,27 %) | 117,31 (-12,66 %) |
| 7 | 2807,98 (6,02 %) | 312234,69 (-0,98 %) | 111,20 (-6,61 %) |
| 8 | 786,29 (-70,37 %) | 80719,91 (-73,83 %) | 102,66 (-11,69 %) |
| 9 | Timeout | N/A | N/A |
| 10 | 1052,90 (25,87 %) | 100890,70 (15,47 %) | 95,82 (-8,26 %) |
| 11 | 431,72 (15,85 %) | 52955,55 (4,19 %) | 122,66 (-10,07 %) |
| 12 | 780,58 (40,48 %) | 92138,62 (23,82 %) | 118,04 (-11,86 %) |
| 13 | 727,60 (22,19 %) | 88904,38 (8,21 %) | 122,19 (-11,44 %) |
| 14 | 925,84 (-15,71 %) | 103361,46 (-17,14 %) | 111,64 (-1,69 %) |
| 15 | 2438,61 (138,59 %) | 262388,15 ( 91,50 %) | 107,60 (-19,74 %) |
| 16 | 581,13 (488,87 %) | 55463,93 (327,50 %) | 95,44 (-27,40 %) |
| 17 | Timeout | N/A | N/A |
| 18 | 3322,44 (33,14 %) | 391733,38 (14,66 %) | 117,91 (-13,87 %) |
| 19 | 2556,36 (-13,44 %) | 284862,82 (-14,15 %) | 111,43 (-0,83 %) |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.14: Laufzeiten und Energieverbrauch der SSD bei 2 KB großen Blöcken für die 22 TPC-H-Abfragen. Die Werte in Klammern sind die prozentuale Abweichung zu den Werten der HDD.

| Nr | Laufzeit [s] | Joule | Joule/s |
|---|---|---|---|
| 1 | Timeout | N/A | N/A |
| 2 | 30,21 (-49,37 %) | 3.418,83 (-47,85 %) | 113,15 (+3,00 %) |
| 3 | 998,22 (-36,19 %) | 115.767,78 (-35,94 %) | 115,97 (+0,39 %) |
| 4 | 326,53 (+31,78 %) | 34.099,89 (+10,39 %) | 104,43 (-16,23 %) |
| 5 | 368,76 (-42,08 %) | 39.839,37 (-42,39 %) | 108,03 (-0,54 %) |
| 6 | 465,74 (-2,87 %) | 56.589,47 (-12,14 %) | 121,50 (-9,54 %) |
| 7 | 1.909,42 (-27,90 %) | 223.524,18 (-29,11 %) | 117,06 (-1,68 %) |
| 8 | 1.511,25 (-43,05 %) | 183.359,28 (-40,56 %) | 121,33 (+4,37 %) |
| 9 | 2.024,79 (-43,76 %) | 246.335,26 (-38,01 %) | 121,66 (+10,22 %) |
| 10 | 371,77 (-55,56 %) | 39.468,43 (-54,83 %) | 106,16 (+1,64 %) |
| 11 | 375,91 (+0,88 %) | 47.070,86 (-7,39 %) | 125,22 (-8,19 %) |
| 12 | 510,97 (-8,04 %) | 62.296,16 (-16,29 %) | 121,92 (-8,97 %) |
| 13 | 592,68 (-0,47 %) | 75.163,88 (-8,51 %) | 126,82 (-8,08 %) |
| 14 | 516,59 (-52,97 %) | 61.834,66 (-50,43 %) | 119,70 (+5,40 %) |
| 15 | 1.000,39 (-2,12 %) | 121.524,15 (-11,31 %) | 121,48 (-9,38 %) |
| 16 | 120,31 (+21,91 %) | 13.530,17 (+4,29 %) | 112,46 (-14,46 %) |
| 17 | Timeout | N/A | N/A |
| 18 | 2.624,69 (+5,18 %) | 324.977,03 (-4,88 %) | 123,82 (-9,56 %) |
| 19 | 1.397,94 (-52,66 %) | 166.721,25 (-49,76 %) | 119,26 (+6,14 %) |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.15: Laufzeiten und Energieverbrauch der SSD bei 8 KB großen Blöcken für die 22 TPC-H-Abfragen. Die Werte in Klammern sind die prozentuale Abweichung zu den Werten der HDD mit 8 KB.

| Nr | Laufzeit [s] | Joule | Joule/s |
|---|---|---|---|
| 1 | Timeout | N/A | N/A |
| 2 | 29,47 (-50,63 %) | 3363,20 (-48,70 %) | 114,14 (3,90 %) |
| 3 | 801,47 (-48,77 %) | 98588,24 (-45,45 %) | 123,01 (6,48 %) |
| 4 | 226,72 (-8,50 %) | 25086,93 (-18,79 %) | 110,65 (-11,24 %) |
| 5 | 322,93 (-49,28 %) | 37439,84 (-45,86 %) | 115,94 (6,74 %) |
| 6 | 397,33 (-17,14 %) | 49252,01 (-23,53 %) | 123,96 (-7,71 %) |
| 7 | 1776,97 (-32,90 %) | 222765,82 (-29,35 %) | 125,36 (5,29 %) |
| 8 | 449,76 (-83,05 %) | 53983,88 (-82,50 %) | 120,03 (3,25 %) |
| 9 | Timeout | N/A | N/A |
| 10 | 300,01 (-64,14 %) | 33246,32 (-61,95 %) | 110,82 (6,10 %) |
| 11 | 348,59 (-6,45 %) | 43956,72 (-13,51 %) | 126,10 (-7,55 %) |
| 12 | 446,01 (-19,73 %) | 55511,69 (-25,40 %) | 124,46 (-7,07 %) |
| 13 | 605,64 (1,71 %) | 76726,40 (-6,61 %) | 126,69 (-8,18 %) |
| 14 | 438,34 (-60,09 %) | 54260,84 (-56,50 %) | 123,79 (9,00 %) |
| 15 | 862,07 (-15,66 %) | 106849,39 (-22,02 %) | 123,94 (-7,54 %) |
| 16 | 94,73 (-4,01 %) | 11349,90 (-12,52 %) | 119,81 (-8,87 %) |
| 17 | Timeout | N/A | N/A |
| 18 | 2463,93 (-1,27 %) | 312169,22 (-8,63 %) | 126,70 (-7,45 %) |
| 19 | 1157,52 (-60,80 %) | 143181,29 (-56,85 %) | 123,70 (10,08 %) |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.16: Laufzeiten und Energieverbrauch der SSD bei 64 KB großen Blöcken für die 22 TPC-H-Abfragen. Die Werte in Klammern sind die prozentuale Abweichung zu den Werten der HDD mit 8 KB.

Laufzeit auf der HDD mit 8 KB großen Blöcken betrug 18.318,28 Sekunden, also ca. 5,1 Stunden. Die Laufzeit auf der SSD ist demnach um über eine Stunde angestiegen, was einer Steigerung um 29,4 % entspricht. Bei Betrachtung des Energieverbrauchs fällt auf, dass der geringste Energieanstieg 2,48 % beträgt und der größte 484,40 %.

Der Gesamt-Energieverbrauch der 16 TPC-H-Abfragen ohne Zeitüberschreitung beträgt auf der SSD etwa 2.656.670,88 Joule, wohingegen die gleichen Abfragen auf der HDD mit 8 KB großen Blöcken einen Gesamtverbrauch von nur 2.218.520,25 Joule haben. Der Energieverbrauch ist also um 438.150,64 Joule angestiegen, was einer Steigerung von 19,75 % entspricht. Obwohl die Laufzeit um etwa 30 % angestiegen ist, ist der Energieverbrauch nur um 20 % angestiegen.
Dies liegt vor allem daran, dass die SSD aufgrund ihrer Bauart zwar pro Sekunde weniger Energie benötigt, aber durch die kleine Blockgröße sehr viel mehr Leseoperationen durchführen muss, wodurch die Laufzeit ansteigt.

Neben diesem erwarteten Anstieg der Laufzeit und des Energieverbrauchs konnte aber auch eine Reduktion des Energieverbrauchs für fünf Abfragen (2, 5, 8 ,14 und 19) festgestellt werden. Die kleinste Laufzeiteinsparung liegt bei 11,78 % und die größte bei 70,37 %. Die kleinste Energieeinsparung liegt bei 14,15 % und die größte bei 73,83 %. Ein Vergleich der Leistungsindikatoren (Prozessor, I/O) in den Abbildungen 6.29 und 6.30 für Befehl 5 zeigt, wie es bei der SSD trotz der kleinen Blockgröße zu einer Senkung der Laufzeit und des Energieverbrauchs kommt.

Anhand von Befehl 5 wird deutlich, dass die HDD etwa über 300 Sekunden hinweg nur sehr wenige Daten ausliest, während die SSD hier sogar für wenige Sekunden über ihrer durchschnittlichen Leserate für diese Abfrage liegt. Allerdings liest die HDD nach etwa 400 Sekunden etwa 3 Mal so viele Daten wie die SSD, wodurch sie den Vorteil der SSD zu Beginn der Abfrage wieder etwas relativieren kann. Die Abbildungen 6.31 und 6.32 für Abfrage 8 zeigen ein etwas anderes Bild und verdeutlichen, warum bei dieser Abfrage noch mehr Energie und Zeit gespart werden konnte.

Auch während der Bearbeitung von Befehl 8 kann die HDD während der ersten 400 Sekunden kaum Daten einlesen, wohingegen die SSD sofort wieder etwas über ihrer durchschnittlichen Leserate bei dieser Abfrage liegt. Allerding liest die HDD im Gegensatz zu Abfrage 5 nach diesen 400 Sekunden nicht drei Mal so viele Daten wie die SSD, sondern nur etwa die Hälfte. Für die Abfragen 14 und 18 zeigen sich ähnliche Effekte; zu Beginn der Abfrage liest die HDD weniger Daten als die SSD. Im weiteren Verlauf der Abfrage steigt die Leserate dann zwar leicht über die der SSD, wobei die schlechte Leserate zu Beginn der Abfrage aber nicht mehr ausgeglichen werden kann.

### Analyse der Messergebnisse (SSD Datenbank-Blockgröße 8 KB)

Die nächste von uns untersuchte Blockgröße auf der SSD sind 8 KB große Blöcke. Eine genaue Analyse hierzu wurde bereits in Abschnitt 6.3.4 durchgeführt. Diese Messung musste daher auch nicht durchgeführt werden. Die Ergebnisse sind jedoch

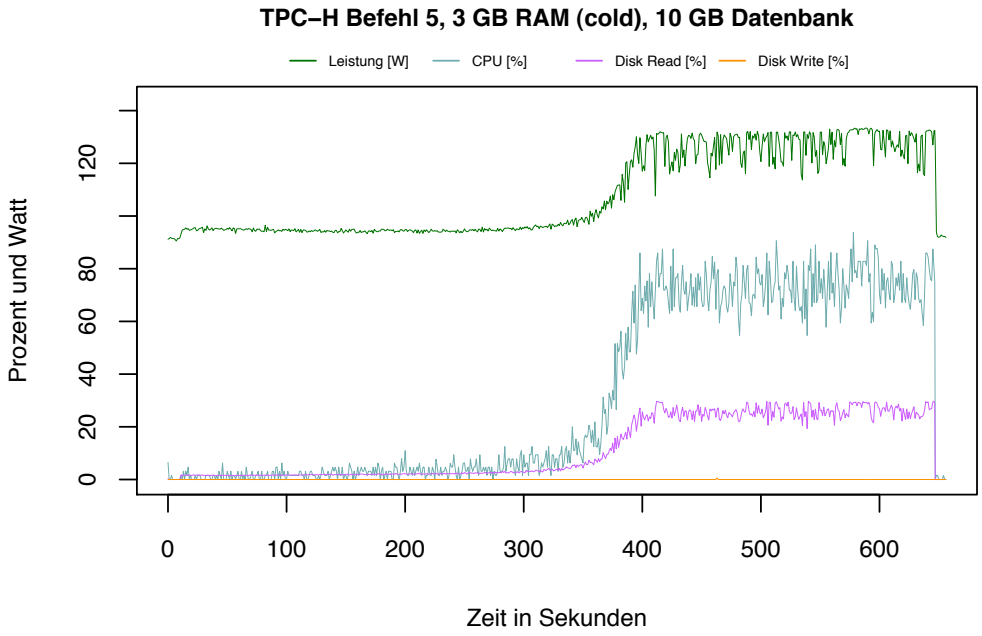



Abbildung 6.29: SQL Befehl 5, 3 GB RAM (cold), 10 GB-Datenbank, Blockgröße 8 KB.

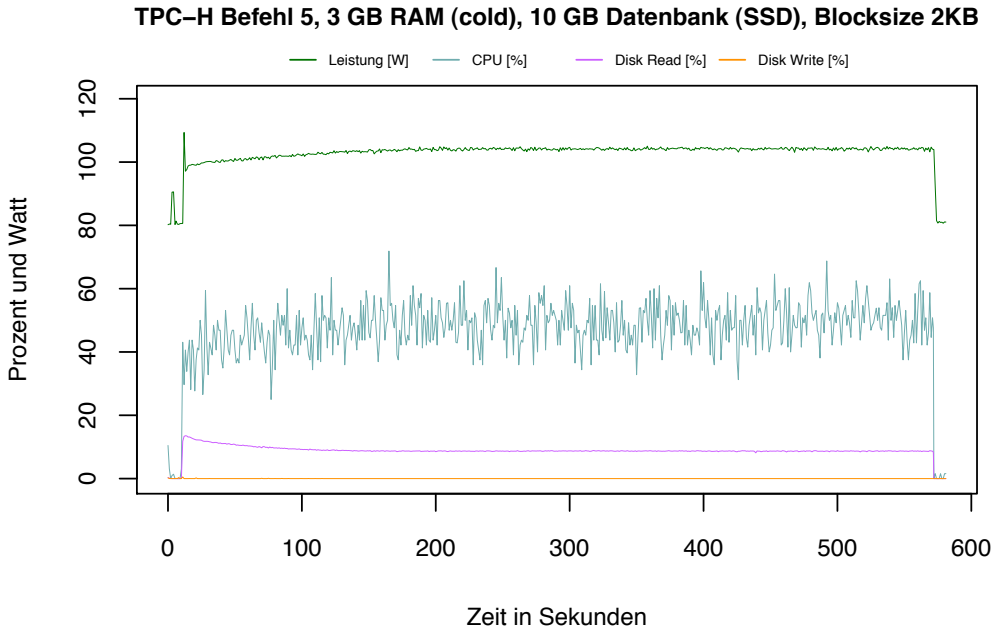



Abbildung 6.30: SQL Befehl 5, 3 GB RAM (cold), 10 GB-Datenbank (SSD), Blockgröße 2 KB.

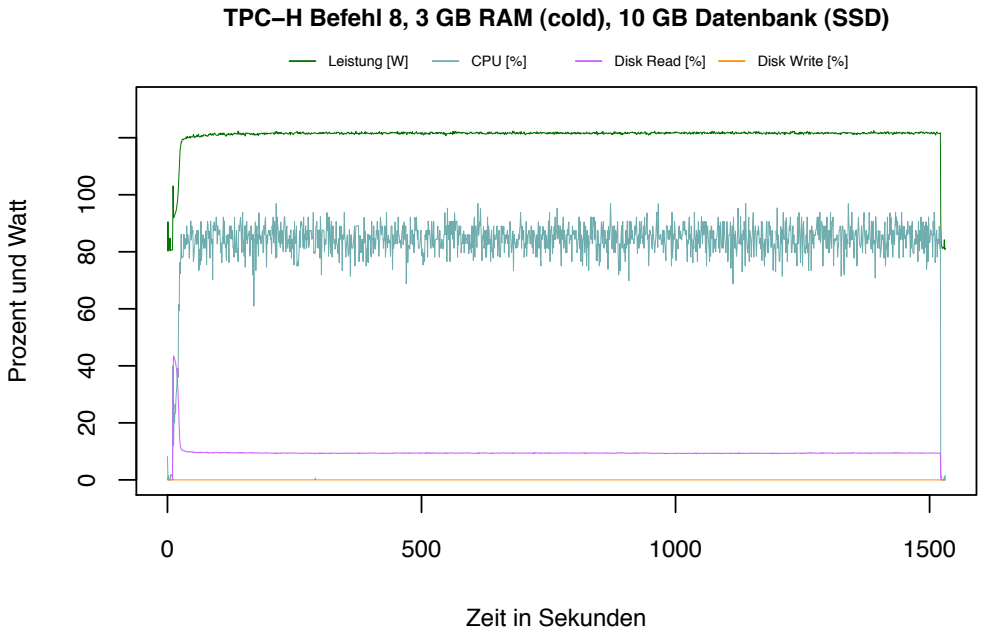



Abbildung 6.31: SQL Befehl 8, 3 GB RAM (cold), 10 GB-Datenbank, Blockgröße 8 KB.

noch einmal in Tabelle 6.15 dargestellt.

### Analyse der Messergebnisse (SSD Datenbank-Blockgröße 64 KB)

In einem weiteren Experiment wurde die Blockgröße der Datenbank auf 64 KB erhöht und die gleichen 22 TPC-H-Abfragen wurden durchgeführt. Die Messwerte dieses Versuches sind in Tabelle 6.16 dargestellt. In Klammern stehen wieder die prozentualen Abweichungen zu den Messwerten der HDD mit 8 KB.

In dieser Tabelle wird deutlich, dass alle 16 berücksichtigten TPC-H-Abfragen einen geringeren Energieverbrauch aufweisen und sich auch die Laufzeit bei 15 der 16 Abfragen verkürzt hat. Die Gesamtlaufzeit der Abfragen konnte von 18.318,28 Sekunden auf der HDD auf eine Laufzeit von 13.300,64 Sekunden gesenkt werden. Dies entspricht einer Laufzeitverkürzung von 5017,64 Sekunden, also ca. 1,4 Stunden bzw. 27,39 %. Der Energieverbrauch für alle 16 Abfragen konnte von 2.218.520,25 Joule auf 1.657.552,11 Joule gesenkt werden. Dies entspricht einer Einsparung von 560.968,14 Joule, was einer Einsparung von 25,29 % entspricht.

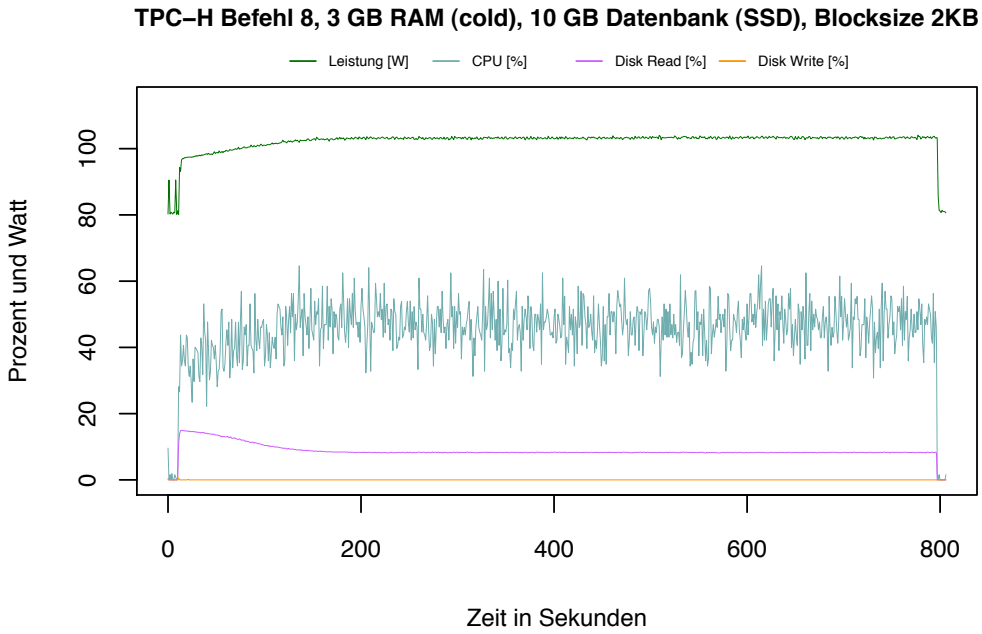



Abbildung 6.32: SQL Befehl 8, 3 GB RAM (cold), 10 GB-Datenbank (SSD), Blockgröße 2 KB.

### Analyse der Messergebnisse (Direkter Vergleich zwischen 2 KB, 8 KB und 64 KB)

Bisher wurden die Abweichungen von Laufzeit und Energieverbrauch der SSD mit unterschiedlichen Blockgrößen der Datenbank zur HDD untersucht. Ebenso interessant ist allerdings auch der Vergleich der SSD mit unterschiedlichen Blockgrößen zueinander. Ein Vergleich der Ressourcennutzung der Abfrage 5 auf der SSD mit unterschiedlichen Blockgrößen kann mit Hilfe der Abbildungen 6.30, 6.33 und 6.34 durchgeführt werden.

Die Abbildungen 6.30 zeigt die Häufigkeit der Lesezugriffe auf der SSD, die Prozessorauslastung sowie den Energieverbrauch des Datenbankservers während der Ausführung von TPC-H-Befehl 5 bei einer Blockgröße von 2 KB. Die Laufzeit des Befehls beträgt etwa 600 Sekunden. Die Grafik zeigt, dass die Leserate der SSD während der gesamten Ausführungszeit bei etwa 10 % liegt und der Prozessor zu etwa 40 % ausgelastet ist. Die Leistungsaufnahme liegt während der Messung zwischen 100 Watt und 105 Watt, was zu einem Durchschnittswert von 103,46 Joule/s führt.

Im Vergleich zu dieser Grafik zeigt Abbildung 6.33 die Ressourcennutzung des gleichen Befehls. Allerdings wurde in diesem Versuch die Standardblockgröße von 8 KB gewählt. Hier wird deutlich, dass die Leserate der Festplatte nun bei 20 % liegt und die Prozessorauslastung auf etwa 50 % angestiegen ist. Auch in diesem Fall liegt

während dieser Messung die Leistungsaufnahme des Servers zwischen 105 Watt und 110 Watt und ist somit um ca. 10 Watt angestiegen. Dies führt zu einer Erhöhung der durchschnittlichen Leistungsaufnahme auf 108,03 Joule/s. Die Laufzeit für diese Konfiguration beträgt nun allerdings nur noch 400 Sekunden und konnte somit um 200 Sekunden gesenkt werden.

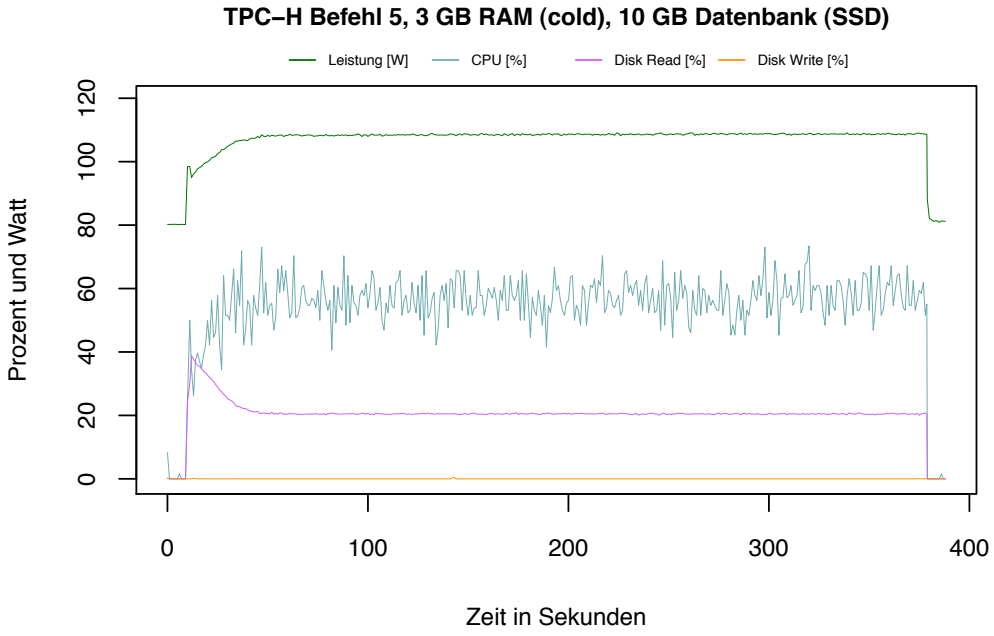



Abbildung 6.33: SQL Befehl 5, 3 GB RAM (cold), 10 GB-Datenbank (SSD), Blockgröße 8 KB.

Abbildung 6.34 schließlich zeigt die Ressourcenausnutzung des Datenbankservers bei der maximal in Caché einstellbaren Blockgröße von 64 KB. Auch in dieser Abbildung wird deutlich, dass sich die Leserate der Festplatte noch weiter erhöht hat und nun bei etwa 40 % liegt. Die Prozessorauslastung liegt jetzt bei etwa 70 %. Die Leistungsaufnahme ist im Gegensatz dazu nur um etwa 10 Watt auf 120 Watt angestiegen. Die durchschnittliche Leistungsaufnahme liegt nun bei 115,94 Joule/s.

Durch eine Vergrößerung der Blöcke erhöht sich die Leserate auf der SSD, wodurch mehr Daten in der gleichen Zeit gelesen werden können. Ein Anstieg der Prozessorleistung ist die Folge des höheren Datenaufkommens. Hierdurch verkürzen sich die durchschnittliche Laufzeit und der Energieverbrauch bei TPC-H-Abfragen.

Durch die Wahl einer großen Blockgröße lässt sich der Energieverbrauch bei den TPC-

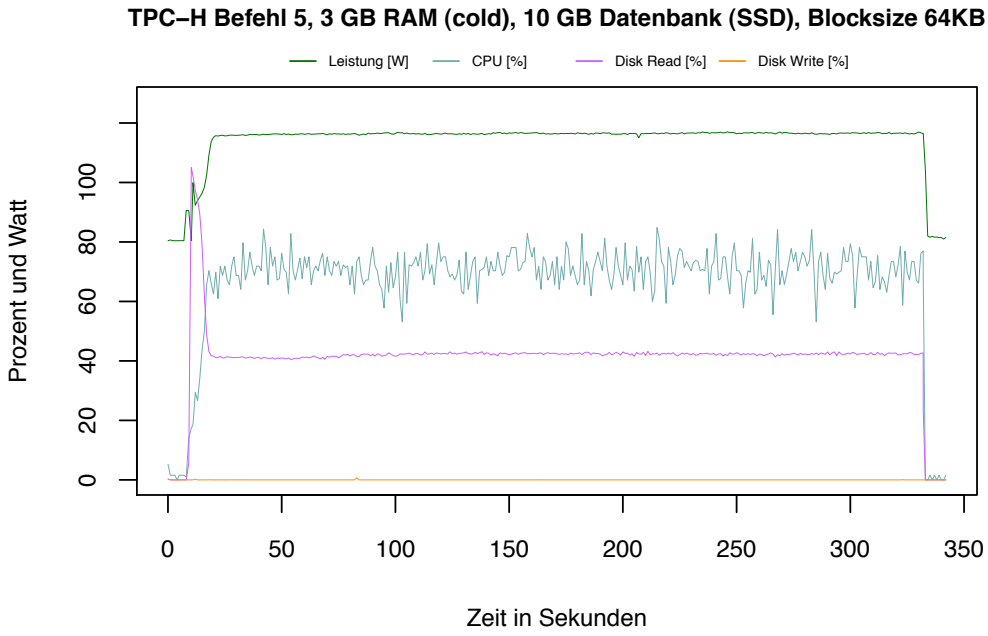



Abbildung 6.34: SQL Befehl 5, 3 GB RAM (cold), 10 GB-Datenbank (SSD), Blockgröße 64 KB.

H-Befehlen, die einen Online Analytical Processing Workload (OLAP)[2] simulieren, senken. Diese Aussage gilt vermutlich nicht für einen *Online Transaction Processing-Workload (OLTP-Workload)*, bei dem es auf die Anzahl durchgeführter Transaktionen ankommt, da für einen solchen Workload kleine Blockgrößen empfohlen werden [Ada10].

### Analyse der Messergebnisse (Gesamtbetrachtung)

Die Ergebnisse der Testreihen zeigen, dass eine Vergrößerung der Blockgröße eine Verkürzung der Laufzeit und eine Senkung des Energieverbrauchs mit sich bringt. Abbildung 6.35 zeigt die Energie, die für jeden TPC-H-Befehl bei unterschiedlichen Blockgrößen benötigt wurde. Auch hier wird deutlich, dass der Energieverbrauch mit zunehmender Blockgröße sinkt.

Die Werte in Tabelle 6.17 zeigen die durchschnittliche prozentuale Abweichung der Gesamtlaufzeit, des Gesamtenergieverbrauchs und der Energieaufnahme pro Sekunde bei unterschiedlichen Blockgrößen der Datenbank auf der SSD im Vergleich zur HDD mit einer Blockgröße von 8 KB. Hier wird deutlich, dass sich der Energieverbrauch und die Laufzeit mit zunehmender Blockgröße der Datenbank senken lassen.

[2]Hierbei kommt es auf die effiziente Verarbeitung großer Datenmengen an und nicht auf die Verarbeitung möglichst vieler Transaktionen.

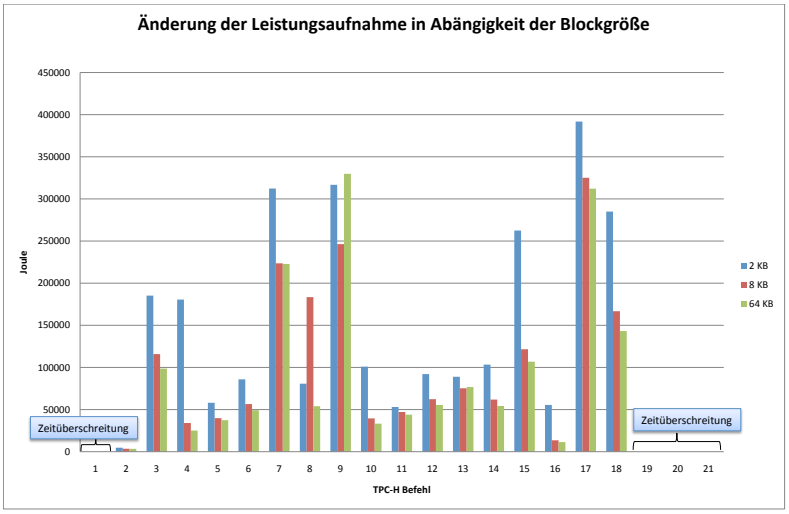



Abbildung 6.35: Energievergleich der 22 TPC-H Befehle bei unterschiedlichen Blockgrößen.

Diese Ergebnisse lassen sich so erklären, dass durch das Lesen größerer Blöcke der Datenbank pro Sekunde mehr Daten zur Verfügung stehen, die ausgewertet werden können, und insgesamt weniger Leseoperationen durchgeführt werden müssen. Die Prozessorleistung steigt an, da mehr Daten zur Auswertung zur Verfügung stehen. Hierdurch können die Abfragen in kürzerer Zeit bearbeitet werden. Die Zunahme der Leistungsaufnahme bei den 8 KB und 64 KB großen Blöcken, meist verursacht durch eine höhere CPU-Nutzung, wird durch die kürzere Gesamtlaufzeit der Abfragen kompensiert, was insgesamt zu einem geringeren Energieverbrauch führt.

| Blockgröße | Laufzeit | Joule | Joule/s |
|---|---|---|---|
| 2 | +30,52 % | +19,75 % | -4,63 % |
| 8 | -17,32 % | -18,17 % | +1,55 % |
| 64 | -27,39 % | -25,29 % | +4,96 % |

Tabelle 6.17: Die durchschnittliche prozentuale Abweichung der Gesamtlaufzeit, des Gesamtenergieverbrauches und der Energieaufnahme pro Sekunde bei unterschiedlichen Blockgrößen der Datenbank auf der SSD im Vergleich zur HDD mit einer Blockgröße von 8 KB.

Abbildung 6.36 veranschaulicht die Ergebnisse aus Tabelle 6.17, für die Laufzeit und die Energie. Hier wird auch deutlich, dass durch Änderung der Blockgröße sowohl

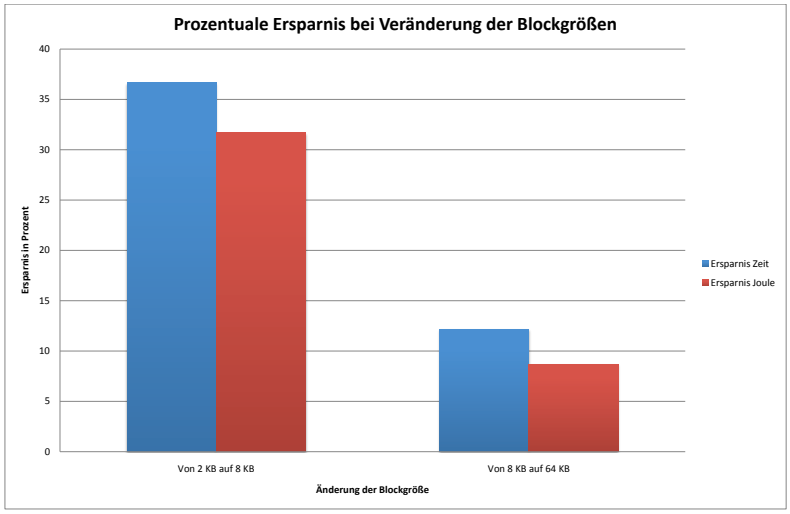



Abbildung 6.36: Die durchschnittliche prozentuale Abweichung der Gesamtlaufzeit und des Gesamtenergieverbrauches bei unterschiedlichen Blockgrößen der Datenbank auf der SSD im Vergleich zu HDD mit einer Blockgröße von 8 KB.

Laufzeit als auch Energie eingespart werden kann. Je größer die Blockgröße ist, desto mehr Energie wird gespart. Allerdings wird auch deutlich, dass die Einsparung durch eine Vergrößerung der Blöcke von 8 KB auf 64 KB geringer ist als die Einsparung durch die Erweiterung von 2 KB großen Blöcken auf 8 KB große Blöcke. Durch Darstellung desselben Sachverhaltes als Liniendiagramm wird diese Tatsache durch eine immer flacher werdende Kurve in Abbildung 6.37 deutlich.

### Zusammenfassung

Wir haben durch unsere Messungen gezeigt, dass durch den Einsatz von Solid State Disks als Sekundärspeicher anstelle von herkömmlichen Festplatten, der durchschnittliche Energieverbrauch im Single-User-Betrieb bei einen *Online Analytical Processing-Workload (OLAP-Workload)* um rund 20 % gesenkt werden kann, wenn als Blockgröße der Datenbank sowohl auf der HDD als auch auf der SSD 8 KB gewählt wird.

Wird hingegen die Datenbank-Blockgröße auf der SSD auf 64 KB vergrößert, können im Vergleich zur HDD mit 8 KB großen Blöcken bis zu 30 % Energie gespart werden. Die Ergebnisse (grafisch dargestellt in Abbildung 6.37) legen die Vermutung nahe, dass eine Erhöhung der Blockgröße auch über 64 KB hinaus den Energieverbrauch des TPC-H Benchmarks zusätzlich, wenn auch weniger deutlich senken kann. Die ma-

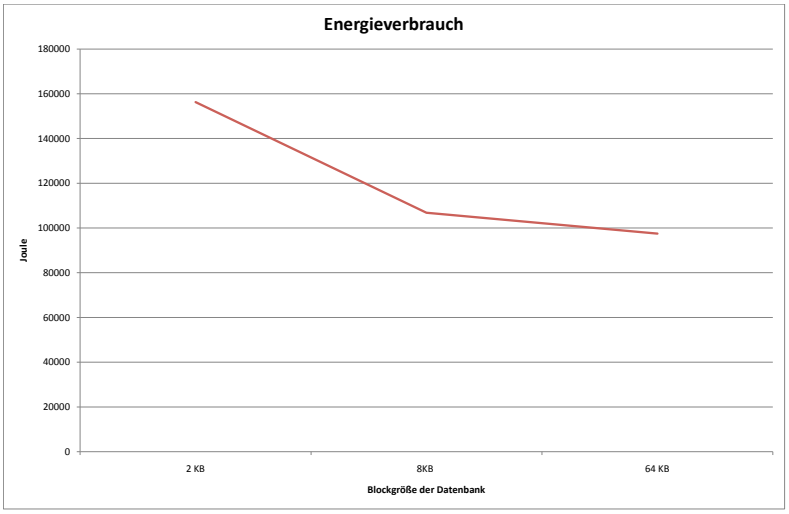



Abbildung 6.37: Der durchschnittliche Energieverbrauch der 16 berücksichtigten Abfragen bei unterschiedlichen Blockgrößen (2 KB, 8 KB und 64 KB) auf der SSD.

ximal zulässige Datenbank-Blockgröße in Caché beträgt allerdings 64 KB, weshalb keine weiteren Messungen mit größeren Blockgrößen durchgeführt werden konnten.

## 6.4 Spannungs- und Frequenzskalierung des Prozessors

### 6.4.1 Einleitung

In den folgenden Abschnitten wird eine Möglichkeit aufgezeigt, mit welcher sich der Energieverbrauch eines Datenbank-Servers direkt reduzieren lässt. Bei dieser Möglichkeit wird Performance gegen Energie getauscht. Als Workload dienen im Folgenden die 22 Abfragen des TPC-H Benchmarks, welche einen realistischen Workload in einem Unternehmensumfeld simulieren. Die Implementierung des TPC-H Datenmodells in Caché wurde bereits in Abschnitt 6.1.1 beschrieben. Ziel der folgenden Untersuchungen ist es, Konfigurationsmöglichkeiten von Prozessoren zu nutzen, um den Energieverbrauch für die Verarbeitung des TPC-H Workloads zu reduzieren. Zusätzlich wird eine Heuristik angegeben, mit der sich die Veränderung des Energieverbrauchs und der Laufzeit für einzelne Abfragen ermitteln lässt.

Bei modernen Prozessoren gibt es die Möglichkeit, die Spannung und die Frequenz zu verändern. Um den Energieverbrauch eines DBMS zu senken, ist es wünschenswert, die Spannung und die Frequenz des Serverprozessors an den aktuellen Nutzungsgrad des Servers dynamisch anzupassen. Messungen in Kapitel 5 haben bereits gezeigt, dass es während der Bearbeitung einer Abfrage I/O-lastige und CPU-lastige Phasen gibt. Durch die Messergebnisse in den folgenden Abschnitten wird gezeigt, dass ein Prozessor in I/O-lastigen Phasen mit einer niedrigeren Spannung und Frequenz betrieben werden kann bei einer vergleichbar geringen Zunahme der Laufzeit. In unseren Messungen mit dem DBMS *Caché* konnte der Energieverbrauch um 2,74 % gesenkt werden bei einer Zunahme der Laufzeit von 2,09 %. Der Grund für die Reduzierung des Energieverbrauchs sind unterschiedliche Prozessorzustände gemäß des ACPI-Standards [ACP].

Als eine Möglichkeit zur Reduzierung der durchschnittlichen Leistungsaufnahme eines Datenbank-Servers, werden die Auswirkungen einer Frequenz-Drosselung (engl. *Frequency-Throttling*) des Prozessors auf das DBMS Caché untersucht. Bei einer Frequenz-Drosselung ohne Änderung der Prozessor-Spannung konnten allerdings auch bei I/O-lastigen Abfragen, die den Prozessor nicht vollständig auslasten, nur eine niedrigere durchschnittliche Leistungsaufnahme und keine Reduktion des Energieverbrauchs des DBMS festgestellt werden. Dies liegt an den deutlich erhöhten Laufzeiten der Abfragen. Damit führt Prozessor-Throttling lediglich zu einer Senkung der durchschnittlichen Leistungsaufnahme und nicht zur Senkung des Energieverbrauchs eines Datenbank-Servers.

### 6.4.2 Prozessorzustände nach ACPI-Standard

Das *Advanced Configuration and Power Interface (ACPI)* ist ein Industriestandard für die Energieverwaltung von Computern. Der Standard erlaubt es, Geräte an- und

abzuschalten, je nachdem ob diese Geräte gerade benutzt werden. Außerdem erlaubt rd der Standard, die Prozessorgeschwindigkeit an den Grad der Prozessornutzung anzupassen, abhängig von den Anwendungs-Anforderungen. Die Energiezustände, in denen sich ein Prozessor befinden kann, sind in sogenannte *C-States* eingeteilt. Der ACPI-Standard [ACP] definiert die vier C-States (*Processor Power States*) [ACP, S. 307] C0, C1, C2 und C3, welche einen unterschiedlichen Grad an Energiesparmaßnahmen beschreiben. C-States mit einer höheren Nummer erfordern drastischere Sparmechanismen und je höher die Nummer hinter dem C ist, desto länger dauert die Rückkehr in den aktiven Zustand. Prozessoren, welche den ACPI-Standard unterstützen, implementieren prozessorspezifische Zustände, welche die Vorgaben der vier C-States erfüllen und diesen entsprechen. Die meisten aktuellen Prozessoren implementieren mindestens den C0- und C1-Zustand.

- **C0** - In diesem Zustand in der Prozessor aktiv und es werden Instruktionen ausgeführt.
- **C1** - Dies ist der erste Energiesparzustand, in den ein Prozessor häufig wechselt. In diesem Zustand werden Einheiten des Prozessors deaktiviert. Der L1- und L2-Cache des Prozessors wird aufrechterhalten und durch einen Interrupt kann er schnell wieder in den C0-Zustand wechseln.
- **C2** - Der nächsthöhere Schlafzustand erhöht das Maß an Energieeinsparungen. Der Kontext der L1-Caches wird nicht aufrechterhalten und die Rückkehrzeit in einen aktiveren Zustand ist erhöht.
- **C3** - Der C3-Schlafzustand aktiviert weitere Energiesparmechanismen als C2. Der Prozessor generiert kein Taktsignal mehr und der Kontext der Prozessor-Caches wird nicht aufrechterhalten.

In Abbildung 6.38 sind die vier C-Zustände und deren Übergange dargestellt. Der obere Zustand stellt den C0-Zustand dar. Dies ist der aktive Zustand des Prozessors, in dem Instruktionen ausgeführt werden. Der C0-Zustand ist zusätzlich durch *Performance States* (siehe Abschnitt 6.4.6) unterteilt, in welchen der Prozessortakt und die Versorgungsspannung verringert werden können. Durch einen Throttling-Prozess kann die Leistung des Prozessors zusätzlich beeinflusst werden.

Aus dem C0-Zustand wechselt der Prozessor nach einem HLT-Befehl (Abkürzung für HALT) in den C1-Zustand. Nach einem Interrupt wird er wieder aus dem Haltezustand C1 geholt. Für die weiteren Energiesparzustände C2 und C3 gibt es ebenfalls eigene Befehle.

Um einen geeigneten C-Zielzustand auszuwählen, bewertet das Betriebssystem die Prozessornutzung der letzten Zeitscheibe. In Abbildung 6.39 beträgt die Dauer einer Zeitscheibe 100 Millisekunden. In der ersten Zeitscheibe werden HLT-Befehle (halt) an den Prozessor gesendet, worauf dieser in den Energiesparzustand C1 wechselt.

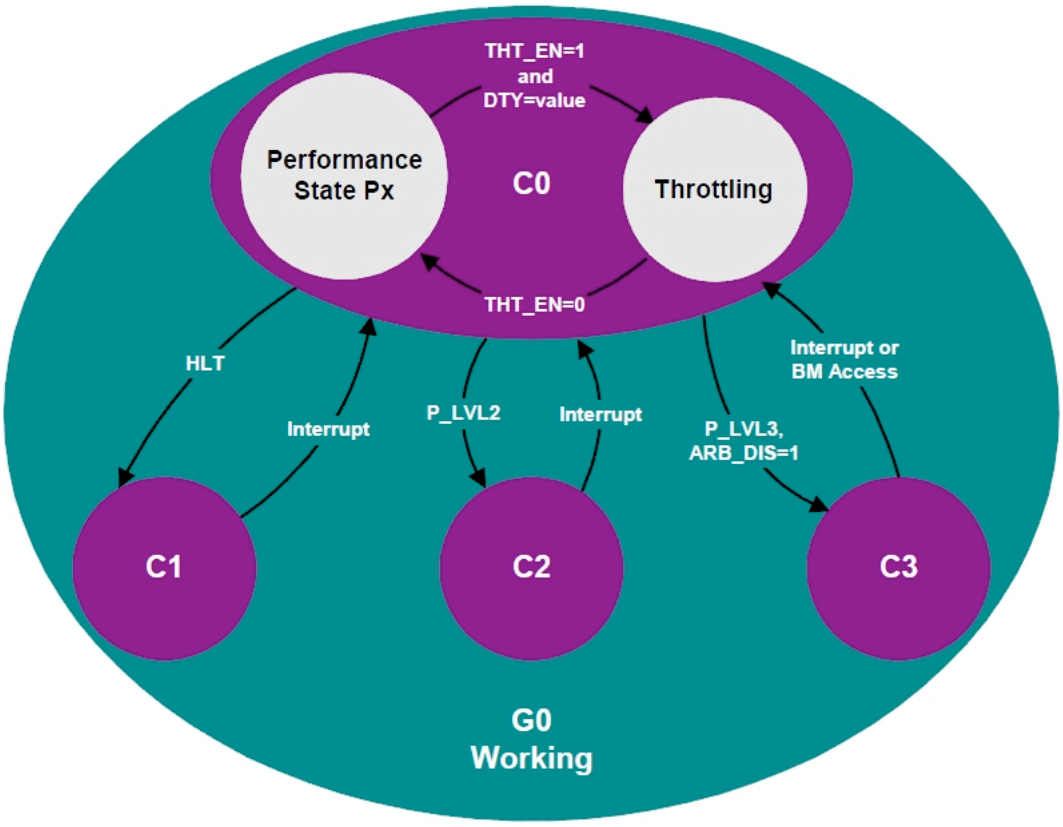



Abbildung 6.38: Energiesparzustände eines Prozessors nach ACPI [ACP].

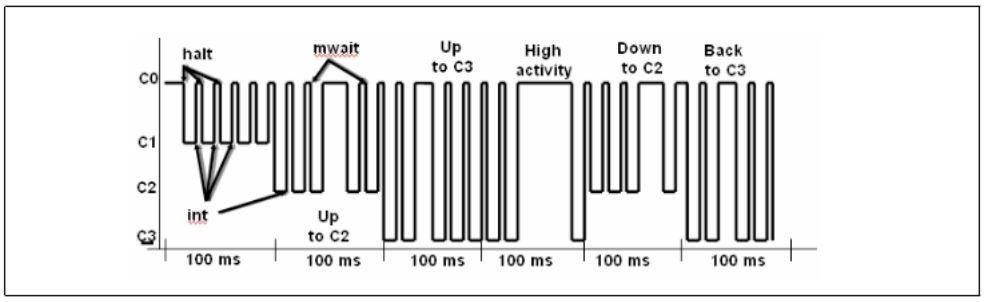



Abbildung 6.39: Wahl eines C-Zielzustandes nach Analyse der aktuellen Zeitscheibe [Intb].

Nach einem Interrupt (*int*) wechselt er wieder in den aktiven Zustand C0. Da die Prozessornutzung in dieser ersten Zeitscheibe zu gering war, wechselt der Prozessor in der zweiten Zeitscheibe in den nächsthöheren Schlafzustand C2. Zu beachten ist, dass ein Prozessor auch in jeder Schlafphase durch einen Interrupt jederzeit aufgeweckt werden kann, um Instruktionen auszuführen. Die Aufweckzeiten reichen je nach Tiefe des Schlafzustandes von 1 $\mu s$ bis 250 $\mu s$ [Intb].

### 6.4.3 Reduktion der Kernspannung bei niedriger Prozessorauslastung (C1E-State)

Die *Enhanced Halt State*-Funktion (kurz *C1E* genannt) [Intc, S. 86] ist eine Erweiterung des C1-Zustands von Intel-Prozessoren ab der Pentium 4-Generation, welche es dem Betriebssystem erlaubt, den Multiplikator und die Spannung des Prozessors im Betrieb zu verändern.
In einer Leerlaufphase kann das Betriebssystem einen erweiterten HLT-Befehl an den Prozessor senden, welcher bewirkt, dass der Multiplikator und zusätzlich die Versorgungsspannung des Prozessors reduziert werden. Bei reduziertem Multiplikator und reduzierter Versorgungsspannung befindet sich der Prozessor in dem C1E-Zustand, in welchem die Prozessor-Leistungsaufnahme reduziert ist. Die *Enhanced Halt State*-Funktion lässt sich im BIOS vieler Mainboards aktivieren. Einmal im BIOS aktiviert, kann das Betriebssystem dann in Leerlaufphasen in den C1E-Zustand wechseln. Die meisten Betriebssysteme wechseln ständig in den C1E-Zustand, wenn das System nicht unter voller Last steht [Intd].

In dem Rechner, welcher für unsere Tests verwendeten, ist ein Pentium 4-Prozessor mit einem Takt von 3,20 GHz verbaut. Bei einem *Front Side Bus* (FSB) von 200 MHz und einem Multiplikator von 16 ergibt sich dieser Prozessortakt von 3,20 GHz (FSB $\times$ Prozessor-Multiplikator = Prozessortakt). Die Betriebsspannung bei diesem Takt ist 1,3875 Volt. Wird bei aktivierter *Enhanced Halt State*-Funktion ein Halt-Befehl an den Prozessor gesendet, wird der Multiplikator bei unserem Prozessor von 16 auf 14 reduziert. Dies ergibt einen Prozessortakt von 2,80 GHz (200 MHz $\times$ 14 = 2,80 GHz) bei einer zusätzlich auf 1,2625 Volt reduzierten Betriebsspannung.

Allerdings unterscheidet die *Enhanced Halt State*-Funktion nur zwischen zwei Möglichkeiten: geringe Prozessoraktivität (2,80 GHz) und keine geringe Prozessoraktivität (3,20 GHz). Um den Energieverbrauch eines Prozessors noch besser an die tatsächliche Prozessornutzung anzupassen, wurde bei neueren Prozessoren der C0-Zustand in weitere Zustände, die sogenannten P-States, aufgeteilt.

### 6.4.4 Messergebnisse bei aktiviertem C1E

Um die Auswirkungen der *Enhanced Halt State*-Funktion zu bewerten, wurden die 22 TPC-H-Abfragen bei aktivierter C1E-Funktion durchgeführt. In Tabelle 6.18 sind die Laufzeit und der Energieverbrauch für jede Abfrage angegeben. Die Messergebnisse bei deaktivierter C1E-Funktion, welche als Vergleichsdaten dienen, wurden bereits in Abschnitt 6.1.3 in Tabelle 6.4 angegeben.

### 6.4.5 Analyse der Messergebnisse bei aktiviertem C1E

Für jede Abfrage ist die prozentuale Veränderung der Laufzeit und des Energieverbrauchs in Tabelle 6.18 nach Aktivierung der C1E-Funktion angegeben (die Prozentwerte in den Klammern). Der Tabelle ist direkt zu entnehmen, dass die Laufzeiten der

| | TPC-H Benchmark (mit C1E) | | |
|---|---|---|---|
| Nr. | Laufzeit [s] | Joule | Joule/s |
| 1 | Timeout | N/A | N/A |
| 2 | 60,51 (+1,39 %) | 6121,45 (-6,63 %) | 101,17 (-7,91 %) |
| 3 | 1586,00 (+1,39 %) | 172731,20 (-4,42 %) | 108,91 (-5,73 %) |
| 4 | 256,45 (+3,49 %) | 29730,73 (-3,75 %) | 115,93 (-7,00 %) |
| 5 | 660,57 (+3,76 %) | 65792,18 (-4,85 %) | 99,60 (-8,30 %) |
| 6 | 501,56 (+4,60 %) | 64681,31 (+0,43 %) | 128,96 (-3,98 %) |
| 7 | 2702,16 (+2,03 %) | 306209,18 (-2,89 %) | 113,32 (-4,82 %) |
| 8 | 2684,87 (+1,17 %) | 296050,37 (-4,04 %) | 110,27 (-5,14 %) |
| 9 | Timeout | N/A | N/A |
| 10 | 841,89 (+0,64 %) | 80028,10 (-8,41 %) | 95,06 (-8,99 %) |
| 11 | 383,52 (+2,92 %) | 51847,80 (+2,01 %) | 135,19 (-0,88 %) |
| 12 | 571,66 (+2,88 %) | 73733,18 (-0,92 %) | 128,98 (-3,69 %) |
| 13 | 604,52 (+1,52 %) | 82674,06 (+0,63 %) | 136,76 (-0,88 %) |
| 14 | 1104,32 (+0,54 %) | 116438,30 (-6,66 %) | 105,44 (-7,16 %) |
| 15 | 1067,92 (+4,48 %) | 137831,10 (+0,59 %) | 129,07 (-3,72 %) |
| 16 | 104,56 (+5,95 %) | 13112,15 (+1,06 %) | 125,40 (-4,61 %) |
| 17 | Timeout | N/A | N/A |
| 18 | 2544,82 (+1,98 %) | 344965,73 (+0,97 %) | 135,56 (-0,98 %) |
| 19 | 3025,37 (+2,45 %) | 315809,86 (-4,83 %) | 104,39 (-7,10 %) |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.18: Messergebnisse der 22 Abfragen des TPC-H Benchmarks auf der 10 GB-Datenbank bei aktiviertem C1E. Die Prozentangaben in Klammern geben den Unterschied zu den Messergebnissen ohne C1E an.

einzelnen Abfragen um ungefähr 0,5 % bis 6 % zugenommen haben. Dem entgegen steht bei den meisten Abfragen eine Reduktion des Energieverbrauchs von ungefähr 1 % bis 8 % Prozent. Bei einer geringen Anzahl von Abfragen ist keine Reduktion des Energieverbrauchs festzustellen.

### Der energiesparende Effekt der *Enhanced Halt State*-Funktion am Beispiel von Abfrage 8

Bei Abfrage 8 beträgt die Reduktion des Energieverbrauchs 4,04 %. Anhand dieser Abfrage lässt sich besonders gut erkennen, worauf der energiesparende Effekt beruht, da er in den ersten 500 Sekunden der Ausführungszeit isoliert in Erscheinung tritt. In Abbildung 6.40 und Abbildung 6.41 ist die Leistungsaufnahme des Servers (grün)

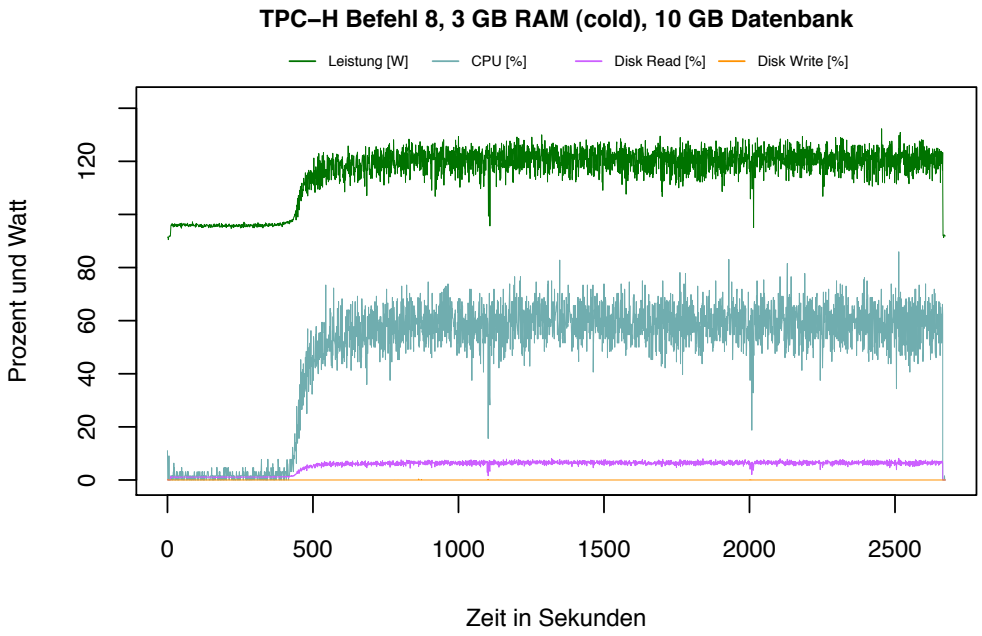



Abbildung 6.40: TPC-H Abfrage 8 (ohne C1E).

und die Prozessor- (blau) und I/O-Aktivität (violett und orange) während der Ausführung der Abfrage grafisch dargestellt. Die Zeit, welche zur Ausführung des Befehls 8 benötigt wird, lässt sich hinsichtlich der Prozessoraktivität in beiden Abbildungen in zwei Phasen einteilen:

- **Phase 1:** Eine Phase niedriger Prozessorauslastung von Anfang bis ungefähr Sekunde 450.
- **Phase 2:** Eine Phase durchschnittlicher Prozessorauslastung ab ungefähr Sekunde 450 bis Ende.

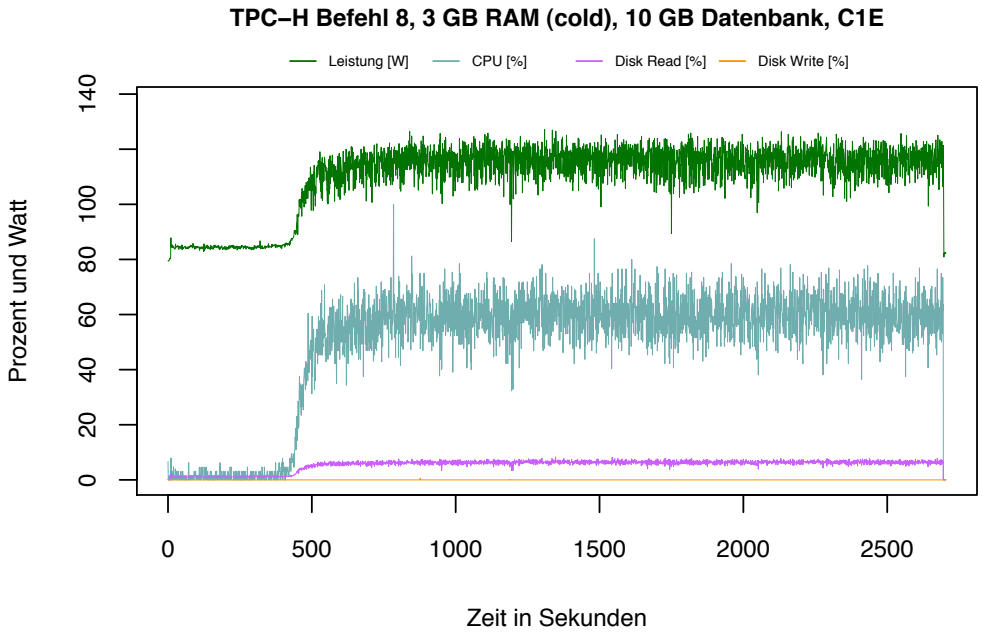



Abbildung 6.41: TPC-H Abfrage 8 (mit C1E).

Eine Auswertung der Logdateien unserer Messungen ergab, dass die durchschnittliche Leistungsaufnahme des Servers in der ersten Phase 96 Watt (ohne C1E) beziehungsweise 86 Watt (mit C1E) beträgt. Durch Aktivierung der C1E-Funktion kann der Prozessor aufgrund der geringen Auslastung in dieser Phase fast durchgängig in den C1E-Zustand wechseln, in welchem er ungefähr 10 Watt weniger benötigt. Dieser energiesparende Effekt ist durch einen Vergleich der Abbildungen 6.40 und 6.41 auch optisch besonders deutlich zu erkennen.

Aber auch in der zweiten Phase ist der energiesparende Effekt der C1E-Funktion in den Logdateien zu den Messungen feststellbar, da der Prozessor auch in dieser Zeit immer wieder kurzzeitig in den energiesparenden C1E-Zustand wechselt. Die durchschnittliche Leistungsaufnahme des Servers in dieser Phase beträgt 120 Watt (ohne C1E) beziehungsweise 115 Watt (mit C1E). Die häufigen und kurzzeitigen Wechsel in den energiesparenden C1E-Zustand, auch in einer Phase, in welcher der Prozessor zu immerhin 60 % ausgelastet ist, führen zu einem niedrigeren Energieverbrauch.

Die Kombination der Energieeinsparungen der beiden Phasen führt letztlich zu einer Reduktion von 4,04 % des Gesamtenergiebedarfs für die Ausführung dieser Abfrage. Die Latenz aufgrund der ständigen Wechsel des Prozessors aus dem C0-Zustand in den C1E-Zustand und umgekehrt führte zu einer Verlängerung der Laufzeit von 1,17 %.

### Zusammenhang von Prozessorauslastung und Energieverbrauch

Es gibt Abfragen, bei denen statt einer Reduktion eine Zunahme des Energieverbrauchs festzustellen ist. Die Ausnahmen bilden die Abfragen 6, 11, 13, 15, 16 und 18. Die prozentuale Zunahme des Energieverbrauchs (0,5 % bis 2 %) liegt allerdings bei jeder dieser Abfragen unter dem Wert für die prozentuale Zunahme der Laufzeit (2 % bis 6 %). Somit tritt auch hier der energiesparende Effekt des C1E in Erscheinung, aber dieser Effekt kann den aufgrund der längeren Laufzeit erhöhten Energieverbrauch dieser Abfragen nicht völlig ausgleichen. Diese sechs Ausnahmen haben eine hohe Prozessorauslastung über die gesamte Laufzeit gemeinsam. Dies deutet auf einen Zusammenhang zwischen der Prozessorauslastung und der Auswirkung von C1E auf die Laufzeit und den Energieverbrauch hin.

Der Zusammenhang der durchschnittlichen Prozessorauslastung (bezogen auf die Ausführungszeit einer Abfrage) und der Veränderung der Laufzeit und des Energieverbrauchs nach Aktivierung der C1E-Funktion ist in Tabelle 6.19 angegeben. Diese Tabelle ist nicht nach der Nummer der TPC-H-Abfragen, sondern nach der Spalte „Durchschnittliche Prozessorauslastung“ aufsteigend sortiert. Durch die Sortierung

| Nr. | Durchschnittliche Prozessorauslastung [%] | Laufzeit-Veränderung [%] | Joule-Veränderung [%] |
|---|---|---|---|
| 10 | 24,54 | +0,64 | -8,41 |
| 5 | 33,79 | +3,76 | -4,85 |
| 2 | 36,28 | +1,39 | -6,63 |
| 19 | 40,35 | +2,45 | -4,83 |
| 14 | 41,96 | +0,54 | -6,66 |
| 8 | 49,37 | +1,17 | -4,04 |
| 3 | 49,52 | +1,39 | -4,42 |
| 7 | 56,54 | +2,03 | -2,89 |
| 4 | 68,37 | +3,49 | -3,75 |
| 16 | 83,21 | +5,95 | +1,06 |
| 6 | 87,25 | +4,60 | +0,43 |
| 12 | 87,40 | +2,88 | -0,92 |
| 15 | 87,85 | +4,48 | +0,59 |
| 18 | 95,83 | +1,98 | +0,97 |
| 11 | 96,19 | +2,92 | +2,01 |
| 13 | 96,41 | +1,52 | +0,63 |

Tabelle 6.19: Auswirkung der durchschnittlichen Prozessorauslastung auf die Änderung der Laufzeit und des Energieverbrauchs nach Aktivierung der C1E-Funktion. Die Tabelle ist nach der Spalte „Durchschnittliche Prozessorauslastung“ aufsteigend sortiert.

ist die Auswirkung der durchschnittlichen Prozessorauslastung auf die Änderung der

Laufzeit und des Energieverbrauchs nach Aktivierung der C1E-Funktion besser zu erkennen. Die Abfragen, bei denen eine Zeitüberschreitung auftrat, sind in dieser Tabelle nicht enthalten.

Die Abfragen, bei denen durch C1E eine Zunahme des Energieverbrauchs festgestellt wurde, weisen die höchste Prozessorauslastung auf. Aufgrund der hohen Prozessorauslastung kann der Prozessor nur für kurze Zeiten in den C1E-Zustand wechseln. Bei Abfrage 12 ist trotz hoher Prozessorauslastung noch eine Reduktion des Energieverbrauchs festzustellen, aber diese Reduktion ist prozentual die kleinste verglichen mit den Werten der anderen Abfragen. Die größte Reduktion des Energieverbrauchs ist bei der Abfrage mit der geringsten Prozessorauslastung erkennbar. Damit ergibt sich folgende Abhängigkeit: Je niedriger die durchschnittliche Prozessorauslastung ist, desto größer ist die Reduktion der benötigten Energie für die Ausführung einer Abfrage. Die *Enhanced Halt State*-Funktion ist für I/O-intensive, aber nicht für CPU-intensive Abfragen empfehlenswert.

Um eine funktionale Beschreibung der Daten in Tabelle 6.19 zu erhalten, wurde eine lineare Regressionsanalyse (nach der Methode der kleinsten Quadrate) durchgeführt. Die daraus resultierende Funktion $f_{Laufzeit}(x)$ für die Berechnung der prozentualen Veränderung der Laufzeit anhand der durchschnittlichen Prozessorauslastung (Variable $x$) lautet:

$$f_{Laufzeit}(x) = 0,0292x + 0,6825 \tag{6.1}$$

Für die prozentuale Veränderung des Energieverbrauchs $f_{Joule}(x)$ gilt:

$$f_{Joule}(x) = 0,1231x - 10,568 \tag{6.2}$$

Zu beachten ist, dass beide Funktionen nur für den Wertebereich $20 \leq x < 100$ gute Ergebnisse berechnen, da in diesem Wertebereich die Datensätze, auf denen die Approximation basiert, der Tabelle 6.19 liegen. Abbildung 6.42 stellt den approximierten Zusammenhang zwischen der durchschnittlichen prozentualen Prozessorauslastung und der prozentualen Veränderung der Laufzeit bzw. der prozentualen Veränderung des Energieverbrauchs grafisch da. Anhand dieser Funktionen können die Laufzeit und der Energieverbrauch anhand der durchschnittlichen Prozessorauslastung approximiert werden, und ein Datenbank-Administrator kann entscheiden, ob eine Aktivierung der *Enhanced Halt State*-Funktion für ihn lohnenswert ist. Liegt die durchschnittliche Prozessorauslastung einer Abfrage oder eines Datenbank-Workloads unter 86 %, dann kann mit C1E Energie gespart werden.

### Betrachtung der Gesamtlaufzeit und des Gesamt-Energieverbrauchs

Der energiesparende Effekt ist bei den meisten TPC-H-Abfragen festzustellen und ein Vergleich der Gesamtlaufzeit und des Gesamt-Energieverbrauchs des TPC-H Benchmarks mit C1E und ohne C1E (siehe Tabelle 6.20) zeigt, dass mit C1E die Energiekosten des TPC-H Benchmarks insgesamt gesenkt werden konnten. Die Reduktion des

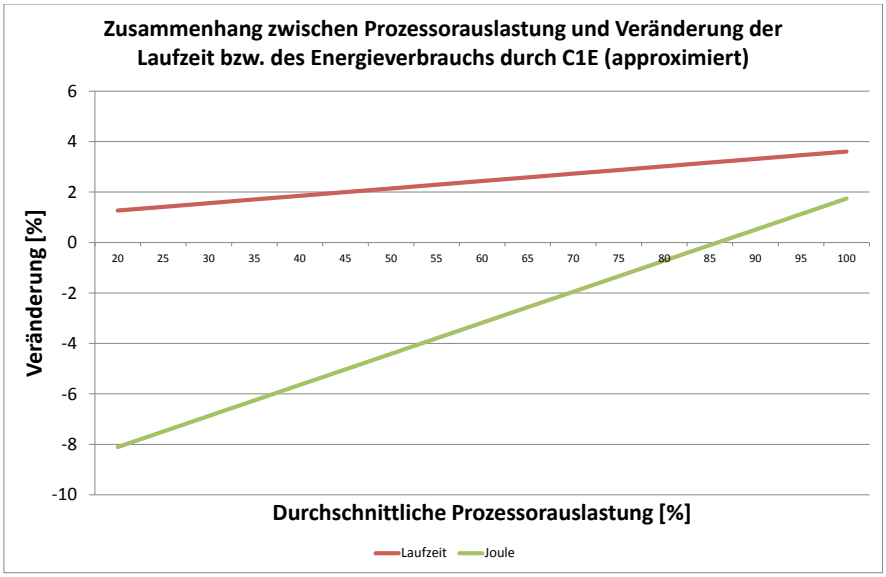



Abbildung 6.42: Approximierter Zusammenhang zwischen der durchschnittlichen Prozessorauslastung und der Veränderung der Laufzeit bzw. des Energieverbrauchs durch die C1E-Funktion.

Energieverbrauchs des TPC-H Benchmarks beträgt 2,74 % und ist damit etwas größer als die Zunahme der Gesamtlaufzeit um 2,09 %. Damit ist es im Sinne einer Verbes-

| | Laufzeit [s] | Joule |
|---|---|---|
| TPC-H | 18318,28 | 2218520,25 |
| TPC-H (mit C1E) | 18700,68 (+2,09 %) | 2157756,69 (-2,74 %) |

Tabelle 6.20: Gesamtlaufzeit und Gesamt-Energieverbrauch des TPC-H Benchmarks auf der 10 GB-Datenbank mit und ohne aktivierter C1E-Funktion.

serung der Energieeffizienz einer Datenbank für den TPC-H Benchmark sinnvoll, die C1E-Funktion zu aktivieren. Dies bedeutet, dass eine dynamische Anpassung der Spannung und der Frequenz eines Prozessors an die Auslastung der Datenbank einen energiesparenden Effekt hat. Dies gilt schon bei einer Unterscheidung von nur zwei unterschiedlichen Prozessor-Zuständen, wie in unserem Fall (3,20 GHz bei 1,3875 Volt Kernspannung und 2,80 GHz bei 1,2625 Volt Kernspannung).

### 6.4.6 Anpassung der Kernspannung und der Frequenz an die Prozessorauslastung (P-States)

Als Erweiterung und zur noch besseren Anpassung des Taktes und der Spannung des Prozessors an die tatsächliche Prozessoraktivität wird im ACPI-Standard eine zusätzliche Unterteilung des C0-Zustands (der aktive Prozessorzustand) beschrieben. Der C0-Zustand wird in sogenannte *P-States (Processor Performance States)* [ACP, S. 43] $P_1, P_2, ..., Pn$ unterteilt, welche die Granularität des C0-Zustandes erhöhen. Anders als bei C-States gibt es keine fest definierte Anzahl von P-States. Jeder Hersteller legt für einen Prozessor-Typ eine geeignete Anzahl unterschiedlicher P-States fest. Ein Prozessor im P1-Zustand nutzt sein vollständiges Rechenpotenzial und hat die höchste Leistungsaufnahme. In jedem Nachfolgezustand werden die Frequenz und die Versorgungsspannung des Prozessors sukzessive verringert. In Tabelle 6.21 sind exemplarisch die P-States mit den unterschiedlichen Taktraten und Spannungen für einen modernen Prozessor angegeben. Ähnlich wie bei den C-Zuständen

| Frequenz | Spannung |
|---|---|
| 1.6 GHz | 1.484 V |
| 1,4 GHz | 1.420 V |
| 1,2 GHz | 1.276 V |
| 1,0 GHz | 1.164 V |
| 800 MHz | 1.036 V |
| 600 MHz | 0.956 V |

Tabelle 6.21: Performance States des Pentium M 1,6 GHz-Prozessors [Inta, S. 4].

wird auch der jeweils nächste P-Zielzustand vom Betriebssystem nach einer Analyse der Prozessornutzungs-Historie festgelegt [Intb]. Abbildung 6.43 zeigt, wie nach einer Veränderung des Grades der Prozessornutzung nach einer kurzen Verzögerung der P-Zustand durch das Betriebssystem gewechselt wird.

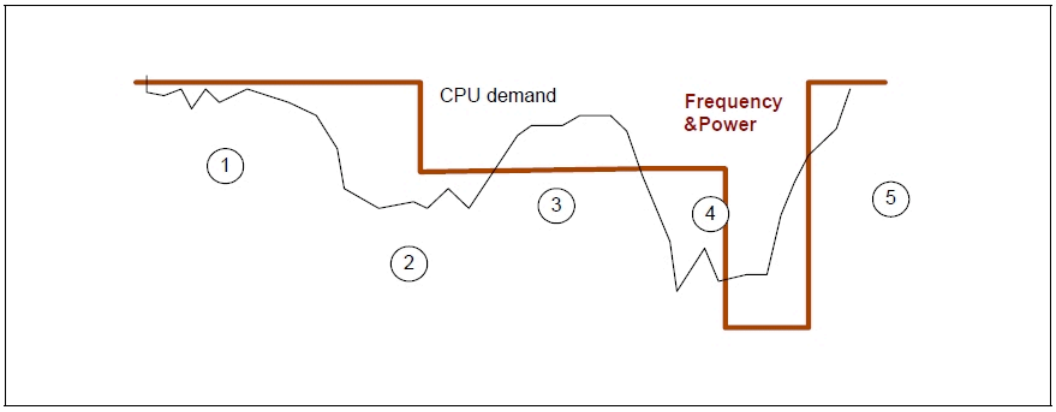



Abbildung 6.43: Leistungs-Historie und P-State Transitionen [Intb].

Intel nennt seine Implementierung dieser P-States in modernen Prozessoren *Enhanced Intel SpeedStep Technology* (EIST). Prozessoren der Pentium 4-Generation, also auch der Prozessor, welcher in unserem Server verbaut ist, besitzen dieses Feature jedoch nicht; daher konnten dazu keine Tests durchgeführt werden.

Die Vorzüge der P-States hinsichtlich des Energieverbrauchs des Prozessors nutzten Lang und Patel [Lan09]. Sie stellten 2009 eine Methode vor, mit welcher der Nutzungsgrad eines DBMS aufm passenden P-State abgebildet werden kann. In Phasen, in denen das System nicht vollständig ausgelastet ist, kann das Betriebssystem oder auch das DBMS den Prozessor dazu veranlassen, in einen anderen P-Zustand mit einer niedrigeren Spannung und Frequenz zu wechseln. Diese Methode nennen Lang und Patel PVC ***P**rocessor **V**oltage/Frequency **C**ontrol*. In ihren Experimenten konnten sie unter Verwendung eines nicht weiter genannten kommerziellen DBMS den Energieverbrauch um bis zu 49 % senken, während gleichzeitig die Antwortzeit (*engl. Responsetime*) der Abfragen nur um 3 % zunahm. Bei Verwendung von MySQL konnten sie den Energieverbrauch um 20 % senken, bei einer Zunahme der Antwortzeit von 6 %. Als Workload diente ihnen Abfrage 5 des TPC-H Benchmarks.

Obwohl der in unserem Server verbaute Prozessor keine unterschiedlichen P-States bietet, konnte die Frequenz des Prozessors gedrosselt werden, ohne jedoch gleichzeitig die Kernspannung zu senken. Die Auswirkung einer statischen Drosselung der Prozessorfrequenz ohne gleichzeitige Reduktion der Spannung wird in den nächsten beiden Abschnitten untersucht.

### 6.4.7 Prozessor-Throttling

Als Prozessor-Throttling wird das Auslassen von Takten eines Prozessors bezeichnet. Obwohl die Taktfrequenz des Prozessors gleich bleibt, so stehen dem Prozessor nach dem Drosseln (engl. *to throttle*) weniger Takte für das Ausführen von Instruktionen zur Verfügung. Bei einem Throttling von beispielsweise 50 % steht dem Prozessor nur jeder zweite Takt zum Ausführen von Instruktionen zur Verfügung. Für unsere 3,20 GHz-CPU bedeutet dies, dass die Taktfrequenz zwar bei 3,20 GHz bleibt, aber dem Prozessor effektiv nur die Hälfte, also 1,60 GHz, zur Verfügung steht. Die restliche Zeit verbringt der Prozessor in einer Leerlaufphase, in der er weniger Energie verbraucht. Wird parallel zum Throttling zusätzlich die Spannung des Prozessors gesenkt, dann ist der Energiespareffekt höher.

Die von uns verwendete Hardware lies jedoch nur Frequenz-Änderungen zu. Eine mit der Drosselung der Frequenz einhergehende Senkung der Kernspannung verspricht weiteres Energiesparpotenzial. Allerdings bringt allein das Drosseln der Prozessor-Frequenz eine Energieeinsparung mit sich. Der Energieverbrauch skaliert beispielsweise bei einem *Intel 80200*-Prozessor proportional zu der Prozessor-Frequenz [Mul02, S. 14].

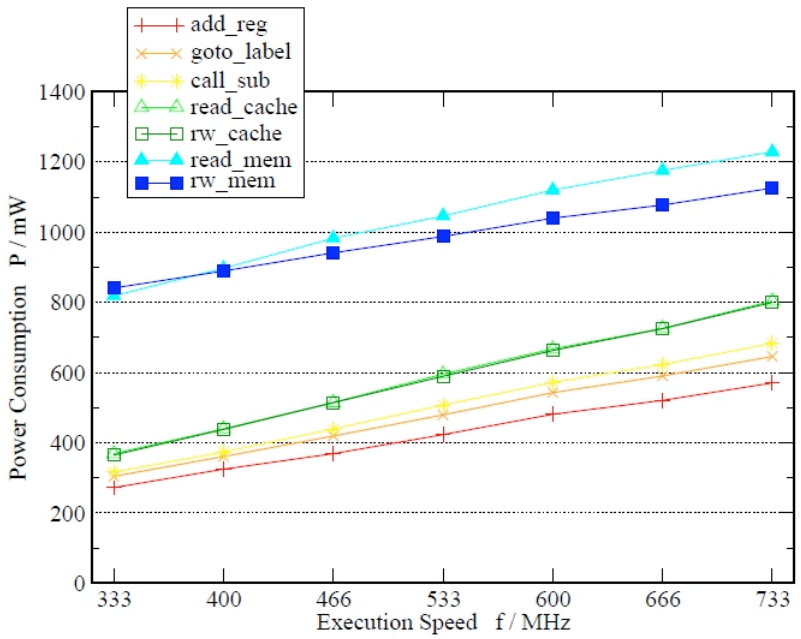



Abbildung 6.44: Verhältnis von Prozessorfrequenz und Energieverbrauch [Mul02].

Abbildung 6.44 visualisiert das Verhältnis von Frequenz und Energieverbrauch bei identischer Kernspannung. Bei dem Wert für den Energieverbrauch handelt sich um den durchschnittlichen Energieverbrauch. Für eine Gesamtbewertung der Auswirkung des Throttlings muss allerdings auch die Veränderung der Performance, messbar zum

Beispiel anhand der Laufzeit eines Abfrage, berücksichtigt werden.

### 6.4.8 Messergebnisse bei aktivem Prozessor-Throttling

In den Tabellen 6.22, 6.23 und 6.24 sind die Ergebnisse des TPC-H Benchmarks mit unterschiedlichen *Throttling*-Stufen angegeben. Die Tests wurden durchgeführt mit aktivierter C1E-Funktion und einer Prozessor-Drosselung auf 75 % (2,40 GHz), 50 % (1,60 GHz) und 25 % (800 MHz) der maximalen Prozessor-Taktfrequenz. Gedrosselt wurde die Frequenz mit dem Programm *RightMark CPU Clock Utility*. Die Kernspannung betrug bei allen Tests 1,3875 Volt und wurde nicht verändert. Die Prozentangaben in Klammern geben den Unterschied zu den gemessenen Werten ohne Prozessor-Throttling an (siehe Abschnitt 6.4.4, Tabelle 6.18) und ermöglichen so einen direkten Vergleich.

| | TPC-H Benchmark (Throttling-Stufe 75 %) | | |
|---|---|---|---|
| Nr. | Laufzeit [s] | Joule | Joule/s |
| 1 | Timeout | N/A | N/A |
| 2 | 60,91 (+0,67 %) | 6262,27 (+2,30 %) | 104,52 (+3,32 %) |
| 3 | 1854,45 (+16,93 %) | 198466,84 (+14,90 %) | 107,02 (-1,73 %) |
| 4 | 310,55 (+21,10 %) | 35473,91 (+19,32 %) | 114,23 (-1,47 %) |
| 5 | 730,24 (+10,55 %) | 72603,39 (+10,35 %) | 99,42 (-0,18 %) |
| 6 | 659,65 (+31,52 %) | 80057,39 (+23,77 %) | 121,36 (-5,89 %) |
| 7 | 3294,56 (+21,92 %) | 362995,80 (+18,55 %) | 110,18 (-2,77 %) |
| 8 | 3238,75 (+20,63 %) | 348335,49 (+17,66 %) | 107,55 (-2,46 %) |
| 9 | Timeout | N/A | N/A |
| 10 | 897,91 (+6,65 %) | 85811,68 (+7,23 %) | 95,57 (+0,54 %) |
| 11 | 516,52 (+34,68 %) | 64583,68 (+24,56 %) | 125,04 (-7,51 %) |
| 12 | 763,54 (+33,57 %) | 92968,06 (+26,09 %) | 121,76 (-5,60 %) |
| 13 | 815,01 (+34,82 %) | 102999,88 (+24,59 %) | 126,38 (-7,59 %) |
| 14 | 1265,68 (+14,61 %) | 132847,63 (+14,09 %) | 104,96 (-0,45 %) |
| 15 | 1409,56 (+31,99 %) | 171632,90 (+24,52 %) | 121,76 (-5,66 %) |
| 16 | 133,74 (+27,90 %) | 15998,09 (+22,01 %) | 119,62 (-4,61 %) |
| 17 | Timeout | N/A | N/A |
| 18 | 3440,41 (+35,19 %) | 432494,38 (+25,37 %) | 125,71 (-7,26 %) |
| 19 | 3440,51 (+13,72 %) | 356439,22 (+12,87 %) | 103,600694 (-0,75 %) |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.22: Messergebnisse der 22 Abfragen des TPC-H Benchmarks mit C1E und einer Frequenz-Drosslung auf 75 % der maximalen Frequenz. Die Prozentangaben in Klammern geben den Unterschied zu den Messergebnissen ohne Throttling an.

| | TPC-H Benchmark (Throttling-Stufe 50 %) | | |
|---|---|---|---|
| Nr. | Laufzeit [s] | Joule | Joule/s |
| 1 | Timeout | N/A | N/A |
| 2 | 67,59 (+11,70 %) | 7080,16 (+15,66 %) | 104,75 (+3,54 %) |
| 3 | 2603,95 (+64,18 %) | 269120,75 (+55,80 %) | 103,35 (-5,10 %) |
| 4 | 419,87 (+63,73 %) | 46184,74 (+55,34 %) | 110,00 (-5,12 %) |
| 5 | 880,28 (+33,26 %) | 87319,13 (+32,72 %) | 99,20 (-0,41 %) |
| 6 | 938,54 (+87,13 %) | 106867,74 (+65,22 %) | 113,87 (-11,71 %) |
| 7 | Timeout | N/A | N/A |
| 8 | Timeout | N/A | N/A |
| 9 | Timeout | N/A | N/A |
| 10 | 1162,77 (+38,11 %) | 110935,99 (+38,62 %) | 95,41 (+0,37 %) |
| 11 | 760,55 (+98,31 %) | 87977,76 (+69,68 %) | 115,68 (-14,43 %) |
| 12 | 1091,98 (+91,02 %) | 124320,24 (+68,61 %) | 113,85 (-11,73 %) |
| 13 | 1233,54 (+104,05 %) | 142908,25 (+72,86 %) | 115,85 (-15,29 %) |
| 14 | 1734,91 (+57,10 %) | 176332,73 (+51,44 %) | 101,64 (-3,60 %) |
| 15 | 2022,79 (+89,41 %) | 230250,79 (+67,05 %) | 113,83 (-11,81 %) |
| 16 | 190,22 (+81,92 %) | 21317,22 (+62,58 %) | 112,07 (-10,63 %) |
| 17 | Timeout | N/A | N/A |
| 18 | Timeout | N/A | N/A |
| 19 | Timeout | N/A | N/A |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.23: Messergebnisse der 22 Abfragen des TPC-H Benchmarks mit C1E und einer Frequenz-Drosslung auf 50 % der maximalen Frequenz. Die Prozentangaben in Klammern geben den Unterschied zu den Messergebnissen ohne Throttling an.

| | TPC-H Benchmark (Throttling-Stufe 25 %) | | |
|---|---|---|---|
| Nr. | Laufzeit [s] | Joule | Joule/s |
| 1 | Timeout | N/A | N/A |
| 2 | 101,72 (+68,11 %) | 10922,61 (+78,43 %) | 107,38 (+6,14 %) |
| 3 | Timeout | N/A | N/A |
| 4 | 714,84 (+178,75 %) | 77953,09 (+162,20 %) | 109,05 (-5,94 %) |
| 5 | 1237,00 (+87,26 %) | 126722,77 (+92,61 %) | 102,44 (+2,86 %) |
| 6 | 1650,15 (+229,01 %) | 185289,34 (+186,47 %) | 112,29 (-12,93 %) |
| 7 | Timeout | N/A | N/A |
| 8 | Timeout | N/A | N/A |
| 9 | Timeout | N/A | N/A |
| 10 | 1391,81 (+65,32 %) | 138942,27 (+73,62 %) | 99,83 (+5,02 %) |
| 11 | 1393,50 (+263,34 %) | 158149,30 (+205,03 %) | 113,49 (-16,05 %) |
| 12 | 1946,96 (+240,58 %) | 218691,27 (+196,60 %) | 112,32 (-12,91 %) |
| 13 | 2215,00 (+266,41 %) | 252301,24 (+205,18 %) | 113,91 (-16,71 %) |
| 14 | 2327,38 (+110,75 %) | 246713,13 (+111,88 %) | 106,00 (+0,54 %) |
| 15 | 3594,16 (+236,56 %) | 403238,83 (+192,56 %) | 112,19 (-13,07 %) |
| 16 | 332,53 (+218,02 %) | 36877,64 (+181,25 %) | 110,90 (-11,56 %) |
| 17 | Timeout | N/A | N/A |
| 18 | Timeout | N/A | N/A |
| 19 | Timeout | N/A | N/A |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |

Tabelle 6.24: Messergebnisse der 22 Abfragen des TPC-H Benchmarks mit C1E und einer Frequenz-Drosslung auf 25 % der maximalen Frequenz. Die Prozentangaben in Klammern geben den Unterschied zu den Messergebnissen ohne Throttling an.

### 6.4.9 Analyse der Messergebnisse bei aktivem Prozessor-Throttling

Die Messdaten des vorherigen Abschnitts haben gezeigt, dass ein gedrosselter Prozessor bei den meisten Abfragen eine niedrigere durchschnittliche Leistungsaufnahme hat und zugleich die Ausführungszeit einer Abfrage zunimmt. Als Vergleichsdaten zu den Tests zum Prozessor-Throttling dienen die Daten der Messungen ohne Prozessor-Throttling (siehe Abschnitt 6.4.4, Tabelle 6.18). Die Zunahme der Laufzeit ist bei allen Throttling-Stufen so groß, dass trotz einer niedrigeren durchschnittlichen Leistungsaufnahme des Prozessors der Gesamt-Energieverbrauch zunimmt. Der Einfluss der erhöhten Laufzeit ist damit stärker zu gewichten als die geringere Leistungsaufnahme des Servers durch Throttling.

#### Verbesserung des Nutzungsgrads des Prozessors durch Throttling

Am Beispiel von Abfrage 3 bei 50 %-Throttling wird die Auswirkung des Throttlings deutlich. Die Abbildungen 6.45 (kein Throttling) und 6.46 (50 %-Throttling) zeigen,

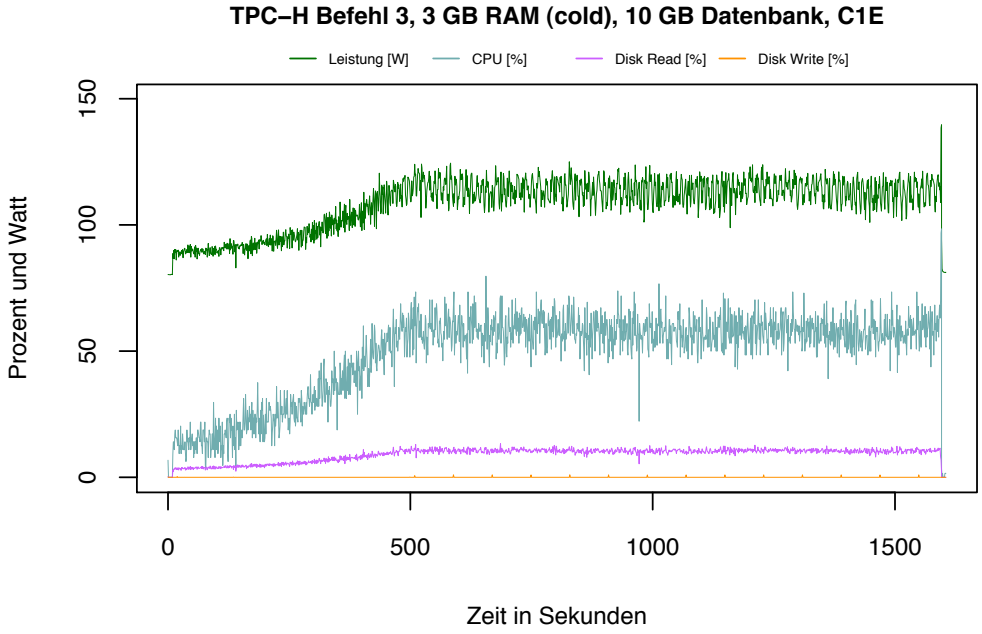



Abbildung 6.45: TPC-H Abfrage 3 (kein Throttling).

dass durch Throttling der Nutzungsgrad des Prozessors (blaue Linie) in I/O-lastigen Phasen verbessert wird. Zusätzlich sinkt die durchschnittliche Leistungsaufnahme um 5,10 % (siehe Tabelle 6.23). Allerdings nimmt die Laufzeit mit 64,18 % so stark zu, dass die Reduktion der durchschnittlichen Leistungsaufnahme mehr als kompensiert wird. Auch für alle anderen Abfragen kann durch Throttling der durchschnittliche

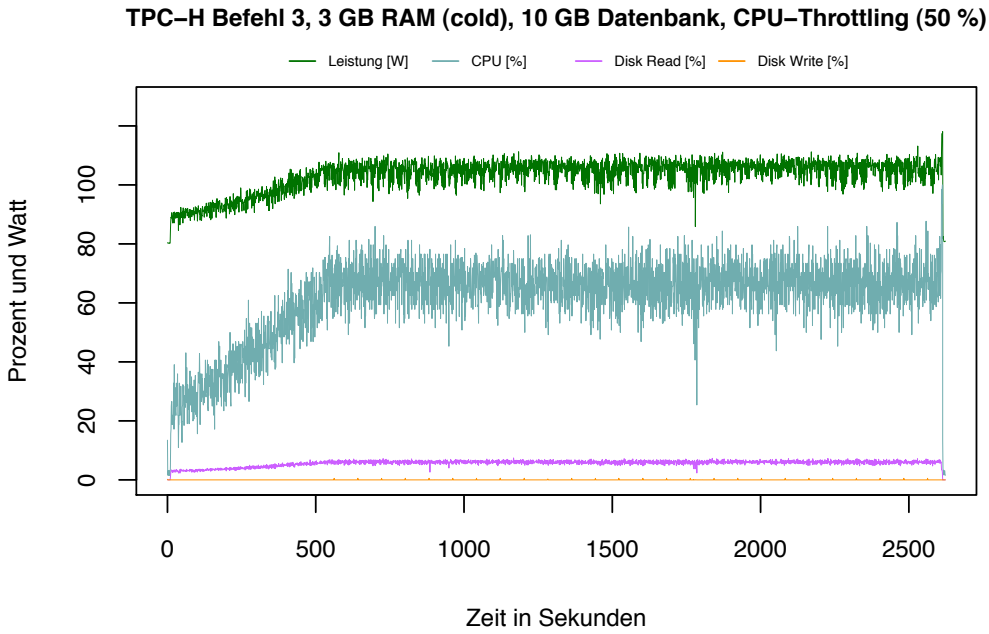



Abbildung 6.46: TPC-H Abfrage 3 (50 %-Throttling).

Nutzungsgrad des Prozessors verbessert werden, allerdings nimmt auch für diese Abfragen der Energieverbrauch durch die erhöhten Laufzeiten nicht ab.

### Zusammenhang von Energieverbrauch und Frequenz-Drosselung

Bei Abfrage 5 nimmt die Laufzeit bei den unterschiedlichen Throttling-Stufen weniger stark zu als bei den meisten anderen Abfragen. Unsere Logfiles zu den Messungen zeigen, dass diese Abfrage, verglichen mit den anderen TPC-H-Abfragen, mit einer durchschnittlichen Prozessorauslastung von 33,79 % nicht sonderlich CPU-lastig ist. Wird der Prozessor auf 75 % seiner Gesamtleistung gedrosselt (eine Reduktion der Leistung um 25 %), nehmen die Laufzeit und der Energieverbrauch nur um ungefähr 10 % zu. Auch bei einer Drosselung des Prozessors um 50 % liegt die Zunahme der Laufzeit und des Energieverbrauchs mit ungefähr 33 % unter der prozentualen Reduktion der Prozessorleistung. Erst bei einer Drosselung des Prozessors auf nur 25 % seiner Gesamtleistung (eine Reduktion der Leistung um 75 %) ist die prozentuale Zunahme der Laufzeit und des Energieverbrauchs mit jeweils ungefähr 90 % größer. Dies zeigt, dass der Einfluss unterschiedlicher Throttling-Stufen auf die Laufzeit und den Energieverbrauch dieser Abfrage nicht proportional ist.

Die für Abfrage 5 getroffenen Aussagen gelten auch für die anderen Abfragen, jedoch nimmt die Laufzeit bei CPU-lastigen Abfragen stärker zu. Der Zusammenhang der durchschnittlichen Prozessorauslastung und der Veränderung der Laufzeit, des Energieverbrauchs und der durchschnittlichen Leistungsaufnahme (Joule/s) bei un-

terschiedlichen Throttling-Stufen ist in den Tabellen 6.25, 6.26 und 6.27 angegeben. Diese Tabellen sind nicht nach den Nummern der TPC-H-Abfragen, sondern nach

| Nr. | Durchschnittliche Prozessorauslastung [%] | Laufzeit-Veränderung [%] | Joule-Veränd. [%] | Joule/s-Veränd. [%] |
|---|---|---|---|---|
| 10 | 24,54 | +6,65 | +7,23 | +0,54 |
| 5 | 33,79 | +10,55 | +10,35 | -0,18 |
| 2 | 36,28 | +0,67 | +2,30 | +3,32 |
| 14 | 41,96 | +14,61 | +14,09 | -0,45 |
| 4 | 68,37 | +21,10 | +19,32 | -1,47 |
| 16 | 83,21 | +27,90 | +22,01 | -4,61 |
| 6 | 87,25 | +31,52 | +23,77 | -5,89 |
| 12 | 87,40 | +33,57 | +26,09 | -5,60 |
| 15 | 87,85 | +31,99 | +24,52 | -5,66 |
| 11 | 96,19 | +34,68 | +24,56 | -7,51 |
| 13 | 96,41 | +34,82 | +24,59 | -7,59 |

Tabelle 6.25: Auswirkung der durchschnittlichen Prozessorauslastung auf die Änderung der Laufzeit, des Energieverbrauchs und die durchschnittliche Leistungsaufnahme (Joule/s) bei einer Drosselung der Prozessor-Frequenz auf 75 % der maximalen Frequenz. Die Tabelle ist nach der Spalte „Durchschnittliche Prozessorauslastung“ aufsteigend sortiert.

der Spalte „Durchschnittliche Prozessorauslastung“ aufsteigend sortiert. Durch die Sortierung ist der Einfluss der durchschnittlichen Prozessorauslastung besser zu erkennen. Die Abfragen, bei denen eine Zeitüberschreitung bei dem Testlauf mit der größten Frequenz-Drosselung auftrat, sind in dieser Tabelle nicht enthalten, damit eine direkte Vergleichbarkeit über alle Throttling-Stufen hinweg gegeben ist.

Anhand dieser Informationen kann folgende Heuristik aufgestellt werden: Je stärker die Prozessor-Frequenz gedrosselt wird, desto stärker nehmen die Laufzeit und der Energieverbrauch zu. Durch Throttling kann lediglich die durchschnittliche Leistungsaufnahme gesenkt werden. Wird die Frequenz eines Prozessors maximal auf die Hälfte gedrosselt, dann sinkt die durchschnittliche Leistungsaufnahme des Datenbankservers für Abfragen, die mindestens eine durchschnittliche Prozessorauslastung von 40 % aufweisen. Wird die Prozessor-Frequenz hingegen auf 25 % der maximalen Frequenz gedrosselt, dann muss die durchschnittliche Prozessorauslastung einer Abfrage mindestens 70 % betragen, damit die durchschnittliche Leistungsaufnahme des Datenbankservers sinkt.

Wird die Frequenz des von uns verwendeten Prozessors auf 75 % der maximalen Frequenz gedrosselt, dann sinkt die durchschnittliche Leistungsaufnahme des Datenbankservers um 0 bis 8 % für Abfragen, die mindestens eine durchschnittliche Prozessorauslastung von 40 % aufweisen. Bei einer Frequenz-Drosselung um 50 % sinkt

| Nr. | Durchschnittliche Prozessorauslastung [%] | Laufzeit-Veränderung [%] | Joule-Veränd. [%] | Joule/s-Veränd. [%] |
|---|---|---|---|---|
| 10 | 24,54 | +38,11 | +38,62 | +0,37 |
| 5 | 33,79 | +33,26 | +32,72 | -0,41 |
| 2 | 36,28 | +11,70 | +15,66 | +3,54 |
| 14 | 41,96 | +57,10 | +51,44 | -3,60 |
| 4 | 68,37 | +63,73 | +55,34 | -5,12 |
| 16 | 83,21 | +81,92 | +62,58 | -10,63 |
| 6 | 87,25 | +87,13 | +65,22 | -11,71 |
| 12 | 87,40 | +91,02 | +68,61 | -11,73 |
| 15 | 87,85 | +89,41 | +67,05 | -11,81 |
| 11 | 96,19 | +98,31 | +69,68 | -14,43 |
| 13 | 96,41 | +104,05 | +72,86 | -15,29 |

Tabelle 6.26: Auswirkung der durchschnittlichen Prozessorauslastung auf die Änderung der Laufzeit, des Energieverbrauchs und die durchschnittliche Leistungsaufnahme (Joule/s) bei einer Drosselung der Prozessor-Frequenz auf 50 % der maximalen Frequenz. Die Tabelle ist nach der Spalte „Durchschnittliche Prozessorauslastung“ aufsteigend sortiert.

| Nr. | Durchschnittliche Prozessorauslastung [%] | Laufzeit-Veränderung [%] | Joule-Veränd. [%] | Joule/s-Veränd. [%] |
|---|---|---|---|---|
| 10 | 24,54 | +65,32 | +73,62 | +5,02 |
| 5 | 33,79 | +87,26 | +92,61 | +2,86 |
| 2 | 36,28 | +68,11 | +78,43 | +6,14 |
| 14 | 41,96 | +110,75 | +111,88 | +0,54 |
| 4 | 68,37 | +178,75 | +162,20 | -5,94 |
| 16 | 83,21 | +218,02 | +181,25 | -11,56 |
| 6 | 87,25 | +229,01 | +186,47 | -12,93 |
| 12 | 87,40 | +240,58 | +196,60 | -12,91 |
| 15 | 87,85 | +236,56 | +192,56 | -13,07 |
| 11 | 96,19 | +263,34 | +205,03 | -16,05 |
| 13 | 96,41 | +266,41 | +205,18 | -16,71 |

Tabelle 6.27: Auswirkung der durchschnittlichen Prozessorauslastung auf die Änderung der Laufzeit, des Energieverbrauchs und die durchschnittliche Leistungsaufnahme (Joule/s) bei einer Drosselung der Prozessor-Frequenz auf 25 % der maximalen Frequenz. Die Tabelle ist nach der Spalte „Durchschnittliche Prozessorauslastung“ aufsteigend sortiert.

die durchschnittliche Leistungsaufnahme um 0 bis 16 % für Abfragen, die mindestens eine durchschnittliche Prozessorauslastung von 40 % aufweisen. Wird die Prozessor-Frequenz hingegen auf 25 % der maximalen Frequenz gedrosselt, dann muss die durchschnittliche Prozessorauslastung einer Abfrage mindestens 70 % betragen, damit die durchschnittliche Leistungsaufnahme des Datenbankservers um 0 bis 17 % sinkt. Bei allen drei Throttling-Stufen führt eine höhere durchschnittliche Prozessorauslastung zu einer größeren Reduktion der durchschnittlichen Leistungsaufnahme.

Je CPU-lastiger eine Abfrage ist, desto stärker nimmt jedoch auch die Laufzeit aufgrund der Frequenz-Drosselung zu. Mit Zunahme der Laufzeiten von maximal 35 % (Throttling-Stufe 75 %), 105 % (Throttling-Stufe 50 %) und 267 % (Throttling-Stufe 25 %) sind diese viel zu hoch, um trotz der niedrigeren durchschnittlichen Leistungsaufnahme Energie zu sparen.

### Gesamtbetrachtung des TPC-H Benchmarks bezüglich Throttling

In Tabelle 6.28 sind die Gesamtlaufzeit und der Gesamt-Energieverbrauch mit C1E und bei unterschiedlichen Throttling-Stufen angegeben. Auf niedrigeren Throttling-Stufen wurde die zulässige Berechnungsdauer von einer Stunde häufiger überschritten. Um eine Vergleichbarkeit der Gesamtergebnisse untereinander zu ermöglichen, wurden nur die Abfragen ausgewertet, bei denen auch auf der niedrigsten Throttling-Stufe keine Zeitüberschreitungen auftraten.

| Throttling-Stufe | Laufzeit [s] | Joule |
|---|---|---|
| 100 % (kein Throttling) | 6157,46 | 721990,36 |
| 75 % | 7563,30 (+22,83 %) | 861238,87 (+19,29 %) |
| 50 % | 10503,03 (+70,57 %) | 1141494,76 (+58,10 %) |
| 25 % | 16905,05 (+174,55 %) | 1855801,50 (+157,04 %) |

Tabelle 6.28: Gesamtlaufzeit und Gesamt-Energieverbrauch mit C1E und bei unterschiedlichen Throttling-Stufen. Die Prozentangaben in Klammern geben den Unterschied zu den Messergebnissen ohne Throttling an.

Die Messungen ohne Prozessor-Throttling brachten bezüglich des Energieverbrauchs des TPC-Benchmarks die besten Ergebnisse. Je weiter der Prozessor gedrosselt wurde, desto stärker nahmen die Laufzeit (Abbildung 6.47) und der Energieverbrauch (Abbildung 6.48) zu.

Insgesamt kann die Energieeffizienz einer Datenbank allein durch Prozessor-Throttling nicht gesteigert werden, da die Reduktion der Leistungsaufnahme des Prozessors durch die Senkung der Prozessor-Frequenz allein zu gering ist, um den zusätzlichen Energiebedarf aufgrund der längeren Laufzeiten aufzufangen. Lediglich die durchschnittliche Leistungsaufnahme des Datenbank-Servers kann durch Throttling reduziert werden. Eine parallel mit der Senkung der Frequenz einhergehende Senkung

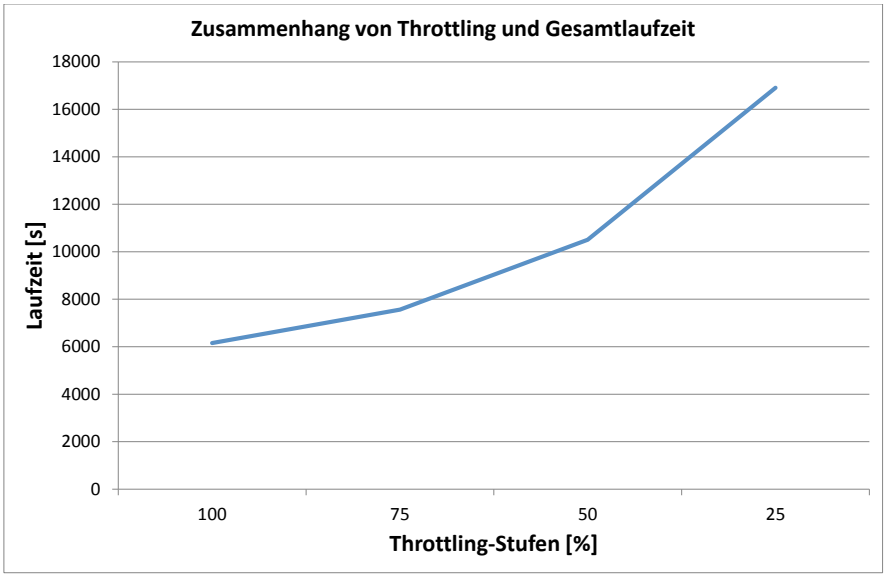



Abbildung 6.47: Zusammenhang von Throttling und Gesamtlaufzeit.

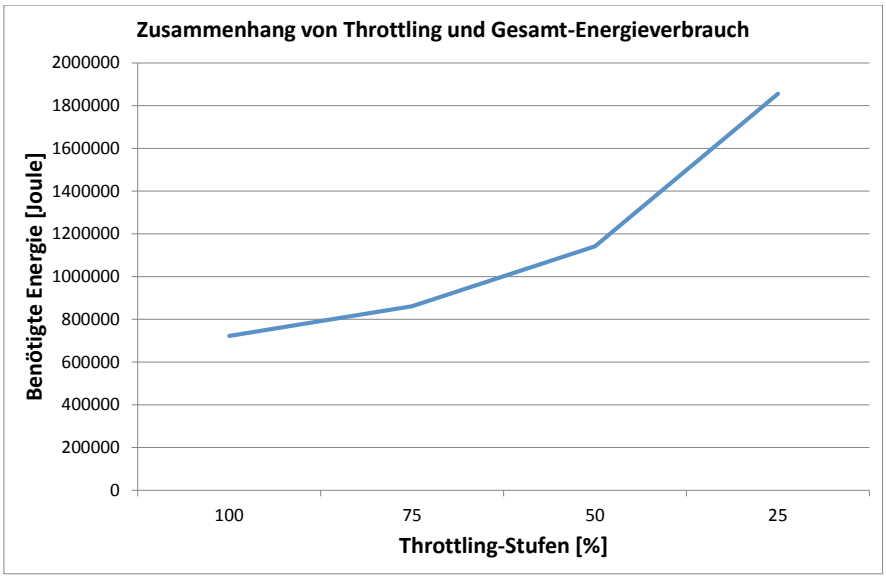



Abbildung 6.48: Zusammenhang von Throttling und Gesamt-Energieverbrauch.

der Prozessor-Spannung (siehe *P-States*, Abschnitt 6.4.6) liefert letztlich einen Beitrag zur Verbesserung der Energieeffizienz einer Datenbank, allerdings unterstützt der von uns verwendete Prozessor keine Spannungsänderungen.

### 6.4.10 Zusammenfassung der Ergebnisse

#### Spannungs- und Frequenzanpassung (Enhanced Halt State, C1E)

Durch Spannungs- und Frequenzanpassungen des Prozessors an die tatsächliche Datenbanknutzung kann die Energieeffizienz einer DBMS erhöht werden. Schon bei zwei unterschiedlichen Prozessorzuständen (C1E) kann eine Anpassung der Spannung und der Frequenz an die Prozessornutzung durch ein DBMS erreicht werden. Für den TPC-H Benchmark konnte insgesamt eine Reduktion des Energieverbrauchs um 2,74 % bei einer Zunahme der Laufzeit um 2,09 % festgestellt werden.

Bezüglich einzelner Abfragen gilt folgende Heuristik: Je niedriger die durchschnittliche Prozessorauslastung ist, desto größer ist die Reduktion der benötigten Energie für die Ausführung einer Abfrage. Die *Enhanced Halt State*-Funktion ist für I/O-intensive Abfragen, bei denen der Prozessor nicht vollständig ausgelastet ist, empfehlenswert. Die prozentuale Veränderung der Laufzeit $f_{Laufzeit}(x)$ anhand der durchschnittlichen Prozessorauslastung (Variable $x$) kann mit folgender Funktion approximiert werden:

$$f_{Laufzeit}(x) = 0,0292x + 0,6825 \tag{6.3}$$

Für die prozentuale Veränderung des Energieverbrauchs $f_{Joule}(x)$ gilt:

$$f_{Joule}(x) = 0,1231x - 10,568 \tag{6.4}$$

Abbildung 6.49 stellt den Zusammenhang zwischen der durchschnittlichen prozentualen Prozessorauslastung und der prozentualen Veränderung der Laufzeit beziehungsweise der prozentualen Veränderung des Energieverbrauchs grafisch für den Wertebereich $20 \leq x < 100$ dar. Anhand dieser Heuristik können die Laufzeit und der Energieverbrauch anhand der durchschnittlichen Prozessorauslastung approximiert werden und ein Datenbank-Administrator kann entscheiden, ob eine Aktivierung der *Enhanced Halt State*-Funktion für ihn lohnenswert ist. Liegt die durchschnittlich Prozessorauslastung einer Abfrage oder eines Datenbank-Workloads unter 86 %, dann kann mit C1E Energie gespart werden.

#### Drosselung der Prozessor-Frequenz (Prozessor-Throttling):

Eine zusätzliche Drosselung der Prozessor-Frequenz ohne Senkung der Kernspannung führt zu einer Reduktion der durchschnittlichen Leistungsaufnahme des Datenbank-Servers, aber gleichzeitig zu einem deutlichen Anstieg der Laufzeiten von Abfragen. Trotz der niedrigeren Leistungsaufnahme wird aufgrund der erhöhten Laufzeiten keine Energie gespart.

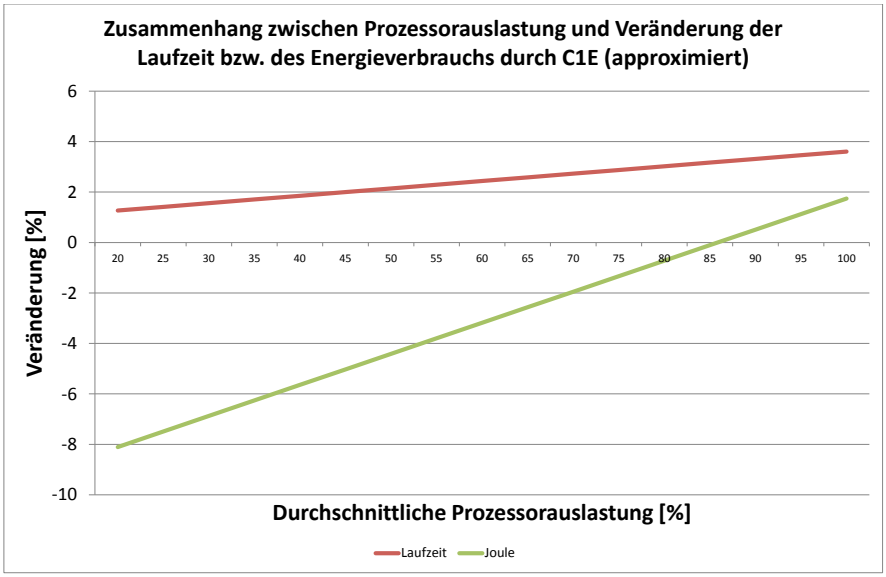



Abbildung 6.49: Approximierter Zusammenhang zwischen der durchschnittlichen Prozessorauslastung und der Veränderung der Laufzeit bzw. des Energieverbrauchs durch die C1E-Funktion.

Es gilt folgende Heuristik: Wird die Frequenz des von uns verwendeten Prozessors auf 75 % der maximalen Frequenz gedrosselt, dann sinkt die durchschnittliche Leistungsaufnahme des Datenbankservers um 0 bis 8 % für Abfragen, die mindestens eine durchschnittliche Prozessorauslastung von 40 % aufweisen. Bei einer Frequenz-Drosselung um 50 % sinkt die durchschnittliche Leistungsaufnahme um 0 bis 16 % für Abfragen, die mindestens eine durchschnittliche Prozessorauslastung von 40 % aufweisen. Wird die Prozessor-Frequenz hingegen auf 25 % der maximalen Frequenz gedrosselt, dann muss die durchschnittliche Prozessorauslastung einer Abfrage mindestens 70 % betragen, damit die durchschnittliche Leistungsaufnahme des Datenbankservers um 0 bis 17 % sinkt. Bei allen drei Throttling-Stufen führt eine höhere durchschnittliche Prozessorauslastung zu einer größeren Reduktion der durchschnittlichen Leistungsaufnahme.

Je CPU-lastiger eine Abfrage ist, desto stärker nimmt jedoch auch die Laufzeit aufgrund der Frequenz-Drosselung zu. Mit Zunahme der Laufzeiten von maximal 35 % (Throttling-Stufe 75 %), 105 % (Throttling-Stufe 50 %) und 267 % (Throttling-Stufe 25 %) sind diese viel zu hoch, um trotz der niedrigeren durchschnittlichen Leistungsaufnahme Energie zu sparen. Eine alleinige Drosselung der CPU-Frequenz ist damit keine geeignete Maßnahme, um Energie zu sparen. Erst durch eine dynamische Anpassung der CPU-Frequenz und der CPU-Spannung an den Nutzungsgrad des DBMS (siehe *P-States*, Abschnitt 6.4.6) kann Energie gespart werden.

## 6.5 TPC-H Benchmark mit allen Energieoptimierungen

### 6.5.1 Einleitung

Nachdem die Auswirkungen verschiedener Energieoptimierungen der Datenbank Caché in den vorherigen Abschnitten des Kapitels 6 bereits getrennt voneinander untersucht wurden, sind in Tabelle 6.29 die Ergebnisse einer kombinierten Anwendung dieser energieeinsparenden Methoden angegeben. Als Workload dient auch hier wieder der TPC-H Benchmark. Ein Vergleich der Ergebnisse in diesem Abschnitt mit den TPC-H Ergebnissen auf der nicht optimierten Datenbank (siehe Abschnitt 6.1.3, Tabelle 6.4) verdeutlichen das hohe Energie-Einsparpotenzial.

#### Analyse der Energieeinsparungen

Die Konfiguration für den Test bildeten die Einstellungen, die sich in den einzelnen Abschnitten als beste Konfiguration für die Reduzierung des Energiebedarfs herausgestellt haben.

- Indizes auf abgefragten Wertebereichen im WHERE-Teil der Abfrage.

| | 10 GB Datenbank (TPC-H Benchmark) mit allen Optimierungen | | |
|---|---|---|---|
| Nr. | Laufzeit [s] | Joule | Joule/s |
| 1 | Timeout | N/A | N/A |
| 2 | 15,07 (-74,74 %) | 1439,43 (-78,04 %) | 95,49 (-13,07 %) |
| 3 | 816,64 (-47,80 %) | 98596,89 (-45,44 %) | 120,74 (+4,51 %) |
| 4 | 151,71 (-38,78 %) | 14050,27 (-54,52 %) | 92,61 (-25,71 %) |
| 5 | 263,18 (-58,66 %) | 28632,92 (-58,59 %) | 108,80 (+0,16 %) |
| 6 | 264,93 (-44,75 %) | 30710,25 (-52,32 %) | 115,92 (-13,69 %) |
| 7 | 1890,57 (-28,62 %) | 236597,74 (-24,97 %) | 125,15 (+5,11 %) |
| 8 | 1007,21 (-62,05 %) | 122660,01 (-60,24 %) | 121,78 (+4,76 %) |
| 9 | Timeout | N/A | N/A |
| 10 | 207,76 (-75,16 %) | 20755,24 (-76,25 %) | 99,90 (-4,36 %) |
| 11 | 320,45 (-14,00 %) | 41002,67 (-19,32 %) | 127,95 (-6,19 %) |
| 12 | 20,88 (-96,24 %) | 1637,96 (-97,80 %) | 78,44 (-41,43 %) |
| 13 | 602,29 (+1,15 %) | 76549,05 (-6,83 %) | 127,10 (-7,88 %) |
| 14 | 117,42 (-89,31 %) | 11144,70 (-91,07 %) | 94,91 (-16,42 %) |
| 15 | 237,89 (-76,73 %) | 23000,84 (-83,21 %) | 96,69 (-27,88 %) |
| 16 | 81,21 (-17,71 %) | 9405,04 (-27,51 %) | 115,81 (-11,91 %) |
| 17 | Timeout | N/A N/A | N/A |
| 18 | 2463,926 (-1,27 %) | 312169,22 (-8,63 %) | 126,70 (-7,45 %) |
| 19 | 629,73 (-78,68 %) | 72082,84 (-78,28 %) | 114,47 (+1,87 %) |
| 20 | Timeout | N/A | N/A |
| 21 | Timeout | N/A | N/A |
| 22 | Timeout | N/A | N/A |
| Gesamt | 9090,86 (-50,37 %) | 1100435,05 (-50,40 %) | - |

Tabelle 6.29: Messergebnisse der 22 Abfragen des TPC-H Benchmarks auf der 10 GB-Datenbank mit allen Optimierungen.

- SSD mit einer Datenbankblockgröße von 64 KB.
- Aktivierter C1E-Zustand für den Prozessor.

Das Gesamtergebnis zeigt, dass die Energieeinsparung im Vergleich zur separaten Anwendung der Optimierungen angestiegen ist. Die Abbildung 6.50 verdeutlicht dies. Wir analysieren wie die zusätzlichen Einsparungen zustande kommen. Abbildung 6.51

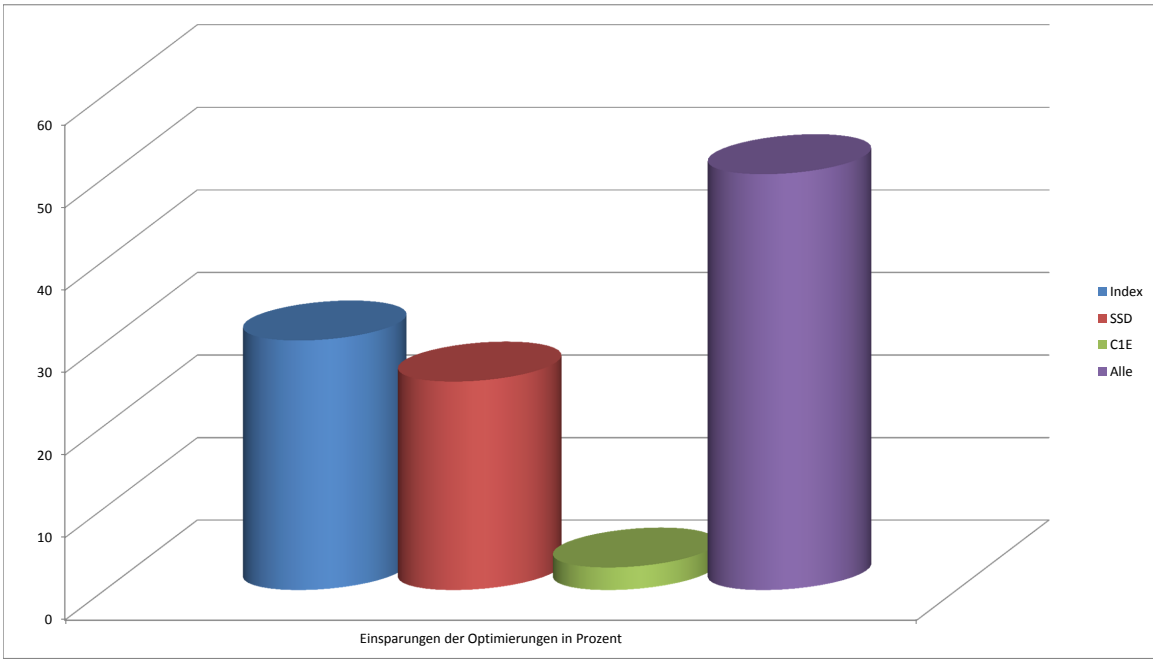



Abbildung 6.50: Vergleich der einzelnen Optimierungen.

zeigt die Auslastung des Gesamtsystems bei der Ausführung von Abfrage 5 mit zusätzlichem Index auf der HDD. In der Abbildung ist ersichtlich, dass die Ausführung mit der Index-Auswertung und mit einer CPU-Aktivität von 100 % beginnt. Dies ist an Modul B in Abfrageplan 6.53 ersichtlich. Die folgende Phase ist geprägt durch eine niedrige CPU-Aktivität und eine niedrige Leserate der HDD. In diesem Schritt der Verarbeitung wird Modul C aus Abfrageplan 6.53 aufgerufen. Die niedrige Leserate ergibt sich durch die Verbundoperation auf 5 Tabellen, wodurch die Festplatte häufig den Speicherbereich wechseln muss. Das Ende der Verarbeitung ist durch einen Anstieg der CPU-Aktivität und der Leserate charakterisiert. Abbildung 6.52 zeigt die Leistungsparameter bei der Ausführung auf der SSD und mit aktiviertem C1E-Zustand. Im Unterschied zu Abbildung 6.51 ist zu erkennen, dass keine Phase mit niedriger CPU-Aktivität und I/O-Aktivität existiert. Stattdessen gibt es einen kurzzeitigen Anstieg der Leserate auf annähernd 140 %.

Die SSD hat durch ihre Bauart keinen Nachteil beim Lesen in verschiedenen Speicherbereichen. Die Verarbeitung von Modul B kann deshalb innerhalb weniger Sekunden abgeschlossen werden. Die zusätzliche Energieersparnis ergibt sich hierbei durch die höhere Leserate der SSD bei häufigem Wechsel der Speicherbereiche. In Abschnitt

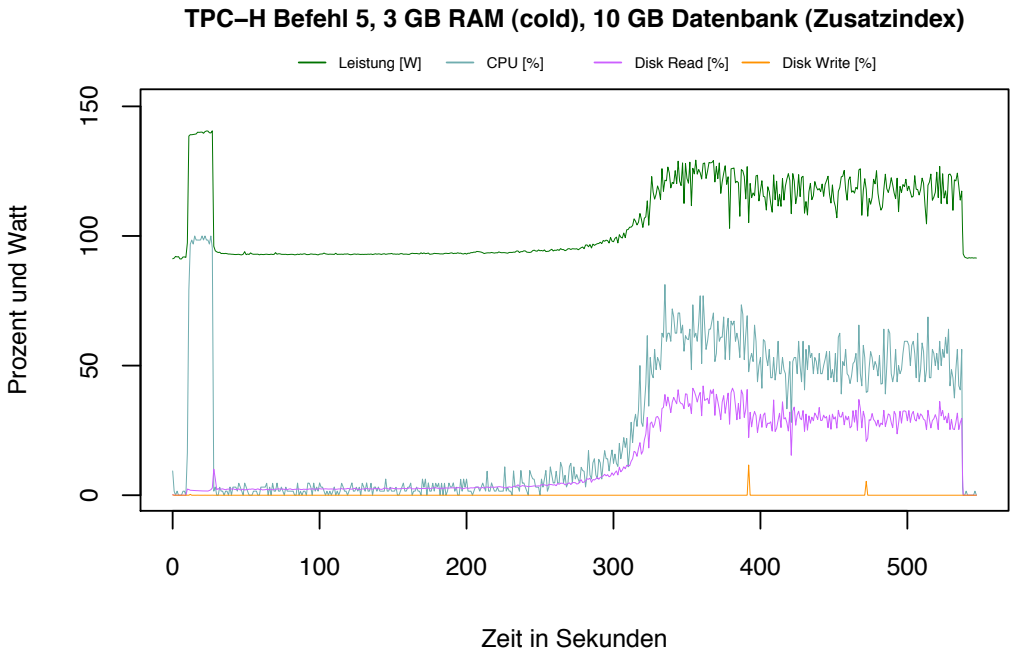



Abbildung 6.51: Auslastung TPC-H Abfrage 5 mit Index auf HDD.

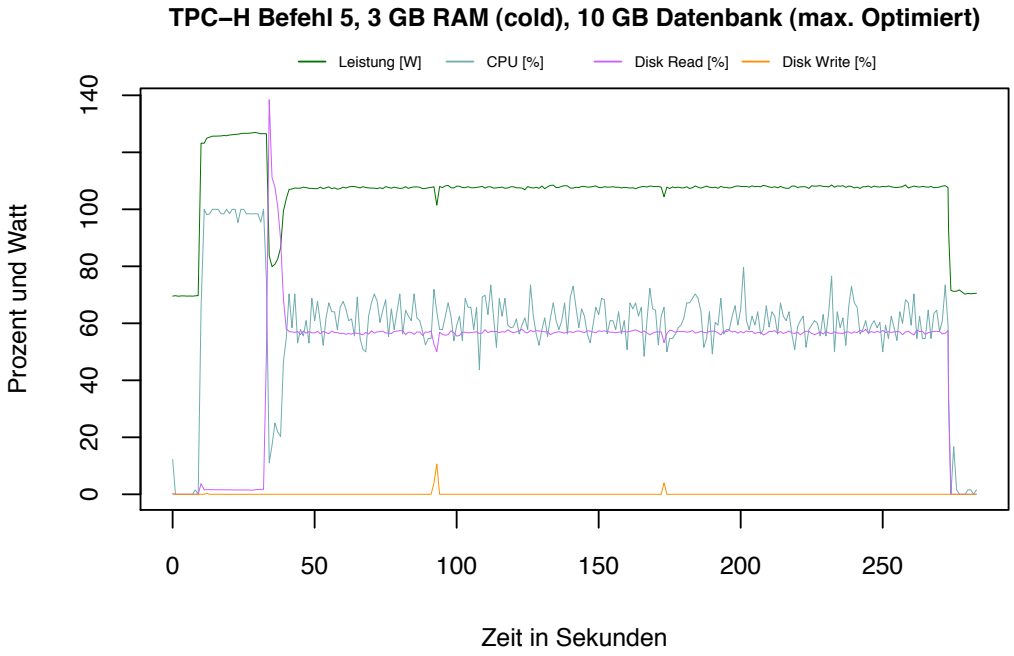



Abbildung 6.52: Auslastung TPC-H Abfrage 5 mit Index auf SSD.

**Abfrageplan**

Relative Kosten = 67169566

- Call module C, which populates temp-file B.
- Call module F, which populates temp-file C.
- Read temp-file C, looping on sum([expression]) and a counter.
- For each row:

  Output the row.

**module C**

- Call module B, which populates bitmap temp-file A.
- Read bitmap temp-file A, looping on ID.
- For each row:

  Read master map SQLUser.ORDERS.ORDERSPRIMARYIDX, using the given idkey value.
  Read master map SQLUser.CUSTOMER.CUSTORMERPRIMARYIDX, using the given idkey value.
  Read master map SQLUser.NATION.NATIONPRIMARYIDX, using the given idkey value.
  Read master map SQLUser.REGION.REGIONPRIMARYIDX, using the given idkey value.
  Read master map SQLUser.LINEITEM.LINEITEMPRIMARYIDX, using the given L_ORDERKEY, and looping on L_LINENUMBER.
  For each row:
  Read master map SQLUser.SUPPLIER.SUPPLIERPRIMARYIDX, using the given idkey value.

  Check distinct values for %SQLUPPER(N_NAME) using temp-file B,
  subscripted by a hashing of these values.
  For each distinct row:
  Add a row to temp-file B, subscripted by the hashing,
  with node data of %SQLUPPER(N_NAME), L_DISCOUNT, and L_EXTENDEDPRICE.
  Update the accumulated sum([expression]) in temp-file B,
  subscripted by the hashing.

**module F**

- Read temp-file B, looping on the hashing subscript.
- For each row:

  Add a row to temp-file C, subscripted by sum([expression]) and a counter,
  with node data of [expression].

**module B**

- Read index map SQLUser.ORDERS.OORDERDATEIDX, looping on O_ORDERDATE (with a range condition) and ID.
- For each row:

  Add ID bit to bitmap temp-file A.

Abbildung 6.53: Auslastung TPC-H Abfrage 5 mit Index auf SSD.

6.2.4 wurde festgestellt, das Abfrage 16 einen höheren Energiebedarf bei der Ausführung mit Index hat. Dies kann damit erklärt werden, dass beim Lesen ein ständiger Wechsel zwischen den Speicherbereichen des Index und der Daten stattfindet. Abbildung 6.54 zeigt die Ausführung mit einem Index auf der HDD. Abbildung 6.55 zeigt

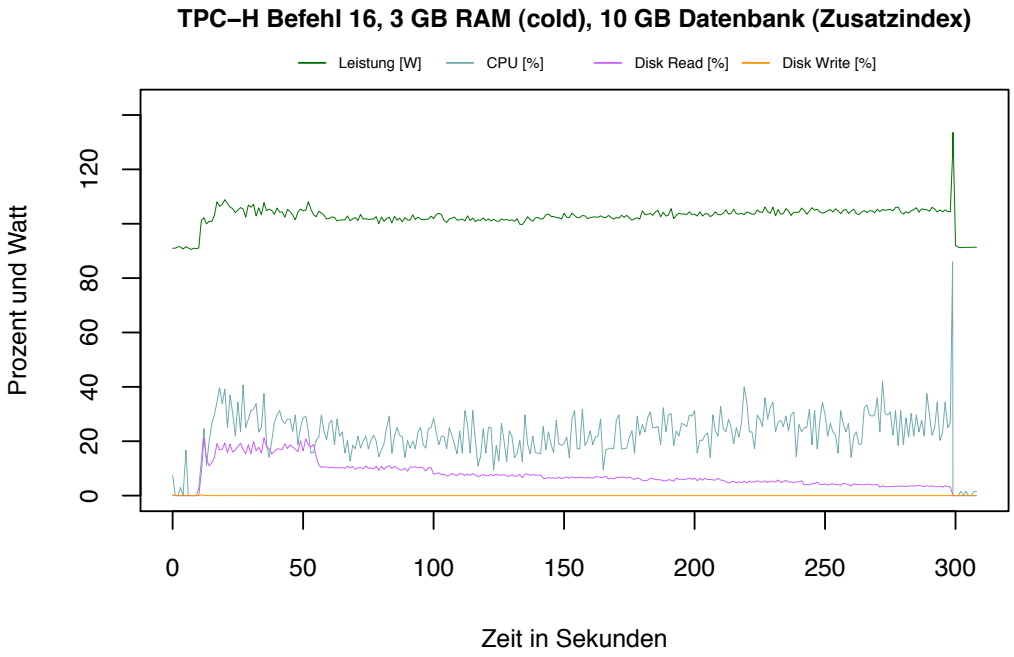



Abbildung 6.54: Auslastung TPC-H Abfrage 16 mit Index auf HDD.

die Verarbeitung auf der SSD. Die Leserate ist um ein Vielfaches höher als auf der HDD. Hier kommt der gleiche Effekt wie bei Abfrage 5 zum Tragen. Die SSD hat bei randomisiertem Lesen eine viel höhere Leserate als die HDD.

Während die Nutzung des Index zu einer Verschlechterung der Energiebilanz führt, ist es bei der SSD umgekehrt. Hierbei profitiert die Abfrage von dem zusätzlichen Index. Der Energiebedarf bei der Ausführung auf der SSD ohne zusätzlichen Index beträgt 11.349,90 Joule. Bei der Ausführung mit zusätzlichem Index und C1E liegt der Energiebedarf bei 9.405,04 Joule. Dies bedeutet eine zusätzliche Energieersparnis von 17 % und eine zusätzliche Verkürzung der Laufzeit.

Abbildung 6.56 zeigt die Hardware-Auslastung durch Abfrage 4 mit zusätzlichem Index auf der HDD. Zu Beginn der Ausführung, ist die für die Auswertung des Index eine charakteristische CPU-Auslastung von 100 % zu erkennen. Es folgt eine Phase mit einer hohen Leserate der HDD und einer CPU-Auslastung zwischen 25 % und 50 %. Diese Abfrage operiert auf nur 2 Tabellen, daher die vergleichsweise hohe Leserate.

Abbildung 6.57 zeigt die Auslastung auf der SSD mit C1E. Die Charakteristik der

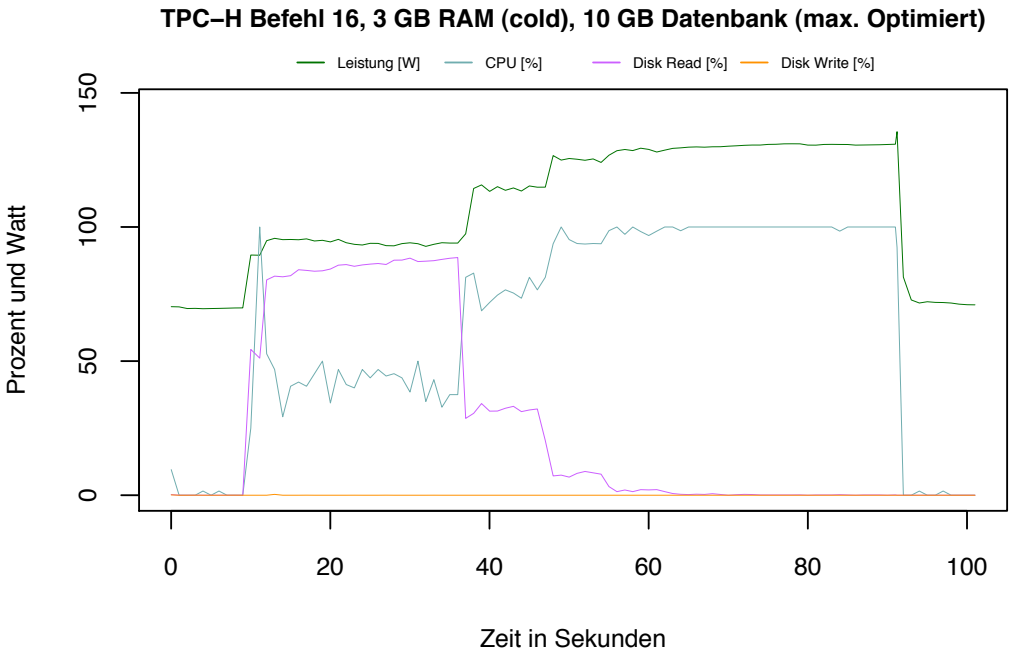



Abbildung 6.55: Auslastung TPC-H Abfrage 16 mit Index auf SSD.

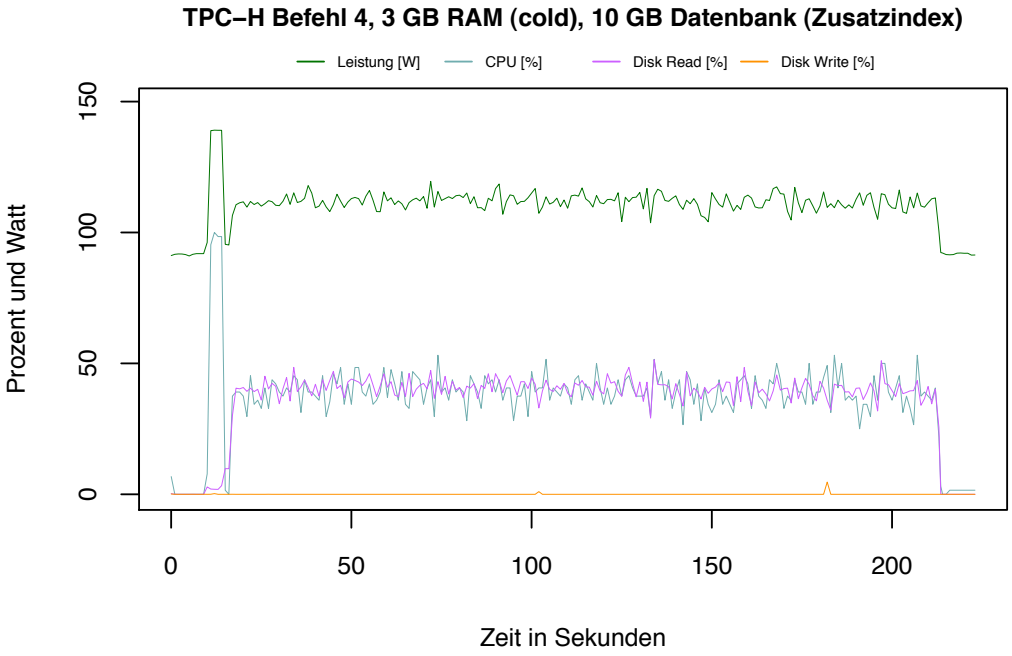



Abbildung 6.56: Auslastung TPC-H Abfrage 4 mit Index auf HDD.

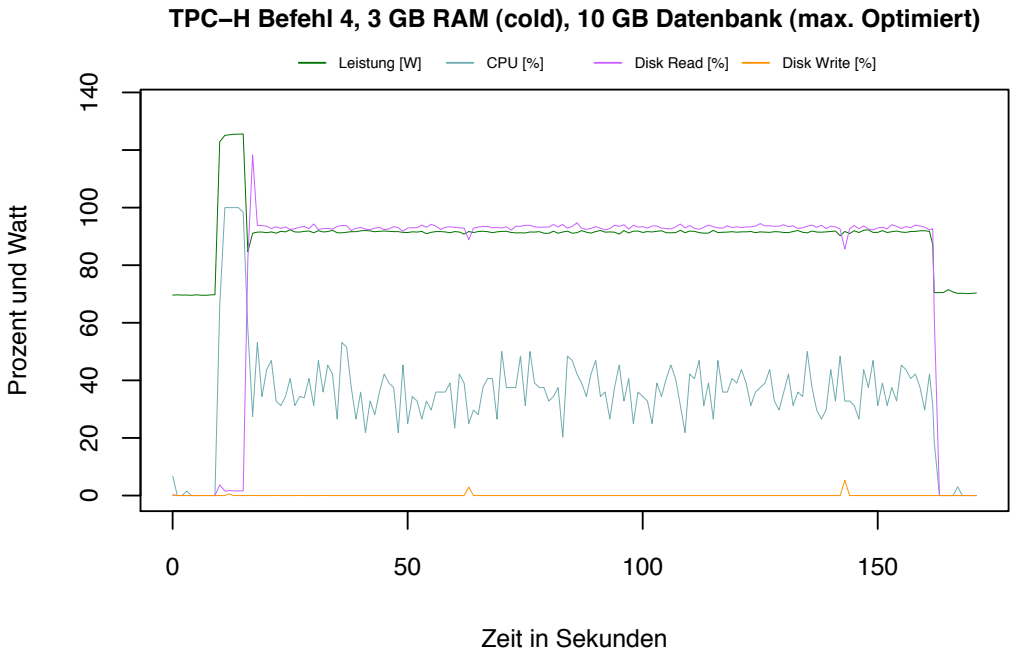



Abbildung 6.57: Auslastung TPC-H Abfrage 4 mit Index auf HDD.

Verarbeitung ändert sich nicht, wie am Vergleich beider Abbildung zu sehen ist. Der Unterschied besteht in der Leserate, die bei der Verarbeitung auf der SSD deutlich höher ist. Dadurch ist die Laufzeit auf der SSD insgesamt geringer und damit auch der Energieverbrauch.

Die CPU-Aktivität ist bei beiden Testläufen auf ähnlichem Niveau. Bei der Ausführung auf der HDD lag der durchschnittliche Energieverbrauch bei 112,12 Watt, auf der SSD bei 92,6 Watt. Die SSD verbraucht durch ihre Bauart 10 Watt weniger Energie 6.3.3. Die zusätzlichen 10 Watt, welche im Durchschnitt eingespart wurden, ergeben sich aus der vergleichsweise niedrigen CPU-Auslastung und dem aktivierten C1E-Zustand.

### 6.5.2 Zusammenfassung

Die Nutzung aller Optimierungen aus Kapitel 6 in Kombination führt zu einem niedrigerem Energieverbrauch. Die einzelnen Optimierungen wirken an unterschiedlichen Stellen der Verarbeitung der Abfragen. Dies führt in der Summe zu einer höheren Energieersparnis. Wir haben bei der Ausführung des TPC-H Benchmarks mit allen Optimierungen 50,40 % Energie einsparen können. Zusammengefasst wirken die Optimierungen wie folgt auf den Energiebedarf:

- Der zusätzliche Index schließt nicht benötigte Datensätze effizient aus der Berechnung aus und reduziert somit die Gesamtzahl an Berechnung und die Menge

an Daten, die von der Festplatte geladen werden müssen.

- Die SSD ermöglicht ein schnelleres Laden der erforderlichen Daten für die Berechnung und verkürzt die Verarbeitungszeit. Zusätzlich verbraucht die SSD durch ihre Bauart weniger Energie.
- Der aktivierte C1E-Zustand spart in Phasen niedriger CPU-Aktivität zusätzlich Energie.

# 7 Zusammenfassung und Ausblick

Zu Beginn der vorliegenden Arbeit wird der Begriff *Green IT* erläutert und es wird die ökonomische Bedeutung von Green IT für Rechenzentren und Unternehmen aufgezeigt. Weiterhin werden allgemeine Maßnahmen aufgezählt, mit denen bereits heute in Unternehmen Energie gespart wird. Im weiteren Verlauf des Grundlagen-Kapitels wird die Bedeutung von energieeffizienten Datenbankmanagementsystemen (DBMS) in Rechenzentren beschrieben und aktuelle Forschungsergebnisse erläutert. Da in dieser Diplomarbeit das DBMS *Caché* zum Einsatz kommt, ist ebenso eine Beschreibung der Besonderheiten der Datenbank *Caché* von *Intersystems* Teil des Grundlagen-Kapitels. Das erste Kapitel endet mit einer Beschreibung des TPC-H Benchmarks, welcher im späteren Verlauf der Arbeit zur Leistungsbewertung der Datenbank verwendet wird.

Nach dem Grundlagen-Kapitel wird im dritten Kapitel der Entwurf unseres Versuchsaufbaus vorgestellt. Der Versuchsaufbau erlaubt es, den Energieverbrauch des Datenbankservers und weitere Leistungsindikatoren wie Prozessoraktivität und Festplattennutzung zu messen, ohne den Datenbankserver durch die Messungen zu beeinflussen. Dazu muss der Energieverbrauch und die Leistungsindikatoren auf einem zusätzlichen Rechner aufgezeichnet werden. Zu diesem Zweck wurden zwei Programme, *GreenDB* und *PerformanceServer*, entwickelt.

Das vierte Kapitel beschreibt die Implementierung des im vorherigen Kapitel vorgestellten Entwurfs. Dieses Kapitel beinhaltet eine Beschreibung der Klassen unserer entwickelten Programme, die Einrichtung der Testumgebung bestehend aus Datenbankserver und Messrechner, und einige, für die späteren Messungen notwendigen Vorbereitungsschritte, wie zum Beispiel die Veränderung der Datenbank-Blockgröße oder der Import der Daten in die Datenbank Caché. Interessant ist dieses Kapitel unter Anderem deshalb, da die beschriebene Konfiguration für die Datenbank-Blockgröße nicht in der offiziellen Dokumentation von Caché enthalten ist. Weiterhin wird erläutert, warum die beiden Programme *GreenDB* und *PerformanceServer* zur zuverlässigen Erfassung der Messwerte notwendig sind.

Vor den eigentlichen Messungen in Kapitel fünf und sechs, wurde die durchschnittliche Leistungsaufnahme des Servers im Leerlauf ermittelt. Dazu wurde einmal die durchschnittliche Leistungsaufnahme des Servers im Leerlauf bei aktivierter Datenbanksoftware (ohne jedoch eine SQL-Abfrage an den Server zu schicken) und einmal bei deaktivierter Datenbanksoftware gemessen. Die gemessene durchschnittliche Leistungsaufnahme betrug in beiden Fällen etwa 90 Watt. Es macht demnach keinen Unterschied für den Energieverbrauch des Servers, ob die Datenbank gestartet wurde

oder nicht. Während der Bearbeitung einer Abfrage steigt die Leistungsaufnahme des Datenbankservers auf Werte bis zu 140 Watt.

Im weiteren Verlauf dieser Diplomarbeit werden verschiedene praktische Möglichkeiten untersucht, um den Energieverbrauch eines Datenbankservers zu senken. Dabei werden sowohl Hardwareaspekte, wie die Erweiterung des Arbeitsspeichers, den Einsatz von Solid State Discs (SSDs), oder Spannungs- und Frequenzanpassungen der CPU, aber auch die Möglichkeit den Energieverbrauch durch Softwareoptimierungen zu senken betrachtet. Nicht immer lassen sich Hardwareoptimierungen von Softwareoptimierungen strikt trennen. Insbesondere für den effizienten Einsatz einer SSD ist es relevant, dass ein DBMS eigens für die Verwendung einer SSD konfiguriert wird. Nur hierdurch kann das volle Potential dieser Technologie ausgeschöpft werden.

## 7.1 Ergebnisse

Die Ergebnisse unserer Untersuchungen für die verschiedenen Optimierungsmaßnahmen werden im Folgenden kurz zusammenfasst. Außer für die Testreihen mit unterschiedlichen Arbeitsspeicher- und Datenbankgrößen (Kapitel 5), für die wir einen eigenen Workload entwickelten, verwendeten wir für alle anderen Untersuchungen (Kapitel 6) den standardisierten und komplexen TPC-H Workload. Der Grund für einen eigenen und weniger komplexen Workload im fünften Kapitel ist, dass sich die Auswirkungen von unterschiedlichen Arbeitsspeicher- und Datenbankgrößen bei einfacheren Abfragen leichter ermitteln und kategorisieren lassen.

**Einfluss des Arbeitsspeichers auf den Energieverbrauch:** Anhand von drei unterschiedlich großen Datenbanken (1 GB, 5 GB und 10 GB) wurde festgestellt, dass eine Erweiterung des Arbeitsspeichers, in unserem Fall von 256 MB auf 3 GB, einen positiven Einfluss auf den Energieverbrauch des Datenbankservers hat. Dabei hängt das Einsparpotenzial von dem Verhältnis zwischen der Arbeitsspeicher- und Datenbankgröße ab. Die Reduktion des Energieverbrauchs für den gesamten Workload ist mit 3,36 % auf der 5 GB-Datenbank am größten. Auf der 1 GB-Datebank brauchte eine Erweiterung des Arbeitsspeichers fast keine Änderung, da der größere Arbeitsspeicher nicht vom DBMS für diese Datenbankgröße benötigt wurde. Auf der 10 GB-Datenbank betrug die Energiereduktion 2,08 % (siehe Abschnitt 5.3.2).

Sortierungen (*ORDERED BY*) und Verknüpfungen (*JOIN*) profitieren in besonderem Maße von einem größeren Arbeitsspeicher. Für die Gruppe der Sortierbefehle (Gruppe 3) beträgt die Energiereduktion je nach Datenbankgröße bis zu 5,63 %. Für die Gruppe der Verknüpfungen (Gruppe 6) wurde ein Reduktion von bis zu 15,23 % festgestellt. Diese Befehls-Typen sind es auch, welche den Gesamtwert für die Energiereduktion maßgeblich beeinflussen. Für Aggregatfunktionen, Selektionen oder Update-Funktionen hatte die Erweiterung des Arbeitsspeichers auf den Energieverbrauch einen geringen oder keinen Einfluss (siehe Abschnitt 5.3.3).

**Einfluss der Datenbankgröße auf den Energieverbrauch:** Die Untersuchungen mit unterschiedlichen Datenbankgrößen haben gezeigt, dass es einen linearen Zusammenhang zwischen der Datenbankgröße und dem durchschnittlichen Energieverbrauch beziehungsweise der Laufzeit aller Abfragen gibt. Eine Verdoppelung der Datenbankgröße führt zu einer Verdoppelung der Laufzeit, beziehungsweise des Energieverbrauches. Eine Ausnahme hiervon bilden einfache Selektionen, deren Laufzeit und Energieverbrauch weniger stark anwächst (siehe Abschnitt 5.3.4).

**Einfluss von Indizes auf den Energieverbrauch:** Einleitend wurden zwei Arten der Indizierung von Daten vorgestellt. Der konventionelle Index und der Bitmap-Index und es wurde die Besonderheiten beider Indizierungsarten dargelegt (siehe Abschnitt 6.2.2).

Der Abschnitt 6.2.3 behandelt die Thematik der Analyse von Abfragen und die daraus resultierenden Indizes. Der Ansatz der Indizierung von eingrenzenden Wertebereichen hat sich in Abschnitt 6.2.4 als gute Methode für die Einsparung von Energie erwiesen. Es ergab sich folgender Zusammenhang: Je mehr Datensätze durch den Index von der Berechnung ausgeschlossen werden, umso höher ist die Energieersparnis. Des Weiteren wurden in Abschnitt 6.2.4 festgestellt, dass Indizes nicht zwangsläufig zu einer Energieersparnis führen.

**Einsatz von Solid State Disks:** Zur Einführung in diese Thematik wurde das *In-Page-Logging* vorgestellt, welches teure Schreibzugriffe auf Flash-Speicher minimiert. Dies wird dadurch erreicht, dass Log-Einträge (in einer Log-Datei werden alle Änderungen an der Datenbank protokolliert) nur zusammen mit einer Daten-Seite in einen Block geschrieben werden. Dadurch kann das Potenzial eines Flash-Speichers besser genutzt werden (siehe Abschnitt 6.3.2).
Ebenso wurde die Datenbank FlashDB vorgestellt. FlashDB benutzt einen neuartigen Index, welcher sich dynamisch an die Struktur des zugrundeliegenden Speichers und Workloads anpasst. Der Index ($B^+$-Baum) besteht aus schreib-optimierten und lese-optimierten Knoten. Die Struktur des Baums wird dynamisch an den tatsächlichen Workload angepasst. Bei einem lese-intensiven Workload, besteht solch ein Baum beispielsweise überwiegend aus lese-optimierten Knoten (siehe Abschnitt 6.3.2).

Um die Nutzung einer Solid State Disc (SSD) durch die Datenbank Caché zu verbessern und dadurch den Energieverbrauch zu reduzieren, untersuchten wir die Auswirkungen von unterschiedlichen Datenbank-Blockgrößen auf den Energieverbrauch. Die Datenbank-Blockgröße kann in Caché erst nach Änderung einer Konfigurationsdatei über ein Terminal geändert werden (siehe Abschnitt 4.4). Die Standard-Blockgröße der Datenbank Caché beträgt 8 KB. Schon der alleinige Tausch eines Festplattenlaufwerks (HDD) durch eine SSD resultierte in einer Energieeinsparung von 18 %. Der Energieverbrauch konnte durch eine größere Blockgröße zusätzlich reduziert werden. Bei einer Datenbank-Blockgröße von 64 KB sank der Energieverbrauch um 25 %. Bei einer Blockgröße von 2 KB nahm hingegen der Energieverbrauch zu. Dies führt zu folgender Aussage für die von uns getesteten Blockgrößen im Bereich von 2 KB bis

64 KB: Je größer die Blockgröße desto größer ist die Reduktion des Energieverbrauchs für den von uns verwendeten TPC-H Workload (siehe Abschnitt 6.3.5).

**Spannungs- und Frequenzanpassung des Prozessors:** Bezüglich der Anpassung der Spannung und Frequenz eines Prozessors an den Workload (siehe Abschnitt 6.4.1) ist es zunächst entscheidend, welche Konfigurationsmöglichkeiten und Energiesparmodi ein Prozessor bietet. Nicht alle Prozessoren teilen den aktiven Prozessorzustand, in dem Instruktionen abgearbeitet werden, zusätzlich in mehrere (aktive) Zustände mit jeweils einer anderen Spannung und Frequenz auf (die sogenannte *P-States*, siehe Abschnitt 6.4.6). Allerdings bieten fast alle Prozessoren eine Funktion (*Enhanced Halt State*-Funktion oder auch kurz *C1E*), die eine Unterscheidung zwischen dem aktiven Prozessorzustand und einem Energiesparzustand erlaubt (siehe Abschnitt 6.4.3). Diese Funktion kann im BIOS moderner Mainboards aktiviert werden.

Anhand von Messungen und einer Analyse wird in dieser Arbeit gezeigt, dass durch Aktivierung dieser Funktion der Energieverbrauch eines Datenbankservers für den TPC-H Workload um bis zu 3 % reduziert werden kann. Für einzelne Abfragen beträgt die Energiereduktion bis zu 9 %. Bezüglich einzelner Abfragen gilt folgende Heuristik: Je niedriger die durchschnittliche Prozessorauslastung ist, desto größer ist die Reduktion der benötigten Energie für die Ausführung einer Abfrage. Die *Enhanced Halt State*-Funktion ist für I/O-intensive Abfragen, bei denen der Prozessor nicht vollständig ausgelastet ist, empfehlenswert. Liegt die durchschnittliche Prozessorauslastung einer Abfrage oder eines Datenbank-Workloads unter 86 %, dann kann mit C1E Energie gespart werden (siehe Abschnitt 6.4.4).

Weitere Messungen und Bestrebungen den Energieverbrauch des Datenbankservers durch das sogenannte CPU-Throttling (Drosselung der Prozessor-Frequenz) zu senken, leisteten keine positiven Beitrag. Zwar ließ sich durch eine Drosselung der Frequenz (ohne Änderung der Prozessor-Spannung) die durchschnittliche Leistungsaufnahme des Servers bei fast allen Abfragen senken, jedoch stieg durch eine überproportionale starke Verlängerung der Laufzeit der Energieverbrauch insgesamt an. Bei neueren Prozessoren, welche auch mit der Frequenz-Drosselung einhergehende Spannungsänderungen erlauben, ist dies jedoch eine vielversprechende Möglichkeit um Energie zu sparen (siehe Abschnitt 6.4.7).

Eine umfangreichere Übersicht über die Ergebnisse bezüglich der Spannungs- und Frequenzanpassung und die Heuristik zur Berechnung der Energie- und Laufzeitreduktion durch die *Enhanced Halt State*-Funktion, ist in Abschnitt 6.4.10 zu finden.

## 7.2 Caché Green Checklist

Die in diesem Abschnitt angegebenen Checkliste (*Caché Green Checklist*) gibt einem Anwender oder Administrator einer Caché-Datenbank einen Überblick über die die untersuchten Optimierungsmaßnahmen und deren Auswirkung auf den Energiever-

brauch der Datenbank.

## Caché Green Checklist

| | Energieoptimierung | Auswirkung |
|---|---|---|
| **Arbeitsspeicher** | Erweiterung des Arbeitsspeichers um mindestens ~3 GB. | Abhängig vom Abfrage-Typ werden bis zu 53 % Energie gespart. Empfehlenswert bei Sortierungen und Verknüpfungen (JOINs). |
| **Indexstrukturen** | Anlegen eines Index bei Range-Bedingungen (bei niedriger Selektivität der Spalte der Range-Bedingung). | Spart bis zu 82 % bei stark einschränkenden Range-Bedingungen. |
| **Solid-State-Drive** | Verwendung einer SSD statt einer HDD. | Spart bis zu 18 % Energie und 17 % Laufzeit. |
| | Bei Nutzung einer SSD die Datenbank-Blockgröße auf 64 KB einstellen (bei OLAP-Workload). | Spart zusätzlich 7 % Energie und 10 % Laufzeit. |
| **Prozessor-Nutzung** | Dynamische Spannung- und Frequenzanpassung des Prozessors (z.B. *Enhanced Halt State*-Funktion) im BIOS aktivieren | Spart durchschnittlich 3 % Energie. |
| | *Enhanced Halt State*-Funktion im BIOS aktivieren für I/O-lastige Abfragen. | Spart bis zu 9 % Energie. Je niedriger die Prozessorauslastung desto größer ist die Energieeinsparung. |

## 7.3 Ausblick

Durch die in dieser Arbeit beschriebenen Optimierungen konnte der Energieverbrauch der Datenbank Caché, für den aus komplexen Abfragen bestehenden TPC-H Workload (einem *Online Analytical Processing*-Workload, kurz OLAP-Workload) deutlich gesenkt werden.

Eine weitere vielversprechende Methoden der Softwareoptimierung ist der 1970 entwickelte *Bloom-Filter*. Ein Bloom-Filter ist eine probabilistische Datenstruktur, mit deren Hilfe schnell ermittelt werden kann, ob ein Element Mitglied einer Menge ist. Es gibt allerdings keine hundertprozentige Sicherheit für die Richtigkeit eines Ergebnisses. Ermittelt der Filter, dass ein Element kein Mitglied einer bestimmten Menge ist, dann ist dieses Ergebnis immer richtig. Ermittelt hingegen der Filter, dass ein Element Mitglied einer Menge ist, dann ist dies nur mit einer sehr hohen Wahrscheinlichkeit richtig. Dieses Verfahren wird beispielsweide in der Datenbank *Bigtable* von *Google* verwendet. Die Vermeidung von unnötigen Festplattenzugriffe für nicht existierende Zeilen oder Spalten reduziert die Laufzeit für Abfragen erheblich [CDG 08]. Interessant ist die Frage, ob der zusätzliche Rechenaufwand aufgrund des Bloom-Filters weniger Energie benötigt, als durch die Vermeidung von unnötigen Festplattenzugriffen eingespart werden kann.

Interessant für weiterführende Arbeiten ist ebenso die Frage, ob sich die Optimierungen der vorliegenden Arbeit auch bei anderen Workloads (zum Beispiel einem aus einfachen Abfragen bestehenden *Online-Transaction-Processing*-Workload, kurz OLTP-Workload) positiv auf den Energieverbrauch auswirken. Ein weiterer interessanter Aspekt ist der Einfluss von Mehrkern-Prozessoren und Raid-Systemen auf den Energieverbrauch. Nicht zuletzt kann auch ein Vergleich mit anderen DBMS aufschlussreich sein.

# Literaturverzeichnis


[ACP] ACPI 4.0, Advanced Configuration and Power Interface Specification 2010, http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf

[Ada10] Adar, M.: *Oracle Recovery kompakt Manager*. Books on Demand GmbH, 2010

[AMD] Open Compute Project, AMD Motherboard Spezifikation, http://opencompute.org/specs/Open_Compute_Project_AMD_Motherboard_v1.0.pdf

[BD09] Bertand Dufrasne, U. D. IBM DS8000: Introducing Solid State Drives. 2009

[BMSS10] Beckmann, A.; Meyer, U.; Sanders, P.; Singler, J.: Energy-efficient sorting using solid state disks. **In:** *International Conference on Green Computing* 0 (2010), S. 191–202

[Cac] Caché Technology-Guide, InterSystems Deutschland, http://www.intersystems.de/cache/technology/techguide/

[CDG 08] Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W. C.; Wallach, D. A.; Burrows, M.; Chandra, T.; Fikes, A.; Gruber, R. E.: Bigtable: A Distributed Storage System for Structured Data. **In:** *ACM Trans. Comput. Syst.* 26 (2008), June, S. 4:1–4:26. – ISSN 0734–2071

[Cor09] Corporation, I. B. M. Ready to Access DB2 for z/OS Data on Solid-State Drives. 7 2009

[EPA] U.S. Environmental Protection Agency, ENERGY STAR Program Requirements for Computers 1999, http://www.energystar.gov/ia/partners/prod_development/revisions/downloads/computer/Version5.0_Computer_Spec.pdf

[FC08] Fichter, K.; Clausen, J. Energieeffiziente Rechenzentren - Best Practice-Beispiele aus Europa, USA und Asien, Borderstep Institut (herausgegeben vom BMU) 2008. 2008

[Gad10] Gadatsch, A.: Auswirkungen von Green IT auf das IT-Controlling. **In:** Keuper, F. (Hrsg.); Hamidian, K. (Hrsg.); Verwaayen, E. (Hrsg.); Kalinowski, T. (Hrsg.): *transformIT*. Gabler, 2010, S. 357–373

[GP87] Gray, J.; Putzolu, F.: The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time. **In:** *Proceedings of the 1987 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 1987. – ISBN 0–89791–236–5, S. 395–398

[Gra04] Graefe, G.: Write-optimized B-trees. **In:** *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB Endowment, 2004. – ISBN 0–12–088469–0, S. 672–683

[Gra08] Graefe, G.: Database servers tailored to improve energy efficiency. **In:** *Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management*. New York, NY, USA : ACM, 2008. – ISBN 978–1–59593–964–7, S. 24–28

[Ham08] Hamilton, J. R.: Where Does the Power Go and What to do About it? **In:** *HotPower*, 2008

[HHOS11] Härder, T.; Hudlet, V.; Ou, Y.; Schall, D.: Energy Efficiency is not Enough, Energy Proportionality is Needed! **In:** *DASFAA'11,1st Int. Workshop on FlashDB*, 2011

[HOB09] Härder, T.; 0002, K. S.; Ou, Y.; Bächle, S.: Towards Flash Disk Use in Databases - Keeping Performance While Saving Energy? **In:** *BTW*, 2009, S. 167–186

[HS11a] Hudlet, V.; Schall, D.: Measuring Energy Consumption of a Database Cluster. **In:** *Proc. 14. GI-Fachtagung Datenbanksysteme für Business, Techologie und Web (BTW 2011), Demo-Programm, Kaiserslautern, Germany* Bd. P - 180. Bd. P - 180, 2011, S. 734–737

[HS11b] Hudlet, V.; Schall, D.: SSD != SSD - An Empirical Study to Identify Common Properties and Type-specific Behavior. **In:** *14. GI-Fachtagung Datenbanksysteme für Business, Techologie und Web (BTW 2011), Kaiserslautern, Germany* Bd. P - 180. Bd. P - 180, 2011, S. 430–441

[IBM10] IBM. IBM System Storage DS8000 Series. 2010

[Inta] Intel Corporation: Enhanced Intel SpeedStep, Technology for the Intel Pentium M Processor 2004, ftp://download.intel.com/design/network/papers/30117401.pdf, S. 4

[Intb] Intel Corporation: Intel Optimization Reference Manual 2011, http://www.intel.com/assets/PDF/manual/248966.pdf

[Intc] Intel Corporation: Intel Pentium 4 Processors Datasheet 2005, http://download.intel.com/design/Pentium4/datashts/30056103.pdf, S. 86

[Intd] Intel Technology and Research: Intel Eco-Rack Version 1.5, http://download.intel.com/technology/eep/ecorackwp.pdf, S. 6

[Inte] InterSystems Caché (Datenbank), http://www.intersystems.de/cache/index.html

[Intf] Open Compute Project, Intel Motherboard Spezifikation, http://opencompute.org/specs/Open_Compute_Project_Intel_Motherboard_v1.0.pdf

[KE09] Kemper, A.; Eickler, A.: *Datenbanksysteme - Eine Einführung, 7. Auflage*. Oldenbourg, 2009. – ISBN 978–3–486–59018–0

[Küh09] Kühne, K.: *Verwaltung von Statistiken zur Indexkonfiguration für das Self-Tuning*, Universität Magdeburg, Diplomarbeit, 2009

[Kla10] Klaentsch, P. T.: Den Energieverbrauch für die Kühlung senken. **In:** *HK-Gebäudetechnik* 10 (2010), S. 58–60

[KZSE10] Kolbe, L. M.; Zarnekow, R.; Schmidt, N. H.; Erek, K.: Study: Sustainability and Green IT in IT organizations. **In:** *Research Papers in Information Systems Management* 1 (2010)

[Lan09] *CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings* www.crdrdb.org, 2009

[LD09] Lotus Douglas, Q. G. e. a. Driving Business Value on Power Systems with Solid State Drives. 4 2009

[Lee07] Lee, S. won: Design of flash-based dbms: an in-page logging approach. **In:** *In SIGMOD Conference*, ACM, 2007, S. 55–66

[LMG] ZES ZIMMER Electronic Systems, Datenblatt LMG95, http://www.zes.com/download/products/zes_lmg95_datasheet_e.pdf

[Mul02] Mull, A. Optimizing Energy Consumption by Event-Driven Frequency Scaling. Mai 2002

[Nat07] Nath, S.: Flashdb: dynamic self-tuning database for nand flash. **In:** *In IPSN*, ACM, 2007, S. 410–419

[OH10] Ou, Y.; Härder, T.: CFDC: A Flash-Aware Buffer Management Algorithm for Database Systems. **In:** *ADBIS 2010* Bd. 6295. Bd. 6295, Springer, 9 2010, S. 435–449

[OHS10] Ou, Y.; Härder, T.; Schall, D.: Performance and power evaluation of flash-aware buffer algorithms. **In:** *Proceedings of the 21st international conference on Database and expert systems applications: Part I*. Berlin, Heidelberg : Springer-Verlag, 2010, S. 183–197

[RD05] Ramamurthy, R.; DeWitt, D. J.: Buffer-pool Aware Query Optimization. **In:** *CIDR*, 2005, S. 250–261

[RSRK07] Rivoire, S.; Shah, M. A.; Ranganathan, P.; Kozyrakis, C.: JouleSort: a balanced energy-efficiency benchmark. **In:** *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 2007. – ISBN 978–1–59593–686–8, S. 365–376

[SHH10] Schall, D.; Hudlet, V.; Härder, T.: Enhancing Energy Efficiency of Database Applications Using SSDs. **In:** *C* Conference on Computer Science & Software Engineering (C3S2E-10)* (2010), 5, S. 1–9

[SHKR08] Schmidt, K.; Härder, T.; Klein, J.; Reithermann, S.: Green Computing - A Case for Data Caching and Flash Disks? **In:** *ICEIS*, 2008, S. 535–540

[SMS] SeaMicro SM10000-Server Spezifikation, http://www.seamicro.com/sites/default/files/SM10000-64_DataSheet_v1%202.pdf

[SOH09] Schmidt, K.; Ou, Y.; Härder, T.: The promise of solid state disks: increasing efficiency and reducing cost of DBMS processing. **In:** *Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering*. New York, NY, USA : ACM, 2009. – ISBN 978–1–60558–401–0, S. 35–41

[TCP] Spezifikation TPC-H Benchmark 2010, Transaction Processing Performance Council (TPC), http://www.tpc.org/tpch/spec/tpch2.14.0.pdf

[TPC] Spezifikation TPC-E Benchmark 2010, Transaction Proceeding Performance Council (TPC), http://www.tpc.org/tpce/spec/v1.12.0/TPCE-v1.12.0.pdf

[Wer07] Werner, D.: *Taschenbuch der Informatik*. Hanser Verlag, 2007

# A Anhang

## A.1 Caché-konforme SQL-Syntax der 22 TPC-H Abfragen

**TPC-H Abfrage 1:**

```
select
  l_returnflag,l_linestatus,
  sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price,
  sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
  sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
  avg(l_quantity) as avg_qty,
  avg(l_extendedprice) as avg_price,
  avg(l_discount) as avg_disc,
  count(*) as count_order
from
  lineitem
where
  l_shipdate <= DateAdd('d',-90,'1998-12-01')
group by
  l_returnflag, l_linestatus
order by
  l_returnflag, l_linestatus
```

**TPC-H Abfrage 2:**

```
select
  s_acctbal, s_name, n_name, p_partkey,
  p_mfgr, s_address, s_phone, s_comment
from
  part, supplier, partsupp, nation, region
where
  p_partkey = ps_partkey
  and s_suppkey = ps_suppkey
  and p_size = 15
  and p_type like '%BRASS'
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'EUROPE'
```

```
  and ps_supplycost = (
    select
      min(ps_supplycost)
    from
      partsupp, supplier, nation, region
    where
      p_partkey = ps_partkey
      and s_suppkey = ps_suppkey
      and s_nationkey = n_nationkey
      and n_regionkey = r_regionkey
      and r_name = 'EUROPE')
order by
  s_acctbal desc, n_name, s_name, p_partkey
```

**TPC-H Abfrage 3:**

```
select
  l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) as revenue,
  o_orderdate, o_shippriority
from
  customer, orders, lineitem
where
  c_mktsegment = 'BUILDING'
  and c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate < ToDate('1995-03-15','YYYY-MM-DD')
  and l_shipdate > ToDate('1995-03-15','YYYY-MM-DD')
group by
  l_orderkey, o_orderdate, o_shippriority
order by
  revenue desc, o_orderdate
```

**TPC-H Abfrage 4:**

```
select
  o_orderpriority, count(*) as order_count
from
  orders
where
  o_orderdate >= ToDate('1993-07-01','YYYY-MM-DD')
  and o_orderdate < DateAdd('m',3,'1993-07-01')
  and exists (
    select
      *
    from
```

```
      lineitem
    where
      l_orderkey = o_orderkey
      and l_commitdate < l_receiptdate)
group by
  o_orderpriority
order by
  o_orderpriority
```

**TPC-H Abfrage 5:**

```
select
  n_name,
  sum(l_extendedprice * (1 - l_discount)) as revenue
from
  customer, orders, lineitem, supplier, nation, region
where
  c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and l_suppkey = s_suppkey
  and c_nationkey = s_nationkey
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'ASIA'
  and o_orderdate >= ToDate('1994-01-01','YYYY-MM-DD')
  and o_orderdate < DateAdd('yyyy',1,'1994-01-01')
group by
  n_name
order by
  revenue desc
```

**TPC-H Abfrage 6:**

```
select
  sum(l_extendedprice * l_discount) as revenue
from
  lineitem
where
  l_shipdate >= ToDate('1994-01-01','YYYY-MM-DD')
  and l_shipdate < DateAdd('yyyy',1,'1994-01-01')
  and l_discount between .06 - 0.01 and .06 + 0.01
  and l_quantity < 24
```

**TPC-H Abfrage 7:**

```
select
  supp_nation, cust_nation, l_year, sum(volume) as revenue
```

```
from (
  select
    n1.n_name as supp_nation,
    n2.n_name as cust_nation,
    DatePart('yyyy',l_shipdate) as l_year,
    l_extendedprice * (1 - l_discount) as volume
  from
    supplier, lineitem, orders, customer, nation n1, nation n2
  where
    s_suppkey = l_suppkey
    and o_orderkey = l_orderkey
    and c_custkey = o_custkey
    and s_nationkey = n1.n_nationkey
    and c_nationkey = n2.n_nationkey
    and ((n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
      or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE'))
    and l_shipdate between ToDate('1995-01-01','YYYY-MM-DD')
       and ToDate('1996-12-31','YYYY-MM-DD')) as shipping
group by
  supp_nation, cust_nation, l_year
order by
  supp_nation, cust_nation, l_year
```

**TPC-H Abfrage 8:**

```
select
  o_year,
  sum(case
    when nation = 'BRAZIL' then volume
    else 0
  end) / sum(volume) as mkt_share
from (
  select
    DatePart('yyyy',o_orderdate) as o_year,
    l_extendedprice * (1 - l_discount) as volume,
    n2.n_name as nation
  from
    part, supplier, lineitem, orders,
    customer, nation n1, nation n2, region
  where
    p_partkey = l_partkey
    and s_suppkey = l_suppkey
    and l_orderkey = o_orderkey
    and o_custkey = c_custkey
    and c_nationkey = n1.n_nationkey
    and n1.n_regionkey = r_regionkey
```

```
    and r_name = 'AMERICA'
    and s_nationkey = n2.n_nationkey
    and o_orderdate between ToDate('1995-01-01','YYYY-MM-DD')
      and ToDate('1996-12-31','YYYY-MM-DD')
    and p_type = 'ECONOMY ANODIZED STEEL') as all_nations
group by
  o_year
order by
  o_year
```

**TPC-H Abfrage 9:**

```
select
nation, o_year,
sum(amount) as sum_profit
from (
  select
    n_name as nation,
    DatePart('yyyy',o_orderdate) as o_year,
    l_extendedprice * (1 - l_discount) -
        ps_supplycost * l_quantity as amount
  from
    part, supplier, lineitem,
    partsupp, orders, nation
  where
    s_suppkey = l_suppkey
    and ps_suppkey = l_suppkey
    and ps_partkey = l_partkey
    and p_partkey = l_partkey
    and o_orderkey = l_orderkey
    and s_nationkey = n_nationkey
    and p_name like '%green%') as profit
  group by
    nation, o_year
  order by
    nation, o_year desc
```

**TPC-H Abfrage 10:**

```
select
  c_custkey, c_name,
  sum(l_extendedprice * (1 - l_discount)) as revenue,
  c_acctbal, n_name, c_address, c_phone, c_comment
from
  customer, orders, lineitem, nation
where
```

```
  c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate >= ToDate('1993-10-01','YYYY-MM-DD')
  and o_orderdate < DateAdd('m',3,'1993-10-01')
  and l_returnflag = 'R'
  and c_nationkey = n_nationkey
group by
  c_custkey, c_name, c_acctbal, c_phone,
  n_name, c_address, c_comment
order by
  revenue desc
```

**TPC-H Abfrage 11:**

```
select
  ps_partkey, sum(ps_supplycost * ps_availqty) as value
from
  partsupp, supplier, nation
where
  ps_suppkey = s_suppkey
  and s_nationkey = n_nationkey
  and n_name = 'GERMANY'
group by
  ps_partkey having
    sum(ps_supplycost * ps_availqty) > (
      select
        sum(ps_supplycost * ps_availqty) * 0.0001000000
      from
        partsupp, supplier, nation
      where
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'GERMANY')
order by
  value desc
```

**TPC-H Abfrage 12:**

```
select
  l_shipmode,
  sum(case
    when o_orderpriority = '1-URGENT' or o_orderpriority = '2-HIGH'
      then 1 else 0
  end) as high_line_count,
  sum(case
    when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH'
```

```
      then 1 else 0
  end) as low_line_count
from
  orders, lineitem
where
  o_orderkey = l_orderkey
  and l_shipmode in ('MAIL', 'SHIP')
  and l_commitdate < l_receiptdate
  and l_shipdate < l_commitdate
  and l_receiptdate >= ToDate ('1994-01-01','YYYY-MM-DD')
  and l_receiptdate < DateAdd('y',1,'1994-01-01')
group by
  l_shipmode
order by
  l_shipmode
```

**TPC-H Abfrage 13:**

```
select
  c_count, count(*) as custdist
from (
  select
    c_custkey AS c_custkey,
    count(o_orderkey) AS  c_count
  from
    customer left outer join orders on
      c_custkey = o_custkey
      and o_comment not like '%special%requests%'
    group by
      c_custkey) as c_orders
group by
  c_count
```

**TPC-H Abfrage 14:**

```
select
  100.00 * sum(case
    when p_type like 'PROMO%'
      then l_extendedprice * (1 - l_discount)
    else 0
  end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
  lineitem, part
where
  l_partkey = p_partkey
  and l_shipdate >= ToDate('1995-09-01','YYYY-MM-DD')
  and l_shipdate < DateAdd('m',1,'1995-09-01')
```

**TPC-H Abfrage 15:**

```
create view revenue0 (supplier_no, total_revenue) as
  select
    l_suppkey, sum(l_extendedprice * (1 - l_discount))
  from
    lineitem
  where
    l_shipdate >= ToDate('1996-01-01','YYYY-MM-DD')
    and l_shipdate < DateAdd('m',1,'1996-01-01')
  group by
    l_suppkey

select
  s_suppkey, s_name, s_address, s_phone, total_revenue
from
  supplier, revenue0
where
  s_suppkey = supplier_no
  and total_revenue = (
    select
      max(total_revenue)
    from
      revenue0)
order by
  s_suppkey

drop view revenue0
```

**TPC-H Abfrage 16:**

```
select
  p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt
from
  partsupp, part
where
  p_partkey = ps_partkey
  and p_brand <> 'Brand#45'
  and p_type not like 'MEDIUM POLISHED%'
  and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
  and ps_suppkey not in (
    select
      s_suppkey
    from
      supplier
    where
```

```
      s_comment like '%Customer%Complaints%')
group by
  p_brand, p_type, p_size
order by
  supplier_cnt desc, p_brand, p_type, p_size
```

**TPC-H Abfrage 17:**

```
select
  sum(l_extendedprice) / 7.0 as avg_yearly
from
  lineitem, part
where
  p_partkey = l_partkey
  and p_brand = 'Brand#23'
  and p_container = 'MED BOX'
  and l_quantity < (
    select
      0.2 * avg(l_quantity)
    from
      lineitem
    where
      l_partkey = p_partkey)
```

**TPC-H Abfrage 18:**

```
select
  c_name, c_custkey, o_orderkey,
  o_orderdate, o_totalprice, sum(l_quantity)
from
  customer, orders, lineitem
where
  o_orderkey in (
    select
      l_orderkey
    from
      lineitem
    group by
      l_orderkey having sum(l_quantity) > 300)
  and c_custkey = o_custkey
  and o_orderkey = l_orderkey
group by
  c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by
  o_totalprice desc, o_orderdate
```

**TPC-H Abfrage 19:**

```
select
  sum(l_extendedprice* (1 - l_discount)) as revenue
from
  lineitem, part
where (
  p_partkey = l_partkey
  and p_brand = 'Brand#12'
  and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
  and l_quantity >= 1 and l_quantity <= 1 + 10
  and p_size between 1 and 5
  and l_shipmode in ('AIR', 'AIR REG')
  and l_shipinstruct = 'DELIVER IN PERSON')
  or (
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
    and l_quantity >= 10 and l_quantity <= 10 + 10
    and p_size between 1 and 10
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON')
  or (
    p_partkey = l_partkey
    and p_brand = 'Brand#34'
    and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    and l_quantity >= 20 and l_quantity <= 20 + 10
    and p_size between 1 and 15
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON')
```

**TPC-H Abfrage 20:**

```
select
  s_name, s_address
from
  supplier, nation
where
  s_suppkey in (
    select
      ps_suppkey
    from
      partsupp
    where
      ps_partkey in (
        select
          p_partkey
        from
```

```
          part
        where
          p_name like 'forest%')
      and ps_availqty > (
        select
          0.5 * sum(l_quantity)
        from
          lineitem
        where
          l_partkey = ps_partkey
          and l_suppkey = ps_suppkey
          and l_shipdate >= ToDate('1994-01-01','YYYY-MM-DD')
          and l_shipdate < DateAdd("yyyy", 1, '1994-01-01')))
  and s_nationkey = n_nationkey
  and n_name = 'CANADA'
order by
  s_name
```

**TPC-H Abfrage 21:**

```
select
  s_name, count(*) as numwait
from
  supplier, lineitem l1, orders, nation
where
  s_suppkey = l1.l_suppkey
  and o_orderkey = l1.l_orderkey
  and o_orderstatus = 'F'
  and l1.l_receiptdate > l1.l_commitdate
  and exists (
    select * from
      lineitem l2
    where
      l2.l_orderkey = l1.l_orderkey
      and l2.l_suppkey <> l1.l_suppkey)
  and not exists (
    select * from
      lineitem l3
    where
      l3.l_orderkey = l1.l_orderkey
      and l3.l_suppkey <> l1.l_suppkey
      and l3.l_receiptdate > l3.l_commitdate)
  and s_nationkey = n_nationkey
  and n_name = 'SAUDI ARABIA'
group by
  s_name
```

```
order by
  numwait desc,
  s_name
```

**TPC-H Abfrage 22:**

```
select
  cntrycode, count(*) as numcust, sum(c_acctbal) as totacctbal
from (
  select
    substring(c_phone from 1 for 2) as cntrycode, c_acctbal
  from
    customer
  where
    substring(c_phone from 1 for 2) in
      ('13', '31', '23', '29', '30', '18', '17')
      and c_acctbal > (
        select
          avg(c_acctbal)
        from
          customer
        where
          c_acctbal > 0.00
          and substring(c_phone from 1 for 2) in
            ('13', '31', '23', '29', '30', '18', '17'))
      and not exists (
        select * from
          orders
        where
          o_custkey = c_custkey)) as custsale
group by
  cntrycode
order by
  cntrycode
```

# B Erklärung

Hiermit versichern wir, dass wir die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet haben.

Frankfurt am Main, den 16. Juni 2011,

(Tobias Gontermann, Adam Hermann und Marten Rosselli)