Workload Prediction for Energy-Efficient Distributed Database Management Systems

Christopher Kramer

Bachelorarbeit

Technische Universität Kaiserslautern
Fachbereich Informatik
Kaiserslautern, März 2012

Prüfer:
Prof. Dr.-Ing. Dr. h. c. Theo Härder und
Dipl.-Inf. Volker Höfner

„Prognosen sind schwierig, besonders wenn sie die Zukunft betreffen.“
zugeschrieben Mark Twain, Winston Churchill, Karl Valentin, Niels Bohr u.a.

Kurzzusammenfassung

Energieeffizienz gewinnt in der IT und auch im Bereich von Datenbankmanagementsystemen (DBMS) immer mehr an Bedeutung. Das verteilte DBMS WattDB wurde mit dem Ziel bestmöglicher Energieproportionalität entwickelt, indem es die Clustergröße dynamisch der Last anpasst. Damit zusätzliche Knoten rechtzeitig gestartet werden können, ist eine Lastprognose erforderlich. Diese Arbeit entwickelt ein Prognoseverfahren für die Lastprognose von DBMS, das sowohl kurzfristige Trends als auch langfristige Muster in die Prognose einbezieht. Es erreicht geringe Prognosefehler, obwohl es einen pessimistischen Ansatz verfolgt, der in erster Linie darauf zielt, die Last so selten wie möglich zu unterschätzen. Dadurch ermöglicht es Energieeinsparungen, ohne Leistungseinbußen hinnehmen zu müssen.

Abstract

Energy efficiency gains more and more attention in IT and also around database management systems (DBMS). The distributed DBMS WattDB is being developed with the goal of best possible energy proportionality by dynamically adjusting its cluster size to its workload. To start nodes early enough, a workload prediction is needed. This thesis develops a prediction system for workload prediction of DBMS which considers both short-term trends and long-term patterns. It achieves small prediction errors even though it follows a pessimistic approach that makes it first priority to rarely underestimate the load. This way, it enables energy savings without the need to accept performance penalties.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Diskussion um Klimawandel sowie Bestrebungen zur Kostenreduktion haben in der Informationstechnik zu Bemühungen geführt, IT-Systeme möglichst energieeffizient zu entwickeln. Neben möglichst effizienter Hardware können auch Softwarelösungen zu diesem Ziel beitragen, indem sie Hardware je nach Last steuern und gerade nicht benötigte Hardware abschalten.
Verteilte Datenbankmanagementsysteme (DBMS) sind im Bereich der Informationssysteme schon lange bekannt und verbreitet. Das Ziel beim Entwurf dieser Systeme war dabei in der Regel Leistungssteigerung durch Parallelisierung. Im Hinblick auf Energieeffizienz gewinnt das Konzept eines verteilten DBMS dagegen weitere interessante Möglichkeiten: Je nach Last kann ein verteiltes DBMS Knoten zu- oder abschalten und Energie somit nicht nur effizient nutzen, sondern den Energiebedarf auch annähernd proportional zur anfallenden Last halten. Das an der Technischen Universität Kaiserslautern entwickelte verteilte DBMS WattDB [14] wurde genau mit dieser Zielsetzung entworfen. Es führt Anfragen auf schwachen Knoten mit geringem Energiebedarf aus und schaltet je nach Bedarf weitere solcher Knoten zu, um höhere Last verarbeiten zu können.
Natürlich ergeben sich beim Entwurf eines solchen Systems neue Herausforderungen. So benötigen zugeschaltete Knoten eine gewisse Zeit, bis sie gestartet und bereit sind, Anfragen bearbeiten zu können. Wenn das System zusätzliche Knoten erst dann startet, wenn schon zu viel Last anfällt, entstehen enorme Verzögerungen der Antwortzeit bis die neuen Knoten einsatzbereit sind. Begegnet man diesem Problem, indem eine gewisse Anzahl Knoten als Reserve stets einsatzbereit gehalten wird, so schmälert dies die Energieeinsparungen des Systems erheblich. Besser wäre es dagegen, rechtzeitig die zukünftige Last abschätzen zu können, sodass Knoten genau so gestartet werden, dass sie einsatzbereit sind, sobald mehr Last anfällt.
Ziel dieser Arbeit soll es daher sein, auf Basis vergangener Lastverläufe Prognosen über die zukünftige Last eines solchen Datenbanksystems machen zu können. Dabei sollen sowohl Muster in den Lastverläufen der Vergangenheit, als auch kurzfristige Trends in die Prognose einfließen.

Die weitere Arbeit ist wie folgt aufgebaut: Im folgenden Kapitel 2 wird zunächst er-

örtert, warum Energieeffizienz und Energieproportionalität in der IT und insbesondere für Datenbanksysteme wichtig sind und wie dies durch Lastprognosen unterstützt werden kann. Außerdem wird das verteilte energieproportionale DBMS WattDB vorgestellt. Dabei wird insbesondere auf das Monitoring in WattDB und dessen Bedeutung für die hier entwickelte Lastprognose eingegangen. Daraufhin werden Standard-Prognoseverfahren erläutert und aufgezeigt, warum für die Lastprognose eines verteilten Datenbanksystems ein spezielles Prognoseverfahren sinnvoll ist.
In Kapitel 3 wird die im Rahmen dieser Arbeit entworfene Lastprognose detailliert beschrieben. Nachdem die Problemstellung definiert wird, wird zunächst erklärt, auf welche längerfristigen Muster die Lastprognose eingehen soll. Daraufhin wird der Bedarf für eine kurzfristige Trendanalyse erläutert sowie der Lösungsansatz mittels linearer Regression vorgestellt. Nachdem die Lösung der Lastprognose theoretisch skizziert wurde, erklärt Kapitel 3.4 die Implementierung für WattDB. In Kapitel 3.5 wird schließlich eine Evaluation der vorgestellten Lastprognose durchgeführt und der Overhead untersucht.
In Kapitel 4 folgt ein Ausblick auf weitere Optimierungen und die Nutzung der Lastprognose in WattDB.
Die Arbeit schließt in Kapitel 5 mit einem Fazit.

Kapitel 2

Hintergrund

Bevor die Lastprognose für WattDB beschrieben wird, sollen in diesem Kapitel zunächst wichtige Hintergründe erläutert werden. Zunächst wird begründet, warum es sich lohnt, ein energieproportionales DBMS zu entwickeln und warum dazu Lastprognosen nötig sind. Daraufhin wird WattDB vorgestellt und auf die Eigenschaften von WattDB in Bezug auf Energieproportionalität und Monitoring eingegangen. Außerdem werden Standard-Prognoseverfahren vorgestellt und deren Nutzen für die Lastprognose in WattDB diskutiert.

2.1 Energie

Immer wieder ist in den Medien zu lesen, dass der weltweite Energiebedarf und damit der Ausstoß des Treibhausgases CO_2 stetig ansteigt. So sagt eine jüngst veröffentlichte Prognose voraus, dass sich der weltweite Energiebedarf bis zum Jahr 2030 um 39% erhöhen wird [17]. Angesichts solcher Zahlen versuchen alle Branchen, die Energieeffizienz zu steigern. Aber nicht nur des Klimas wegen lohnt es sich, Energie zu sparen. Durch stetig steigende Energiekosten [18] sind Steigerungen der Energieeffizienz mittlerweile auch aus ökonomischer Sicht interessant, denn die Investition in energiesparende neue Produkte und Anlagen kann sich schon nach wenigen Jahren amortisieren.

2.1.1 Energieeffizienz in der IT

Der Einsatz von immer mehr IT-Systemen führt dazu, dass immer mehr Energie durch diese Systeme verbraucht wird. So verbrauchten 2008 nach Schätzungen allein die Server und Rechenzentren in Deutschland ca. 10,1 TWh Strom, was einen Anteil von rund 1,8% am gesamten Stromverbrauch Deutschlands ausmachte [3]. Arbeitsplatzcomputer verbrauchten zusätzlich im Jahr 2010 rund 3,9 TWh Strom [4]. Solche Zahlen verdeutlichen, dass effizientere IT-Systeme einen wichtigen Beitrag zur Reduktion des Energiebedarfs leisten und damit Kosten und CO_2-Ausstoß verringern können. Die daraus resultierenden Bestrebungen der IT-Branche werden oft mit dem Stichwort „Green IT“ bezeichnet.

Um IT-Systeme effizienter zu machen, liegt es zunächst auf der Hand, den Energiebedarf von Hardware zu verringern. Bei allen Hardware-Komponenten sind in den letzten Jahren Anstrengungen mit dem Ziel unternommen worden, Energie effizienter zu nutzen. Als Beispiele können hier effizientere Netzteile mit über 80% Wirkungsgrad (80 Plus® [2]), moderne Mehrkernprozessoren, welche einzelne Kerne dynamisch in Energiesparmodi versetzen können [9], sowie der Ersatz von Festplatten durch Solid-State-Drives [15] genannt werden. Aber auch die Software-Ebene kann einen großen Teil zur Einsparung von Energie beitragen: Angefangen bei Energieverwaltung in Betriebssystemen über Terminal-Server-Systeme bis hin zur Konsolidierung durch Virtualisierungs-Software gibt es zahlreiche Beispiele für Möglichkeiten, durch Software Energie zu sparen.

Gerade auf Servern spielen DBMS eine zentrale Rolle, da die allermeisten Anwendungen Daten in Datenbanken speichern. Viele Server werden sogar ausschließlich zum Betrieb eines DBMS eingesetzt. Somit fällt auch ein großer Teil des Energiebedarfs, der durch Server verursacht wird, auf den Betrieb von DBMS. Aus diesem Grund sind insbesondere Optimierungen dieser Systeme interessant und können in besonderem Maße zur Reduktion des Energiebedarfs beitragen.

2.1.2 Energieproportionalität von DBMS

Wie die meisten Server-Systeme sind auch DBMS oft nur schwach ausgelastet. Trotzdem müssen sie teils enorme Lastspitzen verarbeiten können. Damit Lastspitzen keine Verzögerungen verursachen, werden Server klassischerweise für die zu erwartende Maximallast ausgelegt. Dies führt dazu, dass die Server die meiste Zeit kaum ausgelastet sind. Da die Hardware der Server, insbesondere Arbeitsspeicher, einen hohen lastunabhängigen Grundverbrauch haben, verhält sich der Energiebedarf dieser Server nicht proportional zur anfallenden Last [20]. Um ein DBMS zu entwickeln, dessen Energiebedarf sich proportional zur Last verhält, ist es daher notwendig, Hardware einzusetzen, die je nach Last zu- oder abgeschaltet werden kann. Ein verteiltes DBMS, welches lastabhängig Knoten zu- oder abschalten kann, ermöglicht es im Gegensatz zu einem einzelnen Server, den Energiebedarf annähernd proportional zur Last zu halten.
In Abb. 2.1 ist der Energieverbrauch eines Standard-Servers, der Verlauf bei perfekter Energieproportionalität, sowie der Verlauf eines verteilten DBMS wie WattDB, welches die Knotenanzahl je nach Last regelt, skizziert. Wie in der Grafik verdeutlicht, wird auch ein solches System keine perfekte Energieproportionalität erreichen. Dies liegt daran, dass immer nur ganze Knoten zugeschaltet werden können und mindestens ein Knoten immer betrieben werden muss. Jedoch ist an der Fläche zwischen dem Energieverlauf des Standard-Servers und dem von WattDB zu erkennen, dass ein solches System deutliche Energieeinsparungen erreichen kann. Außerdem nähert sich der Energieverbrauch eines solchen Systems dem perfekten Verlauf umso mehr an, je mehr Knoten im Cluster vorhanden sind und benötigt werden.

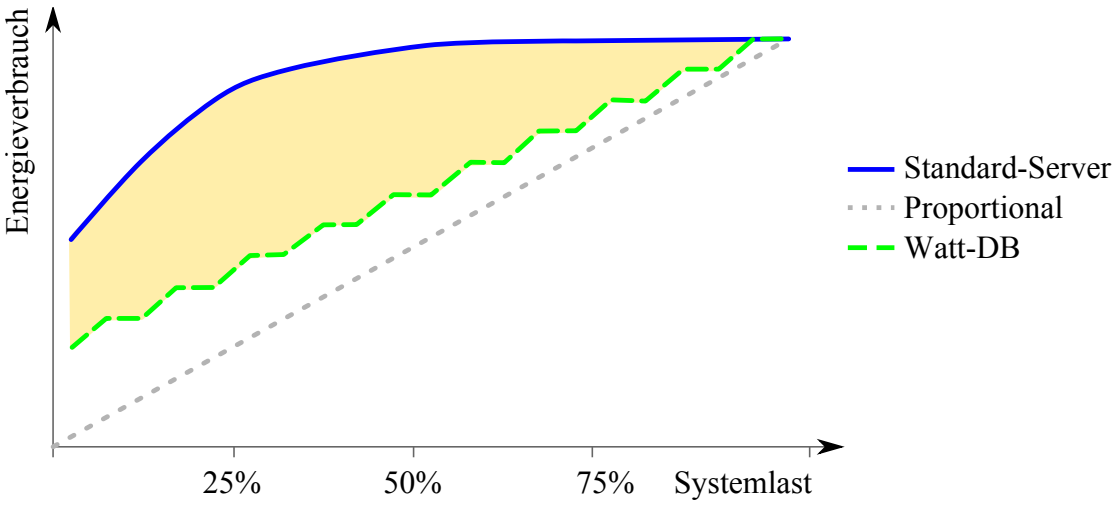

Abbildung 2.1: Energieproportionalität

2.1.3 Lastprognose für bessere Energieproportionalität

Ein DBMS, dessen Energiebedarf sich möglichst proportional zur Last verhält, sei also ein verteiltes DBMS, welches Knoten nach Bedarf zu- oder abschaltet. Zugeschaltete Knoten benötigen allerdings einige Sekunden, bis sie einsatzbereit sind. Würden sie erst gestartet, sobald eine erhöhte Last anfällt, würden zeitweise erhebliche Verzögerungen durch Überlast auftreten. In Abb. 2.2 ist anhand eines beispielhaften Lastverlaufes zu sehen, wie ein verteiltes DBMS ohne Lastprognose Knoten auf Basis der aktuellen Last schalten könnte. Dabei ist klar erkennbar, dass zu spät auf steigende Last reagiert wird und dadurch Überlastsituationen entstehen. Um dies zu verhindern, könnte man eine gewisse Anzahl Reserveknoten bereithalten, um eventuell auftretende Lasterhöhung abfangen zu können. Der Einsatz solcher Reserveknoten würde allerdings die Energieeinsparung des Systems erheblich schmälern. Außerdem ist es schwierig abzuschätzen, wie viele Reserveknoten benötigt werden, da eine sehr plötzlich eintretende Lastspitze mehrere Reserveknoten erfordern kann. Für eine bessere Energieproportionalität ist es daher notwendig, zusätzliche Knoten genau dann zu starten, wenn in wenigen Sekunden eine erhöhte Last zu erwarten ist, sodass der neue Knoten schon einsatzbereit ist, wenn die Last auftritt. Bei starken Lastschwankungen ist es darüber hinaus erforderlich, die Zahl der nötigen Reserveknoten abschätzen zu können. Beides ist nur möglich, indem eine Prognose über zukünftige Last getroffen wird.

Um ein energieeffizientes verteiltes DBMS zu entwickeln, dessen Energiebedarf sich möglichst proportional zur anfallenden Last verhält, ist eine Lastprognose daher zwingend notwendig.

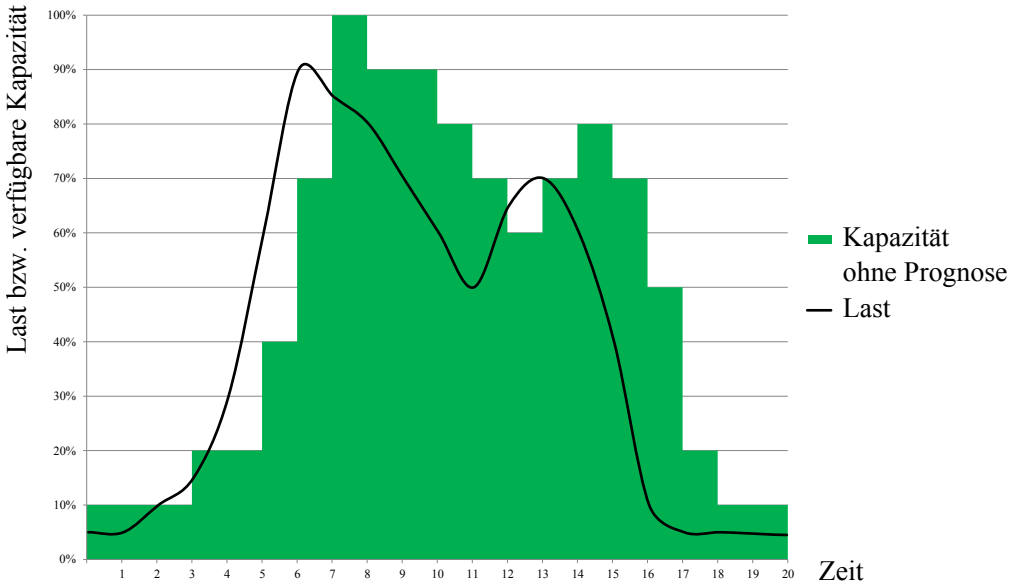

Abbildung 2.2: Last- und Kapazitätsverlauf ohne Reserveknoten und ohne Prognose

2.2 WattDB

WattDB ist ein verteiltes DBMS, welches mit dem Ziel entworfen wurde, Energieproportionalität zu erreichen [7]. Aus diesem Grund läuft WattDB auf einem Cluster schwacher Knoten mit geringem Energieverbrauch pro Knoten. Das aktuelle WattDB-Demo-Cluster sieht wie folgt aus [14]:
Es besteht aus zehn Knoten, die mit je 2 GB RAM und einer Intel® Atom D510 CPU ohne Frequenz-Skalierung ausgestattet sind. Die Netzteile der Knoten sind äußerst effizient und daher 80 Plus® gold zertifiziert. Im Leerlauf benötigen diese Knoten typischerweise ca. 23 Watt, unter Vollast steigt der Verbrauch auf etwa 26 Watt. Aktuell sind zwei der zehn Knoten mit je vier SATA-Festplatten ausgestattet, wobei jede Einzelfestplatte 250 GB Speicherplatz bereitstellt. Aufgrund des Verbrauchs der Festplatten verbrauchen Datenknoten mit Festplatten bis zu 41 Watt, wobei Festplatten mit einem möglichst geringen Energieverbrauch ausgewählt wurden. Das Betriebssystem aller Knoten wird nicht von Festplatten, sondern von USB-Sticks geladen, da diese deutlich weniger Energie benötigen und das Booten durch schnellere Lesezeiten beschleunigt wird.

2.2.1 Energieproportionalität in WattDB

WattDB setzt Energieproportionalität, wie in Abb. 2.1 skizziert, durch das Zu- und Abschalten von Knoten um. Obwohl verteilte DBMS schon lange verbreitet sind, unterstützen es kommerzielle DBMS bisher nicht, Knoten abzuschalten, um Energie zu sparen. Da nicht auf ein bestehendes DBMS zurückgegriffen werden konnte, wurde stattdessen mit WattDB ein neues DBMS von Grund auf mit dem Ziel entworfen, bestmögliche Energieproportionalität zu erreichen.

Das Zu- und Abschalten von Knoten steuert in WattDB ein Masterknoten. Je nach Stromspar-Richtlinie können Knoten in Standby (*suspend-to-RAM*) versetzt werden oder komplett heruntergefahren werden. Knoten im Standby benötigen noch ca. 3 Watt und können per *wake-on-LAN* in weniger als fünf Sekunden gestartet werden [14]. Knoten, die komplett heruntergefahren wurden, verbrauchen zwar fast keine Energie mehr, benötigen aber deutlich länger zum Starten. Obwohl die Systeme auf kurze Bootzeiten optimiert wurden, z.B. indem nicht benötigte Kernel-Module deaktiviert wurden, benötigt ein Knoten zum Starten noch ungefähr zwanzig Sekunden.

2.2.2 Monitoring in WattDB

Angesichts solcher Start-Zeiten können bei steigender Last Überlastsituationen von mehreren Sekunden auftreten, wenn Knoten nicht rechtzeitig gestartet werden. Um dies zu verhindern, ist, wie bereits in Kapitel 2.1.3 erläutert wurde, eine Lastprognose erforderlich. Damit Last prognostiziert werden kann, ist es selbstverständlich nötig, Lastverläufe zu protokollieren. Aus diesem Grund wurde für WattDB ein Monitoring-System entworfen [11]. Das Monitoring-System von WattDB besteht aus Clients, die auf allen Knoten installiert sind und aktuell alle zehn Sekunden verschiedene Lastwerte sammeln und über das Netzwerk an einen Server senden, der auf dem Masterknoten installiert ist. Dieser Server empfängt die protokollierten Daten und speichert sie in einer SQLite-Datenbank [19]. Ermittelt wird eine Vielzahl von Werten zur Auslastung von Prozessor, RAM, Netzwerk und Festplatte.
Die Lastprognose für WattDB kann nun auf die historischen Daten, die vom Monitoring-System aufgezeichnet wurden, zugreifen, um Prognosen über die zukünftige Last machen zu können.

2.3 Prognoseverfahren

Prognosen werden in unterschiedlichsten Bereichen zu unterschiedlichen Zielen eingesetzt: Vom Wetterbericht aus der Meteorologie über Verkehrsprognosen, verschiedene Prognosen aus dem Bereich der Betriebswirtschaften wie z.B. zu Verkaufszahlen oder Gewinnerwartung, demografischen Werten wie der Bevölkerungszahl, Wahlprognosen oder Prognosen zu Aktienkursen. In jeder Prognose geht es darum, Vorhersagen über die Zukunft auf Basis vergangener Daten zu machen. Allerdings unterscheiden sich Prognosen in vielen Punkten. Aus unterschiedlichen Zielsetzungen und Ausgangsdaten ergeben sich unterschiedliche Prognoseverfahren. Prognosen können kurz-, mittel- oder langfristig sein, wobei Prognosen in der Regel umso schlechter werden, je weiter sie in die Zukunft reichen. Darüber hinaus können Prognosen qualitativ, also subjektive Experteneinschätzungen ohne genaue prognostizierte Zahlen, oder quantitativ sein, also auf Basis von Datenmaterial berechnete zahlenmäßige Vorhersagen.

Aus der Vielzahl der Anwendungsbereiche resultiert eine Reihe unterschiedlicher Prognoseverfahren. Dieses Kapitel soll daher einen Überblick über wichtige verbreitete Verfahren

geben und herausstellen, warum diese Verfahren nicht eins zu eins für die Lastprognose eines verteilten DBMS wie WattDB geeignet sind.

2.3.1 Die naive Prognose

Die naive Prognose wird als die wohl einfachste Prognose angesehen. Es werden zwei Typen der naiven Prognose unterschieden. Die *No-Change-Prognose* in Gl. (2.1) geht von einem konstanten Verlauf aus und prognostiziert daher schlicht den letzten Wert x_t als neuen Wert $\hat{x}_{t+1}$ (s. [8]) . Die *Same-Change-Prognose* [8] verwendet den letzten Trend für die Prognose und geht damit von einem linearen Verlauf aus. Dies ist in Gl. (2.2) mathematisch dargestellt.

$$\hat{x}_{t+1} = x_t \tag{2.1}$$

$$\hat{x}_{t+1} = x_t + (x_t - x_{t-1}) \tag{2.2}$$

Die einfache Berechnung der naiven Prognose stellt zwar einen enormen Vorteil dar, in der Regel bieten aber komplexere Prognoseverfahren bessere Ergebnisse und sind dank der Rechenleistung heutiger Computer ebenso schnell berechenbar. Zur Bewertung eines komplexeren Prognoseverfahrens ist ein Vergleich mit der naiven Prognose dagegen äußerst sinnvoll, um abschätzen zu können, ob sich der erhöhte Rechenaufwand lohnt.

Für die Lastprognose eines Datenbanksystems ist die no-change-Prognose wertlos, denn auf Basis dieser Prognose würde stets entschieden, dass weder zusätzlichen Knoten zugeschaltet werden müssen noch Knoten abgeschaltet werden können. Die same-change-Prognose stellt einen sehr einfach berechenbaren Ansatz dar, der für kurzfristige Prognosen erstaunlich gute Ergebnisse liefern kann, jedoch aufgrund von starken Lastschwankungen, die bei Datenbanksystemen sehr häufig sind, oft auch äußerst schlechte Ergebnisse liefern kann.

2.3.2 Gleitende Durchschnitte

Da die naive Prognose nur ein bzw. zwei Datenwerte verwendet, wird sie äußerst stark von statistischen Ausreißern beeinflusst. Um dies zu verbessern, können Durchschnitte aus mehreren Werten gebildet werden. Die Methode der gleitenden Durchschnitte bildet daher das arithmetische Mittel der letzten k Werte, um den neuen Wert $\hat{x}_{t+1}$ zu prognostizieren:

$$\hat{x}_{t+1} = \frac{1}{k} \sum_{i=t-k+1}^{t} x_i \tag{2.3}$$

Augenscheinlich ist auch dieses Verfahren noch sehr einfach und wird daher mitunter auch als weiterer Typ der naiven Prognose angesehen [8]. Grundsätzlich stellt sich bei dieser Methode die Frage nach der Wahl von k. Wird k zu groß gewählt, so werden kurzfristige Trends nicht erkannt, wird es dagegen zu klein gewählt, so gewinnen Ausreißer

ähnlich wie bei der same-change-Prognose ein recht hohes Gewicht.

Um die Last eines Datenbanksystems zu prognostizieren, stellt die Methode der gleitenden Durchschnitte ebenfalls keine zufriedenstellende Lösung dar. Dadurch, dass ein Durchschnitt gebildet wird, unterschätzt diese Prognose die tatsächlich auftretende Last zu häufig, sodass Lastspitzen nicht performant verarbeitet werden können. Darüber ist die Wahl eines sinnvollen k hier äußerst schwierig.

2.3.3 Regressionsanalyse

Die Regressionsanalyse hat das Ziel, eine Funktion zu finden, welche möglichst wenig von den vorhandenen Datenpunkten abweicht. Ist eine solche *Regressionsfunktion* gefunden, kann ein zukünftiger Wert eingesetzt werden, um eine Prognose machen zu können. Wenn das zu erklärende Merkmal (der *Regressand*) nur von einem anderen Merkmal (z.B. der Zeit) abhängt, so spricht man von *Einfachregression* [16]. Hängt es dagegen von mehreren Merkmalen (*Regressoren*) ab, so spricht man von *Mehrfachregression* [16]. So könnte z.B. die Besucheranzahl eines Schwimmbades (Regressand) in Abhängigkeit von der Jahreszeit (erster Regressor) und der Temperatur (zweiter Regressor) untersucht werden.
Neben der Zahl der Regressoren wird unterschieden, ob es sich bei der zu untersuchenden funktionalen Abhängigkeit um eine lineare oder nichtlineare Abhängigkeit handelt [16]. Die allgemeine Form einer Regression ist in Gl. (2.4) dargestellt.

$$\vec{Y} = f(\mathbf{X}) + \vec{\varepsilon} \tag{2.4}$$

Der Regressand $\vec{Y}$ (allgemein ein Vektor) wird durch eine Funktion f von den Regressoren $\mathbf{X}$ (allgemein eine Matrix) und einem Fehlerterm (*Residuum*) $\vec{\varepsilon}$ (allgemein ein Vektor) bestimmt.

2.3.3.1 Lineare Regression

Lineare Regression unterstellt, dass ein linearer Zusammenhang zwischen den Regressoren und dem Regressanden vorliegt. Das sich daraus ergebende Regressionsmodell ist in Gl. (2.5) dargestellt.

$$\vec{Y} = \alpha + \beta\mathbf{X} + \vec{\varepsilon} \tag{2.5}$$

Wird zudem unterstellt, dass der Regressand von nur einem Regressor (z.B. der Zeit) abhängt, so spricht man von *linearer Einfachregression*. Die Vektoren können dann durch Skalare ersetzt werden, wie es in Gl. (2.6) zu sehen ist.

$$y_i = \alpha + \beta x_i + \varepsilon_i \tag{2.6}$$

Ziel der linearen Regression ist es nun, die *Schätzer* α und β so zu wählen, dass die Abweichung der gegebenen Werte y_i von den geschätzten Werten $\hat{y}_i$ möglichst minimal, also das Residuum möglichst gering ist. Durch Einsetzen in die Regressionsfunktion (s. Gl. (2.7)) können dann Prognosen für gegebene x_i gemacht werden.

$$\hat{y}_i = \alpha + \beta x_i \tag{2.7}$$

Die Schätzer α und β sollen also möglichst so gewählt werden, dass insgesamt eine möglichst geringe Abweichung ε entsteht. Würde man die Summe aller Abweichungen minimieren, so würden sich positive und negative Abweichungen aufheben. Um dies zu verhindern, wird die *Methode der kleinsten Quadrate* eingesetzt. Sie minimiert die Summe aller quadrierten Abweichungen Q wie in Gl. (2.8) zu sehen.

$$Q = \sum_{i=1}^{n} \varepsilon^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2 \tag{2.8}$$

Wie aus der Analysis bekannt ist, kann man nun das absolute Minimum von Q bestimmen, indem zunächst nach α und dann nach β abgeleitet und die Ableitungen jeweils Null gesetzt werden (sowie überprüft wird, dass an dieser Stelle die zweite Ableitung positiv ist). Damit erhält man die Schätzer α und β wie in Gl. (2.9) und Gl. (2.10), wobei S_{xy} die Kovarianz zwischen x und y, s_x^2 die Varianz in x, sowie $\bar{x}$ und $\bar{y}$ die arithmetischen Mittel in x bzw. y darstellen. Der Beweis hierzu lässt sich z.B. in [8] nachlesen.

$$\alpha = \bar{y} - \beta \bar{x} \tag{2.9}$$

$$\beta = \frac{S_{xy}}{s_x^2} \tag{2.10}$$

In Abb. 2.3 ist ein Beispiel einer linearen Einfachregression dargestellt. Zwischen den Punkten y_i verläuft die Gerade der Regressionsfunktion $\hat{y}$. An dieser Geraden lässt sich der Ordinatenabschnitt α sowie am Steigungsdreieck die Steigung β erkennen. Außerdem sind die Abweichungen ε_i zwischen der Regressionsgerade und den Datenpunkten markiert.

Die lineare Regression ist für die kurzfristige Trendprognose eines Datenbanksystems ein durchaus sinnvoller Ansatz. Im Vergleich zu den bisherigen Verfahren stellt die Berechnung von Varianz und Kovarianz zwar einen deutlich erhöhten Rechenaufwand dar (der stark davon abhängt, wie viele Datenpunkte genutzt werden), allerdings ist dieser erhöhte Aufwand aufgrund der besseren Ergebnisse selbstverständlich vertretbar, insbesondere in Hinblick auf die heutige Rechenleistung selbst schwacher Computer. Da die Last eines Datenbanksystems aber nicht nur linear, sondern oft auch exponentiell wachsen kann, erscheint es sinnvoll, auch eine nichtlineare (exponentielle) Regression durchzuführen. Für Prognosen, welche in der Lage sein sollen, regelmäßige Muster wie Wochentage oder Feiertage zu erkennen, ist die lineare Regression dagegen völlig ungeeignet.

2.3.3.2 Nichtlineare Regression

Viele Zusammenhänge sind nicht linear, daher sind für diese Zusammenhänge andere Regressionsverfahren nötig. In Gl. (2.11) ist z.B. die Regressionsfunktion für einen quadratischen Zusammenhang gezeigt, wobei die Schätzer α, β und γ ähnlich wie bei der

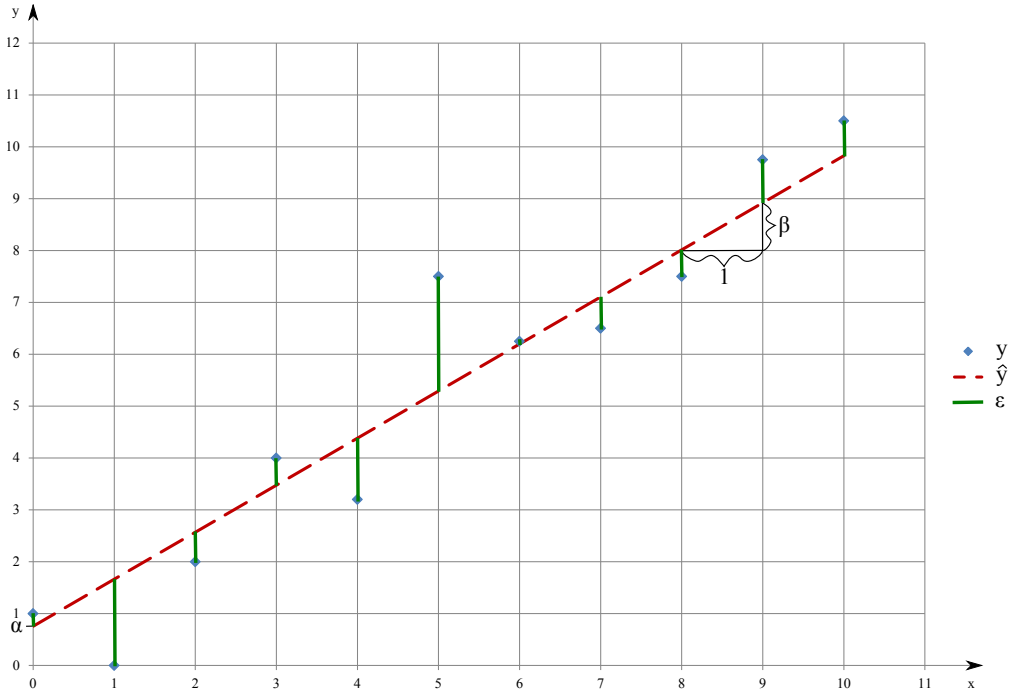

Abbildung 2.3: Lineare Einfachregression

linearen Regression mit Hilfe der Methode der kleinsten Quadrate optimal bestimmt werden können. Auf gleiche Weise kann Regression für z.B. kubische, exponentielle, logarithmische oder sinusförmige Zusammenhänge angewendet werden.

$$\hat{y}_i = \alpha x_i^2 + \beta x_i + \gamma \tag{2.11}$$

Lastverläufe eines Datenbanksystems sind vor allem mittel- und längerfristig in der Regel nichtlinear. Mittelfristig können oft exponentielle, langfristig eher wellenförmige Verläufe auftreten. Daher kann nichtlineare Regression die Lastprognose eines Datenbanksystems unterstützen, indem insbesondere auch exponentielle Trends erkannt werden und prognostiziert werden können. Dies ist insbesondere deshalb wichtig, da die lineare Regression im Fall eines exponentiellen Anstiegs die Last stets deutlich unterschätzen würde und das System daher nicht schnell genug auf den Lastzuwachs reagieren könnte.
Der langfristige Lastverlauf eines Datenbanksystems ist dagegen komplexer als einfache Regressionsfunktionen annähern können. Die Annahme, dass es sich z.B. um eine sinusförmige Funktion handele, wäre zu grob und würde viele Muster des Lastverlaufes ignorieren. Daher ist Regressionsanalyse zur Erkennung langfristiger Muster nicht geeignet.

2.3.4 Künstliche neuronale Netze zur Mustererkennung

Der Lastverlauf eines Datenbanksystems unterliegt i.d.R. gewissen wiederkehrenden Mustern. So kann ein Datenbanksystem beispielsweise nachts deutlich schwächer belastet sein als tagsüber, wochentags stärker als am Wochenende, an Feiertagen besonders schwach oder am letzten Tag des Monats überdurchschnittlich stark. Solche Muster ergeben sich aus der Nutzungsweise dieser Systeme. So können die Beschäftigungszeiten eines Betriebs erheblich die Last eines Datenbanksystems bestimmen. Oder die Last eines Datenbanksystems eines Onlineshops wird genau entgegengesetzt immer dann stärker belastet, wenn Arbeitnehmer frei und daher Zeit zum Onlineshopping haben. Da solche Muster die Last eines Datenbanksystems in erheblichem Maße beeinflussen, lohnt es, sie zur Lastprognose zu erkennen und zu berücksichtigen.

In der Mustererkennung durch Computer werden heutzutage immer häufiger künstliche neuronale Netze eingesetzt [21]. Dabei handelt es sich um „Netzsysteme für Computer, die Daten massiv parallel verarbeiten und zudem lernfähig sind“ [21]. Sie werden häufig als „technische Umsetzung der Gehirnfunktion“ [6] verstanden. Angewendet werden neuronale Netze z.B. zur Text-, Bild- und Spracherkennung, aber eben auch in der Regelungstechnik und zur Zeitreihenanalyse. Insofern wäre die Lastprognose eines Datenbanksystems auf jeden Fall auch eine sinnvolle Anwendung eines künstlichen neuronalen Netzes. Allerdings kann es eine Weile dauern, bis ein neuronales Netz ausreichend trainiert ist, um sinnvolle Prognosen machen zu können. Darüber hinaus steht die Rechenleistung und der Speicherbedarf, der benötigt wird, um eine Prognose mittels eines neuronalen Netzes zu berechnen, kaum in Relation zum angestrebten Nutzen. Angenommen, ein Knoten des verteilten Datenbanksystems wäre permanent damit ausgelastet, Prognosen über zukünftige Last zu berechnen, so könnte man diesen Knoten auch schlicht als Reserveknoten betreiben und auf eine Lastprognose verzichten.

Das Hauptziel ist es nicht, eine perfekte Lastprognose zu erstellen, sondern ein energiesparendes DBMS zu entwerfen. Daher erscheint der Overhead eines künstlichen neuronalen Netzes zu hoch, um für die Lastprognose eines energiesparenden DBMS sinnvoll zu sein.

Kapitel 3

Lastprognose für WattDB

Es wurde erläutert, warum eine Lastprognose für eine gute Energieproportionalität erforderlich ist und welche Standard-Verfahren es für Prognosen gibt. Nun soll im Folgenden das speziell für die Lastprognose eines DBMS wie WattDB entwickelte Prognoseverfahren vorgestellt werden. Dazu werden zunächst die Ziele definiert und dann die grundlegenden Ideen hinter der langfristigen Mustererkennung sowie der kurzfristigen Trendanalyse dargelegt. Nachdem Details der Implementierung erklärt wurden, schließt das Kapitel mit einer umfangreichen Evaluation.

3.1 Problemstellung

In Kapitel 2.3 wurden unterschiedliche Prognoseverfahren und deren Charakteristiken vorgestellt, sowie erläutert, warum diese Verfahren nicht optimal für eine Lastprognose in WattDB sind. Bevor ein neues Prognoseverfahren für WattDB entwickelt wird, soll daher im Folgenden klar definiert werden, welche Charakteristiken dieses Verfahren aufweisen soll. Im Rahmen der Evaluation in Kapitel 3.5 wird überprüft werden, ob das entwickelte Prognoseverfahren dieser Zielsetzung gerecht wird.

3.1.1 Pessimismus

Mit Pessimismus ist gemeint, dass die Prognose die Last eher überschätzen als unterschätzen soll. Wird die Last unterschätzt, so führt dies dazu, dass nicht genügend Knoten aktiv sind, um die tatsächliche Last bearbeiten zu können, und dadurch Verzögerungen in der Bearbeitung entstehen. Wird die Last dagegen überschätzt, sind zwar mehr Knoten als nötig aktiv, dies hat aber keine negativen Auswirkungen auf die Performanz des Systems, es wirkt sich lediglich negativ auf die Energieproportionalität des Systems aus. Trotz pessimistischer Prognose bleibt die Energieproportionalität von WattDB weitaus besser als die eines vergleichbaren Einzel-Servers. Damit die Leistungsfähigkeit des Systems nicht auf Kosten der Energieproportionalität geht, soll die Last daher so gut wie nie unterschätzt werden. Konkret sei das Ziel, dass die Prognose in maximal 10% der Fälle die tatsächliche Last unterschätzen darf. Wird die Last unterschätzt, so darf sie

nur in geringem Maße unterschätzt werden. Bei einem Cluster von 10 Knoten würde eine Unterschätzung von über 10% bedeuten, dass mindestens ein Knoten zu wenig aktiv ist, was zu spürbaren Leistungseinbußen führen würde. Daher soll der Prognosefehler nie mehr als 10% der Maximallast betragen, falls die Last unterschätzt wird. Ob die hier entwickelte Prognose dieses Ziel erreicht, wird in Kapitel 3.5 untersucht.

3.1.2 Geringe Prognosefehler

Ziel einer jeden Prognose ist es natürlich, einen geringen Prognosefehler zu erreichen, also eine möglichst geringe Abweichung zwischen dem prognostizierten und dem tatsächlich auftretenden Wert. Allerdings soll bei der Lastprognose für WattDB das Ziel einer pessimistischen Prognose Vorrang haben, d.h. die Prognose soll eher zum Überschätzen als zum Unterschätzen tendieren, was zu einem durchschnittlich höheren Prognosefehler führt. Trotz Pessimismus sei das Ziel, dass der Prognosefehler im Mittel nicht mehr als 10% der Maximallast beträgt. Da bei einem Cluster von zehn Knoten jeder Knoten in etwa 10% der Gesamtlast verarbeiten kann, lässt sich mit einem Prognosefehler unter 10% die Anzahl benötigter Knoten zuverlässig prognostizieren. Würde der Pessimismus zu einem durchschnittlichen Prognosefehler von mehr als 10% führen, so würde dies bedeuten, dass meist mindestens ein Knoten zu viel prognostiziert wird. In diesem Fall wäre es aber sinnvoller, diesen Knoten stets als Reserveknoten zu betreiben und keine Prognose durchzuführen.

3.1.3 Vertretbarer Overhead

Ein vertretbarer Overhead ist wie in Kapitel 2.3.4 bereits erläutert wichtig, damit eine Lastprognose sinnvoll ist. Würde die Prognose das System zu sehr belasten, wäre es sinnvoller, auf die Prognose zu verzichten und stattdessen Reserveknoten einzusetzen. Der Overhead, den Lastprognose und Monitoring im Cluster erzeugen, sollte daher deutlich geringer sein als die Rechenkapazität eines einzelnen Knotens. Der Overhead wird im Rahmen der Evaluation untersucht werden, um sicherzugehen, dass diese Forderung erfüllt wird.

3.1.4 Berücksichtigung kurzfristiger Trends

Lastverläufe eines Datenbanksystems unterliegen nicht nur langfristigen Mustern, sondern werden auch stark geprägt durch spontane Belastung des Systems. Solche spontanen und damit langfristig unvorhersehbaren Lastspitzen sollen durch die entwickelte Lastprognose möglichst früh erkannt werden, sobald sie absehbar sind. In Kapitel 2.3 wurde bereits erläutert, dass Regressionsverfahren für die kurzfristige Lastprognose eines Datenbanksystems sinnvoll sein können. Daher soll es Ziel sein, dass die entwickelte Lastprognose nicht häufiger als die lineare Regression die Last unterschätzt.

3.1.5 Berücksichtigung langfristiger Muster

Da Lastverläufe wie bereits erwähnt auch häufig von langfristig regelmäßig auftretenden Mustern wie Wochentagen geprägt sind, soll die entwickelte Lastprognose solche Muster auch erkennen und berücksichtigen können. Steigt die Last eines Datenbanksystems z.B. jeden Montag um 8:00 Uhr abrupt von unter 5% auf 95%, wäre dies durch kurzfristige Trendanalyse nur verzögert erkennbar. Wäre das Muster dagegen bekannt, könnte die Prognose diesen Lastanstieg frühzeitig prognostizieren, sodass zusätzliche Knoten rechtzeitig gestartet werden können. In Kapitel 3.5.1 wird untersucht werden, wie gut die entwickelte Prognose solche Muster erkennt und wie lange sie benötigt, um sie zu erkennen.

3.1.6 Schnelle Erkennung langfristiger Muster

Das Problem bei der Mustererkennung ist, dass Systeme eine gewisse Zeit brauchen, bis genügend Daten vorhanden sind, um Muster zu erkennen. Ein jährlich auftretendes Muster kann z.B. erst erkannt werden, wenn es mindestens einmal, i.d.R. mehrfach aufgetreten ist. Selbst hier ergeben sich weitere Probleme: Durch Schaltjahre ändert sich die Zeitspanne, in der jährliche Muster auftreten. Das gleiche Problem stellt sich bei monatlichen Mustern: Da Monate unterschiedlich viele Tage haben, muss ein allgemeines Prognoseverfahren zunächst lernen, wie lang die einzelnen Monate sind, bis monatliche Muster zuverlässig erkannt werden können. Ein allgemeines Prognoseverfahren, wie z.B. eine Prognose durch ein künstliches neuronales Netz, würde daher einige Jahre benötigen, um wichtige Konzepte wie Wochen, Monate, Jahre, Schaltjahre oder Sommerzeit zu lernen und darauf basierende Muster erkennen zu können. Damit eine deutliche Energieersparnis nicht erst nach Jahren erreicht wird, muss das Prognoseverfahren langfristige Muster deutlich schneller erkennen. Es ist also nötig, das Prognoseverfahren mit genügend Startwissen zu versorgen, um wichtige Muster schneller zu erkennen. Das Ziel soll es dabei sein, dass das entwickelte System langfristige Muster erkennt, nachdem sie in der Vergangenheit mindestens dreimal erkennbar waren.

3.2 Längerfristige Mustererkennung

Die größte Herausforderung, die sich aus der genannten Problemstellung ergibt, ist die schnelle Erkennung langfristiger Muster bei gleichzeitig geringem Overhead. Wie bereits in Kapitel 2.3.4 erklärt, erscheint der Einsatz eines künstlichen neuronalen Netzes zur Mustererkennung daher nicht sinnvoll. Auf der Suche nach einem alternativen Prognoseverfahren zur Mustererkennung wurde daher zunächst überlegt, welche Typen von Mustern in Datenbanksystemen typischerweise auftreten und welche davon erkannt werden sollen. Aus dieser Überlegung ergab sich die Idee, gewisse allgemeine Mustertypen vorzudefinieren, anstatt beliebige Muster zu erkennen. So ist es beispielsweise äußerst unüblich, dass in einem Datenbanksystem eine gewisse Last alle 17 Tage auftritt. Deutlich wahrscheinlicher sind dagegen z.B. tägliche, wöchentliche, monatliche oder jährliche Muster. Solche Muster völlig ohne Vorwissen zu erkennen ist dagegen äußerst komplex

und dauert, wie in Kapitel 3.1.6 erläutert, zu lange.
Das hier entwickelte Prognoseverfahren kennt also gewisse Mustertypen, wie tägliche, wöchentliche, monatliche oder jährliche Muster. Wenn es eine Prognose für einen gegebenen Zeitpunkt erstellen soll, prüft es, ob für diesen Zeitpunkt in der Vergangenheit solche Muster erkennbar sind oder nicht. Wenn ein Muster erkannt wird, soll es entsprechend in die Prognose einfließen. Auf die Erkennung von Mustern, welche keinem der vordefinierten Mustertypen folgen, wird aus Effizienz-Gründen verzichtet.
Natürlich muss die Annahme, dass solche Muster stark ausgeprägt sind, nicht immer richtig sein. Ein Datenbanksystem, welches von Benutzern aller Zeitzonen stetig gleich belastet wird, kann z.B. kaum oder gar keine täglich auftretenden Muster aufweisen. Aber selbst bei Benutzern aus allen Zeitzonen werden sich in der Regel Muster ausbilden: Zum einen sind oft nicht in allen Zeitzonen gleich viele Benutzer aktiv. Zum anderen sind Wochenenden zwar in unterschiedlichen Zeitzonen verschoben, trotzdem kann die Last davon abhängig sein, wie viele Zeitzonen gerade Wochenende haben.
Die Annahme der hier genannten Mustertypen gilt nicht nur für von Benutzerlast bestimmte *Online Transaction Processing*-Systeme (OLTP) sondern meistens erst recht auch für von Batch-Betrieb bestimmte *Online Analytical Processing*-Systeme (OLAP): Die Zeitsteuerung von Batch-Anfragen führt zu noch stärkeren Mustern, als es Benutzeranfragen tun können. Da auch solche Systeme in der Regel stündliche, tägliche, wöchentliche, monatliche oder jährliche Berichte erzeugen, folgen auch sie den gleichen Mustertypen wie OLTP-Systeme.

3.2.1 Täglich auftretende Muster

Die wohl wichtigsten Muster in Lastverläufen von Datenbanksystemen sind tägliche. Da viele Datenbanksysteme hauptsächlich von Benutzern einer Zeitzone genutzt werden, ist die Last stark durch Arbeits- und Schlafenszeiten dieser Zeitzone geprägt. Die allermeisten dieser Datenbanksysteme sind dabei während der Arbeitszeiten am stärksten belastet. Man denke etwa an das Datenbanksystem eines Betriebes, dessen Mitarbeiter das System hauptsächlich während der Arbeitszeiten nutzen. Andere Datenbanksysteme sind dagegen genau dann am stärksten belastet, wenn die meisten Arbeitnehmer frei haben, nämlich in den Abendstunden. Dies könnte beispielsweise für ein Datenbanksystem eines Onlineshops oder eines Forums gelten. Viele Systeme nutzen freie nächtliche Kapazitäten zur Bearbeitung von Batch-Jobs. Diese weisen aber wie bereits erwähnt aufgrund ihrer Zeitsteuerung meist noch stärkere tägliche Muster auf als Benutzer-Anfragen.

3.2.2 Wöchentlich auftretende Muster

Neben täglich auftretenden Mustern sind Arbeitszeiten und damit viele Lastverläufe von wöchentlichen Mustern geprägt. Insbesondere an Wochenenden werden Systeme in der Regel anders genutzt als in der Woche. Hier gilt das gleiche wie für tägliche Muster: Manche Systeme können an Wochenenden besonders schwach, andere besonders stark ausgelastet sein. Auch freie Kapazitäten an Wochenenden werden von vielen Systemen für Batch-Anfragen genutzt, aber auch hier führt dies zu noch stärkeren Mustern.

3.2.3 Monatlich auftretende Muster

Monatlich auftretende Muster können z.B. dadurch entstehen, dass am Monatsanfang viele Verbraucher mit Ihrem Monatseinkommen neues Budget bekommen haben, was z.B. bei Onlineshops eine verstärkte Last am Monatsanfang bewirken könnte. Andere Systeme berechnen am Monatsende eventuell einen Monatsbericht, was zu einer verstärkten Last am Monatsende führen kann. Bei solchen Mustern ist es wichtig, zu beachten, dass nicht alle Monate gleich viele Tage haben. Ein System, dass einen Monatsabschlussbericht am letzten Tag des Monats generiert, erzeugt diese Last im Februar schon am 28., im März aber erst am 31. des Monats. Daher sind bei monatlichen Mustern sowohl Muster mit einer gewissen Zeitspanne zum Monatsanfang als auch zum Monatsende zu berücksichtigen.
Eine besondere Form monatlicher Muster sind Muster, die an einen gewissen Wochentag gekoppelt sind, wie etwa „an jedem zweiten Sonntag im Monat“. Solche Muster sind bei monatlichen Veranstaltungen, die an einem bestimmten Wochentag stattfinden sollen, verbreitet. Im betrieblichen Umfeld könnte z.B. eine Besprechung am ersten Montag im Monat dazu führen, dass viele Mitarbeiter ein Datenbanksystem während der Besprechung nicht nutzen und es daher gering ausgelastet ist.

3.2.4 Jährlich auftretende Muster

Jährlich auftretende Muster sind in erster Linie Feiertage, welche in der Regel dazu führen werden, dass Systeme schwächer ausgelastet sind als gewöhnlich. Aber auch jährliche Veranstaltungen wie Betriebsfeiern können jährliche Muster erzeugen. Urlaubszeiten, egal ob durch Betriebsurlaub festgesetzt oder durch Arbeitnehmer frei bestimmt, liegen in der Regel in den Sommermonaten, um Ostern, im Herbst oder um Weihnachten und stellen ebenfalls jährliche Muster dar.
Bei jährlichen Mustern gibt es einfache jährliche Muster, die jedes Jahr am selben Tag auftreten. Insbesondere feste Feiertage wie Weihnachten oder der Tag der Deutschen Einheit sind solche feste jährliche Muster. Darüber hinaus, und deutlich schwieriger zu erkennen, sind dagegen bewegliche jährliche Muster. Als Beispiel wäre hier vor allem Ostern zu nennen und alle beweglichen Feiertage, die an Ostern gekoppelt sind. Darüber hinaus gibt es in anderen Religionen und Kulturen ähnliche Feiertage wie z.B. das Ramadan-Fest, welches sich nach dem islamischen Mondkalender richtet, oder das chinesische Neujahrsfest, welches sich nach dem chinesischen Lunisolarkalender richtet. Durch die Ausrichtung an anderen Kalendersystemen als unserem gregorianischen Kalender werden solche Feiertage zu beweglichen jährlichen Mustern. Ein ähnliches, noch komplexeres Muster, welches sich auf die Last eines Datenbanksystems auswirken kann, sind Schulferien. Neben dauerhaft festgelegten Schulferien wie etwa an Weihnachten, welche einem festen Muster folgen, werden z.B. Sommer- und Herbstferien jedes Jahr erneut von der Kultusministerkonferenz festgelegt. Da solche Beschlüsse keinerlei berechenbarem Muster folgen, ist es nötig, solche Muster jedes Jahr zu definieren, falls sie zuverlässig erkannt werden sollen.
Aufgrund der Vielfalt beweglicher jährlicher Muster soll das entwickelte Prognosever-

fahren es erlauben, derartige Muster für jedes Jahr zu definieren, indem ein Bezugstag festgelegt wird. So soll für Ostern nur der Ostersonntag definiert werden und das Prognoseverfahren automatisch alle daran gekoppelten Feiertage und Muster wie Pfingsten, Christi Himmelfahrt oder auch Rosenmontag erkennen. Ähnlich könnte man auf diese Weise z.B. Ramadan oder Schulferien einpflegen. Das System soll damit flexibel gehalten und nicht zu stark auf ein einzelnes Land, eine einzelne Kultur oder Religion festgelegt werden. Um dem Benutzer den Einsatz dennoch zu erleichtern, werden Tools bereitgestellt, die es erlauben, wichtige Termine wie Ostersonntag automatisch zu befüllen.
Jährlich auftretende Muster sind nicht nur auf der Ebene von Tagen sondern auch Monaten denkbar. So könnte es sein, dass ein Datenbanksystem jedes Jahr zu bestimmten Monaten besonders stark oder schwach ausgelastet ist, z.B. weil mit Saisonware gehandelt wird oder das Vorweihnachtsgeschäft einen erhöhten Umsatz beschert. Solche Muster sollen auch auf Monatsebene erkannt werden.

3.3 Kurzfristige Trendanalyse

Zwar sind Muster in Lastverläufen von Datenbanksystemen häufig erkennbar und daher wichtig zu berücksichtigen. Trotzdem reicht die Erkennung langfristiger Muster für eine zuverlässige Lastprognose nicht aus. Zum einen sind langfristige Muster erst frühestens nach einigen Tagen, einige sogar erst nach Jahren erkennbar, die Lastprognose muss das System aber von Anfang an mit prognostizierten Daten versorgen, damit dieses seine Kapazität an die Last anpassen kann. Zum anderen kann es immer teils gravierende Abweichungen von bisher stark ausgeprägten Mustern geben. Vor allem falls die anfallende Last über der durch langfristige Mustererkennung prognostizierten Last liegt, wäre es fatal, Abweichungen von Mustern zu ignorieren und die Last weiter zu unterschätzen.
Es ist also zwingend notwendig, für die Lastprognose eines Datenbanksystems kurzfristige Trends zu erkennen und diese in die Lastprognose einfließen zu lassen, auch wenn sie stark von langfristigen Mustern abweichen. In Zuge der Evaluation in Kapitel 3.5 wird untersucht werden, inwieweit die hier eingesetzte kurzfristige Trendanalyse die Lastprognose tatsächlich verbessert.

3.3.1 Lineare Regression zur kurzfristigen Trendanalyse

Da es wie in Kapitel 2.3 aufgezeigt bereits einige geeignete Verfahren zur kurzfristigen Trendprognose gibt, soll hier auch kein eigenes Prognoseverfahren entwickelt werden, sondern ein geeignetes Prognoseverfahren an die Problemstellung angepasst werden. In Kapitel 2.3.3 wurde bereits die Regressionsanalyse als für kurzfristige Trendanalyse geeignet herausgestellt, daher soll diese hier auch zum Einsatz kommen. Da ein linearer Lastverlauf am häufigsten anzutreffen ist, wird vereinfachend die lineare Regression genutzt und darauf verzichtet, zusätzlich eine exponentielle Regressionsanalyse durchzuführen. In Kapitel 3.5.2 wird untersucht werden, ob trotz dieser Vereinfachung Ergebnisse erzielt werden, die den in Kapitel 3.1 definierten Zielen entsprechen.
Generell kann eine lineare Regression selbst bei exponentiellem Verlauf gute Ergebnisse

liefern, wenn nicht zu viele vergangene Werte in die Regression mit einfließen. Aus diesem Grund wird in Kapitel 3.5.2 auch evaluiert, wie viele vergangene Datenwerte einbezogen werden sollen, um möglichst gute Ergebnisse zu erhalten.

3.3.2 Pessimistische kurzfristige Trendanalyse

Ziel der Methode der kleinsten Quadrate ist es, einen möglichst geringen Abstand zwischen der Regressionsgeraden und den Punkten zu erreichen. Dies führt dazu, dass in der Regel ungefähr gleich viele Punkte oberhalb wie unterhalb der Geraden liegen. Der Zielsetzung einer pessimistischen Prognose wird die Methode der kleinsten Quadrate daher nicht gerecht. Daher soll eine Gerade gefunden werden, welche zu einer pessimistischeren Prognose führt, indem die Regressionsgerade, welche man durch die Methode der kleinsten Quadrate erhält, angepasst wird.

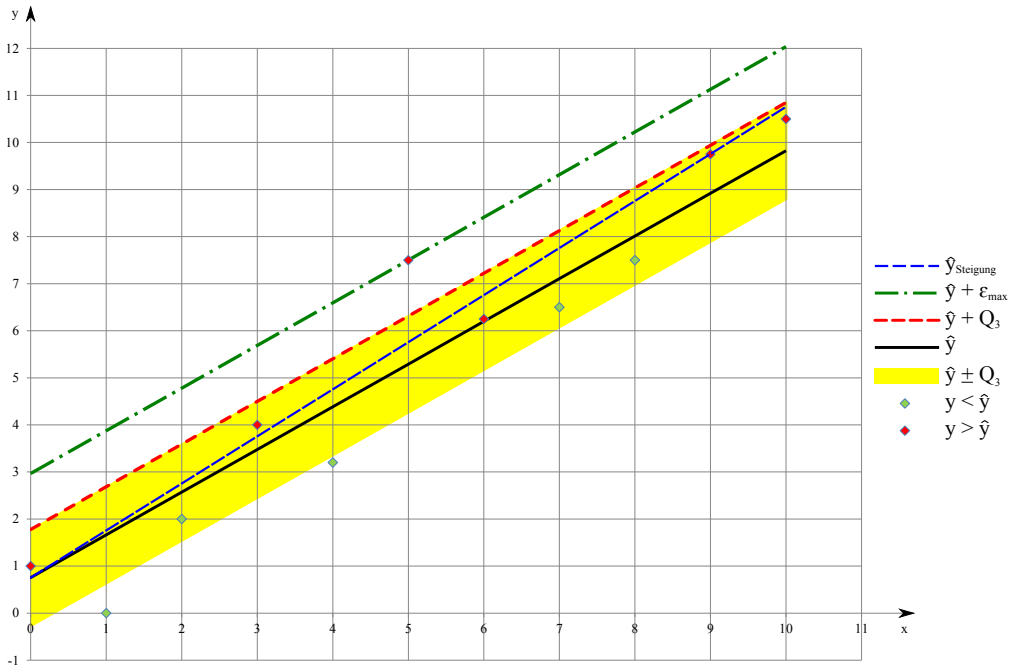

Abbildung 3.1: Pessimistische lineare Einfachregression

Die unterschiedlichen Anpassungen sollen an Abb. 3.1 erklärt werden, welche die gleiche lineare Regression wie Abb. 2.3 darstellt. Im Beispiel sind ungefähr gleich viele Datenpunkte über- wie unterhalb der Regressionsgeraden zu erkennen - für eine pessimistische Prognose werden durch $\hat{y}$ deutlich zu viele Werte unterschätzt.

Generell kann eine Gerade verändert werden, indem entweder der Ordinatenabschnitt oder die Steigung verändert wird. Um die Prognose pessimistischer zu gestalten, müsste mindestens eines von beiden betragsmäßig erhöht werden.

Zunächst soll die Möglichkeit betrachtet werden, die Steigung anzupassen. So könnte man beispielsweise bei positiver Steigung diese pauschal um 10% höher ansetzen und bei negativer Steigung pauschal um 10% niedriger[1]. In Abb. 3.1 ist dies als $\hat{y}_{Steigung}$ eingezeichnet. Bei einem streng linearen Lastverlauf würde eine solche Anpassung dazu führen, dass trotzdem eine gewisse Reserve eingerechnet wird. Wünschenswert wäre es aber, die Prognose umso pessimistischer zu gestalten, je stärker die ursprüngliche Regressionsgerade die vergangenen Werte unterschätzt hat. Dies ist allerdings kaum sinnvoll durch eine Anpassung der Steigung zu erreichen.

Eine Erhöhung des Ordinatenabschnitts dagegen würde es leicht ermöglichen, zu bestimmen, wie viele der vergangenen Werte unterschätzt werden dürfen. Würde man die Gerade um die maximale Abweichung ε_{max} nach oben verschieben, würde dies dazu führen, dass die Gerade den am stärksten unterschätzten Punkt schneidet und alle anderen Punkte darunter liegen (s. Abb. 3.1). Mit dieser Anpassung würde also kein vergangener Wert unterschätzt werden. Da das Maximum von einem statistischen Ausreißer bestimmt werden kann, würde dies zu einer sehr pessimistischen Prognose führen. Andererseits soll das System ebensolche Lastspitzen bewältigen können.
Eine andere weniger ausreißerempfindliche und weniger pessimistische Möglichkeit wäre es, z.B. die Gerade um das obere Quartil der absoluten Abweichungen $Q_3 = |\varepsilon|_{0,75}$ nach oben zu verschieben, wie es ebenfalls in Abb. 3.1 zu sehen ist. Nach Definition des oberen Quartils liegen 75% der Werte darunter. In der Grafik bedeutet das, dass 75% der Werte einen maximalen Abstand von Q_3 haben und damit im markierten Bereich $\hat{y} \pm Q_3$ liegen. Umgekehrt liegen 25% der Werte außerhalb dieses Bereiches. Oberhalb der Oberkante $\hat{y} + Q_3$ dieses Bereichs liegen demnach maximal 25% der Werte, bei gleich vielen Werten ober- wie unterhalb des Bereiches 12,5% der Werte. Die Gerade $\hat{y} + Q_3$ stellt also eine pessimistische Prognose dar, lässt aber zu, dass ein geringer Teil der vergangenen Werte unterschätzt wird.
Eine weitere Möglichkeit zur Erhöhung des Ordinatenabschnitts wäre es, die Gerade um das arithmetische Mittel der absoluten Abweichungen $\overline{|\varepsilon_i|}$ nach oben zu verschieben. Das arithmetische Mittel ist in der Regel kleiner als das obere Quartil und würde daher oft zu einer weniger pessimistischeren Prognose führen. Andererseits ist das arithmetische Mittel im Gegensatz zum oberen Quartil ausreißerempfindlich, was bei starken Ausreißern zu einer pessimistischeren Prognose führen würde.

Welche dieser Anpassung der Regression bei realen Daten zu einer guten pessimistischen Prognose führt, wird in Kapitel 3.5.2.2 untersucht werden.

3.4 Implementierung

Nachdem die grundlegenden Methoden und Verfahren der Lastprognose für WattDB erläutert wurden, werden nun die Details der Implementierung erklärt.

[1] Würde man diesen Ansatz weiter verfolgen wollen, müsste man evaluieren, ob 10% ein guter Wert ist bzw. um wie viel man die Steigung sinnvoll erhöht, ohne einen zu großen Prognosefehler zu erreichen.

Wie in Kapitel 2.2.2 erläutert, sammelt das Monitoring-Framework von WattDB alle zehn Sekunden diverse Kennzahlen zur Last von CPU, Arbeitsspeicher, Netzwerk und Festplatte aller Knoten und speichert sie in einer SQLite-Datenbank. Die Prognose für WattDB ist, wie WattDB selbst, in C++ implementiert. Sie greift mit Hilfe der SQLite-Library auf die SQLite-Datenbank mit den Monitoring-Daten per SQL zu und berechnet aus den von der Datenbank gelieferten Werten die Prognose. Dabei ist die eigentliche Prognose zum größten Teil in SQL implementiert, das C++-Programm verrechnet nur noch die einzelnen Teilprognosen zu einer Gesamtprognose.

3.4.1 Implementierung der längerfristigen Mustererkennung

Die langfristige Mustererkennung besteht im Grunde aus einer SQL-Query und einer C++-Funktion. Die SQL-Query ist aus mehreren **SELECT**-Queries zusammengesetzt, welche per **UNION** verknüft sind. Jede der **SELECT**-Queries repräsentiert einen Mustertypen. Zunächst wird erklärt, wie diese Queries aussehen.

3.4.1.1 Queries der einzelnen Muster

Für jeden Mustertypen gibt es eine **SELECT**-Query, d.h. eine für täglich auftretende Muster, eine für wöchentlich auftretende Muster und so weiter. Die Queries der unterschiedlichen Muster unterscheiden sich in der **WHERE**-Klausel, die jeweils auf alle zu diesem Muster gehörenden Tupel zutrifft.
Die Spalten im Datenbankschema wurden in Hinblick auf die Lastprognose bereits so definiert, dass diese **WHERE**-Klauseln möglichst einfach formuliert und performant ausgewertet werden können. Daher wird der Zeitpunkt des Messwertes nicht in einer Timestamp-Spalte gespeichert, sondern besteht aus einzelnen Spalten für Tag, Monat, Jahr, Stunde, Minute und Sekunde. Dies erlaubt es, durch einfache Selektion ohne die Verwendung von Datumsfunktionen z.B. alle Spalten eines bestimmten Tages oder Monats auszuwählen, was bei Verwendung einer Timestamp-Spalte kompliziertere und langsamere Datumsfunktionen erfordern würde.
Neben Tag, Monat, Jahr, Stunde und Minute werden für die Muster noch weitere Spalten benötigt. Für wöchentlich auftretende Muster wird z.B. der Wochentag benötigt, die beweglichen jährlichen Muster benötigen die Anzahl Tage seit Beginn des Jahres und die monatlichen Muster müssen wissen, der wievielte Wochentag im Monat es ist. Damit solche Informationen nicht zu jedem Datensatz, der vom Monitoring-Framework erfasst wurde, gespeichert und berechnet werden müssen, werden diese Daten für jeden Tag einmal berechnet und per **JOIN** verknüpft.
Das komplette Datenbankschema ist im Anhang ab Seite 61 zu finden.
Hinweis: In die in diesem Kapitel folgenden Beispiel-Queries sind Beispielzahlen so eingesetzt, als würde die Prognose für Samstag, den 31.03.2012, 13:27 durchgeführt. Die richtige Prognose verwendet statt festen Zahlen Datumsfunktionen, welche die aktuell gültigen Werte berechnen. Diese würden allerdings die Übersichtlichkeit verschlechtern und sind daher nur im Anhang ab Seite 63 in der vollständigen SQL-Query zu finden.

Tägliche Muster Eine **WHERE**-Klausel für tägliche Muster soll auf alle Tupel zutreffen, deren Uhrzeit der des zu prognostizierenden Zeitpunktes entspricht und könnte vereinfacht wie folgt aussehen:

SELECT ... **FROM** ... **WHERE hour** = 13 **AND minute** = 27

Wöchentliche Muster Wöchentliche Muster sollen auf alle Tupel zutreffen, deren Wochentag dem des zu prognostizierenden Tages entspricht:

SELECT ... **FROM** ... **WHERE** weekday = 6 */* Wochentag 6 = Samstag */*

Da über den Tagesverlauf oft stark unterschiedliche Last anfällt, ist es sinnvoll, alle Muster zusätzlich auf die Stunde einzugrenzen:

SELECT ... **FROM** ... **WHERE** weekday = 6 **AND hour** = 13

Monatliche Muster Ein monatliches Muster soll auf alle Tupel einschränken, deren Tag im Monat dem des zu prognostizierenden Tages entspricht. Auch hier wird wieder die Stunde dazugenommen:

SELECT ... **FROM** ... **WHERE day** = 31 **AND hour** = 13

Bei Mustern zum Ende des Monats, also z.B. ein Muster für den letzten Tag im Monat, müsste die Anzahl im Monat verbleibender Tage übereinstimmen:

SELECT ... **FROM** ... **WHERE** monthDaysLeft = 0 **AND hour** = 13

Äquivalent für die Muster für den n-ten Wochentag im Monat:

SELECT ... **FROM** ... **WHERE** weekdayThisMonth = 5 **AND hour** = 13

Und für den n-letzten Wochentag im Monat:

SELECT ... **FROM** ... **WHERE** weekdaysLeftThisMonth = 0 **AND hour** = 13

Jährliche Muster Wie bereits in Kapitel 3.2.4 besprochen, gibt es jährliche Muster unterschiedlicher Art. Einfache jährliche Muster sind feste Feiertage oder andere Muster, die jedes Jahr am gleichen Datum auftreten. Auch hier wird zusätzlich noch die Stunde mit dazugenommen:

SELECT ... **FROM** ... **WHERE day** = 31 **AND month** = 3 **AND hour** = 13

Noch einfacher sind jährliche Muster auf Monatsebene:

SELECT ... **FROM** ... **WHERE month** = 3 **AND hour** = 13

Komplizierter sind bewegliche jährliche Muster. Hierfür wird eine eigene Tabelle *specialDay* angelegt, in der jeder besondere Tag des Jahres gespeichert wird. Der Ostersonntag ist im Jahr 2012 am 99. Tag des Jahres und wird daher wie folgt abgespeichert[2]:
(year, dayOfYear, name)=(2012, 99, easterSunday)

[2]Das komplette Datenbankschema ist im Anhang ab Seite 61 zu finden.

Der 31.03.2012 ist der 91. Tag im Jahr, gesucht werden also z.B. alle Tupel die acht Tage vor Ostersonntag aufgezeichnet wurden. Dies ist durch einen Join der Tabelle specialDay mit den aufgezeichneten Daten (monitoredData) wie folgt möglich[3]:

```
SELECT ... FROM monitoredData m, SpecialDay s
WHERE s.year = m.year
        AND m.dayOfYear - s.dayOfYear = 91 - 99
        AND s.name = 'easterSunday'
```

Nun soll aber der Ostertag nicht wie hier fest eingesetzt werden, sondern aus specialDay ausgelesen werden:

```
SELECT ... FROM monitoredData m, SpecialDay s
WHERE s.year = m.year
        AND m.dayOfYear - s.dayOfYear = 91 -
                        (SELECT dayOfYear FROM specialDays s2
                        WHERE s2.year=2012 AND s2.name=s.name)
        AND s.name = 'easterSunday'
```

Da in specialDay beliebig viele unterschiedliche spezielle Tage eingefügt werden können, muss die Abfrage noch für beliebige spezielle Tage verallgemeinert werden:

```
SELECT ..., s.name FROM monitoredData m, SpecialDay s
WHERE s.year = m.year
        AND m.dayOfYear - s.dayOfYear =
                91 - (SELECT dayOfYear FROM specialDays s2
                        WHERE s2.year=2012 AND s2.name=s.name)
GROUP BY s.name
```

Diese Anfrage gruppiert jeweils nach dem „speziellen Tag“ alle Tupel mit dem gegebenen Abstand.

3.4.1.2 Pessimistische Berechnung des Prognosewertes für die Muster

Bisher wurden Tupel durch **WHERE**-Klauseln ausgewählt, es fehlt aber noch die Aggregation der Werte zu einem Prognosewert. Am einfachsten wäre es, den Durchschnitt über das zu prognostizierende Maß zu bilden. Eine einfache Prognose aufgrund wöchentlicher Muster könnte für die CPU-Last z.B. wie folgt aussehen:

```
SELECT avg(cpuLoad) FROM ... WHERE weekday = 6
```

Da die Zielsetzung aber eine pessimistische Prognose ist, eignet sich der Durchschnitt nicht. Das Maximum ist ebenfalls ungeeignet, da bei Mustern, auf die tausende Datenpunkte zutreffen, immer ein besonders hoher statistischer Ausreißer enthalten sein wird. Das obere Quartil ist dagegen ein pessimistisches Maß, welches ausreißerunempfindlich und schnell zu berechnen ist. Nach seiner Definition sind 25% der Datenpunkte größer als das obere Quartil, 75% der Datenpunkte liegen darunter. Unsere Query sähe dann wie folgt aus:

[3] Die Spalte dayOfYear ist tatsächlich im Datenbankschema nicht in der Tabelle monitoredData enthalten, sondern ist erst durch einen Join mit dayInfo verfügbar.

```
SELECT upper_quartile(cpuLoad) FROM ... WHERE weekday = 6
```

Zu beachten ist hier, dass die Funktion **upper_quartile** nicht zu SQLite gehört, aber durch eine Extension [5] leicht nachgerüstet werden kann.
Auf diese Art kann für jeden Mustertyp eine SQL-Query formuliert werden, die das obere Quartil der Werte zurückgibt, die auf dieses Muster passen. In alle bisher genannten Queries setze man dazu einfach das obere Quartil über die zu untersuchende Spalte (hier cpuLoad) ein.

3.4.1.3 Bewertung der Muster

Es sind nun alle Muster als SQL-Query implementiert und jedes Muster gibt einen pessimistischen Prognosewert zurück. Es kann aber sein, dass einige Muster nur schwach ausgeprägt sind oder gar nicht zutreffen. Wenn z.B. der Wochentag gar keinen großen Einfluss auf die Last des Systems hat, so soll das entsprechende Muster auch nicht stark in die Prognose einfließen. Um zu entscheiden, wie gut ein Muster zutrifft, ist ein *Streuungsmaß* erforderlich. Ein solches Streuungsmaß ist die *Varianz*, definiert wie in Gl. (3.1) als der durchschnittliche quadratische Abstand zum arithmetischen Mittel [16].

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{3.1}$$

Die Varianz ist umso größer, je stärker die Datenpunkte vom arithmetischen Mittel abweichen. Ein stark zutreffendes Muster hat also eine geringere Varianz als ein kaum zutreffendes Muster. Die Dimension der Varianz ist das Quadrat der Dimension des Merkmals, was die automatisierte Interpretation schwierig macht. Besser geeignet ist dagegen der auf der Varianz basierende *Variationskoeffizient*. Er ist wie in Gl. (3.2) als Wurzel der Varianz (auch *Standardabweichung* s genannt) geteilt durch das arithmetische Mittel definiert [16].

$$V = \frac{\sqrt{s^2}}{\bar{x}} = \frac{s}{\bar{x}} \tag{3.2}$$

Aus dieser Definition ergibt sich, dass der Variationskoeffizient dimensionslos und genau dann größer als 1 ist, wenn die Standardabweichung größer als das arithmetische Mittel ist.
Für die Mustererkennung der Lastprognose bedeutet dies, dass stark ausgeprägte Muster einen Variationskoeffizienten nahe 0 und schwach ausgeprägte Muster einen Variationskoeffizienten über 1 haben. Daher kann die Lastprognose anhand des Variationskoeffizienten entscheiden, wie stark die einzelnen Muster gewichtet werden müssen.
Die SQL-Queries der Muster werden also um den Variationskoeffizienten erweitert:

```
SELECT upper_quartile(cpuLoad), stdev(cpuLoad)/avg(cpuLoad)
       FROM ... WHERE weekday = 6
```

Für die Berechnung des Variationskoeffizienten wird die Funktion **stdev**, welche die Standardabweichung s berechnet und in der gleichen SQLite-Extension wie **upper_quartile** enthalten ist, sowie die Funktion **avg** zurückgegriffen (s. Gl. (3.2)).

Wenn wir die SQL-Queries aller Muster nun mit **UNION** verknüpfen, erhalten wir für jedes Muster eine Zeile, die einen Prognosewert und den Variationskoeffizienten enthält. Zusätzlich kann man jedem Muster noch in einer Spalte den Namen des Musters als feste Zeichenkette hinzufügen, um die einzelnen Muster wiederzuerkennen. Damit ergibt sich die komplette SQL-Query, wie sie im Anhang ab Seite Seite 63 zu finden ist. Ein beispielhaftes Ergebnis dieser Query ist in Tab. 3.1 zu sehen, wobei der Prognosezeitpunkt hier der 09.03.2012 16:45 Uhr ist.
Zunächst ist in der Spalte „Muster“ erkennbar, dass zwei „spezielle Tage“ eingefügt wurden, nämlich „easterSunday“ (Ostersonntag) und „islamNewYear“ (islamisches Neujahrsfest). Die SQL-Query generiert also wie in Kapitel 3.4.1.1 beschrieben für jeden speziellen Tag ein eigenes Muster. Am Variationskoeffizienten kann nun abgeschätzt werden, wie stark das jeweilige Muster ausgeprägt ist. Den kleinsten Variationskoeffizienten hat in diesem Beispiel das Ostersonntag-Muster, was wohl damit zusammenhängt, dass die Query hier einen Monat vor Ostern ausgeführt wurde. Danach folgen hier die jährlichen Muster mit Tag, Monat und Stunde bzw. nur Monat und Stunde. Diese sind dicht gefolgt vom monatlichen Muster des n-ten Wochentages (es handelte sich um den zweiten Freitag im Monat) und dem reinen wöchentlichen Muster (hier für Freitag). Die restlichen Muster sind schwächer, am schwächsten ist das Muster zum islamischen Neujahr ausgeprägt.

Tabelle 3.1: Beispiel-Ergebnis bei Ausführung der SQL-Query

Muster	Prognosewert	Variationskoeffizient
day + hour	45374	0,1193317374503060
day + month + hour	44877	0,0548789473413432
hour + minute	43245	0,1222452778709580
month + hour	45938	0,0800199996956354
monthDaysLeft + hour	44486	0,1158290674785440
specialDay_easterSunday + hour	45114	0,0497369875568383
specialDay_islamNewYear + hour	43479	0,1263243289628490
weekday + hour	44109	0,0929481327573399
x weekdays left + hour	45898	0,1026434032029750
xth weekday + hour	44604	0,0932576258894913

3.4.1.4 Verrechnung aller Muster zu einem Prognosewert

Jedes Muster liefert einen eigenen Prognosewert (das obere Quartil aller zutreffenden Tupel). Basierend auf diesen Prognosewerten gilt es nun, eine Gesamtprognose basierend auf langfristigen Mustern zu berechnen. Dies wurde in einer einfachen C++-Funktion implementiert. Der vollständige Quelltext dieser Funktion findet sich im Anhang ab Seite 65. Hier soll die Idee und der Algorithmus dieser Funktion dargelegt und an einem Beispiel illustriert werden.
Zunächst werden alle Muster mit einem Variationskoeffizienten ab $0,18$ aussortiert, da

diese Muster ohnehin so schwach sind, dass sie nicht in die Prognose einfließen sollen. Der Wert $0,18$ stelle sich im Verlauf der Evaluation als gute Obergrenze heraus (s. Kapitel 3.5.1). Da der Variationskoeffizient eine dimensionslose Größe darstellt, hängt dieser Wert nicht von dem prognostizierten Maß ab. Je nachdem, ob in sich stark schwankende Muster mit höherem Variationskoeffizienten auch berücksichtigt werden sollen oder nicht, könnte es trotzdem sinnvoll sein, diesen Wert anzupassen. Dies wird im Rahmen dieser Arbeit allerdings nicht weiter untersucht.
Die Idee zur Berechnung eines Gesamt-Prognosewertes ist es nun, einen *gewichteten Durchschnitt* über die Prognosewerte der einzelnen Muster zu errechnen. Wie in Gl. (3.3) gezeigt, wird die Gesamt-Prognose P gebildet, indem das Produkt von Gewicht W_i und Prognosewert P_i aller Muster aufaddiert und durch die Summe aller Gewichte geteilt wird.

$$P = \frac{\sum_{i=1}^{n}(W_i P_i)}{\sum_{i=1}^{n} W_i} \tag{3.3}$$

Die Berechnung der Gewichte erfolgt wie in Gl. (3.4) gezeigt anhand des Variationskoeffizienten.

$$W_i = (0,18 - V_i)^2 \tag{3.4}$$

Die Gewichte sollen umso größer sein, je stärker das Muster ist, also je kleiner der Variationskoeffizient ist. Damit also Muster mit einem kleinen Variationskoeffizient ein großes Gewicht zugewiesen bekommen, werden die Variationskoeffizienten V_i vom maximal erlaubten Variationskoeffizient $0,18$ abgezogen. Die Differenz wird noch quadriert, um starke Muster noch stärker zu gewichten.
Das Beispiel aus Tab. 3.1 wird in Tab. 3.2 erweitert, um die Verrechnung der einzelnen Prognosewerte zu einer Gesamt-Prognose zu veranschaulichen. Die Variationskoeffizienten V_i wurden zunächst auf drei Dezimalstellen gekürzt, um die Übersichtlichkeit zu erhalten. Da alle Variationskoeffizienten hier kleiner als $0,18$ sind, werden alle Muster berücksichtigt. Danach wurden die Gewichte W_i nach Gl. (3.4) berechnet. Die Gesamtprognose wird schließlich als gewichtetes Mittel wie in Gl. (3.5) gezeigt berechnet[4].

$$P = \frac{\sum_{i=1}^{n}(W_i P_i)}{\sum_{i=1}^{n} W_i} = \frac{3489}{0,077689} \approx 44910 \tag{3.5}$$

Um die Gewichte besser vergleichen zu können, wurde zusätzlich in der letzten Spalte der Anteil berechnet, den das einzelne Gewicht W_i an der Summe aller Gewichte ausmacht.

3.4.2 Implementierung der kurzfristigen Trendanalyse

Die kurzfristige Trendanalyse ist komplett in SQL implementiert. Eine einzelne SQL-Anfrage wird auf die Daten angewendet und gibt den gewünschten Prognosewert zurück. Dies bietet sich an, da die in SQL enthaltenen Aggregatfunktionen die Implementierung sehr einfach machen. Zunächst wird erklärt, wie die lineare Einfachregression in SQL

[4]Die Spalte $W_i * P_i$ wurde auf ganze Zahlen gerundet.

Tabelle 3.2: Beispiel für die Berechnung der Gewichte und der Gesamtprognose

Muster	P_i	V_i	W_i	$W_i * P_i$	Anteil W_i
day + hour	45374	0,119	0,003721	169	5%
day + month + hour	44877	0,055	0,015625	701	20%
hour + minute	43245	0,122	0,003364	145	4%
month + hour	45938	0,080	0,010000	459	13%
monthDaysLeft + hour	44486	0,116	0,004096	182	5%
specialDay_easterSunday + hour	45114	0,050	0,016900	762	22%
specialDay_islamNewYear + hour	43479	0,126	0,002916	127	4%
weekday + hour	44109	0,093	0,007569	334	10%
x weekdays left + hour	45898	0,103	0,005929	272	8%
xth weekday + hour	44604	0,093	0,007569	338	10%
Summe			0,077689	3489	

umgesetzt ist. Daraufhin wird die Umsetzung erweitert, um die Prognose pessimistisch zu machen. Abschließend wird angegeben, wie die kurzfristige Trendprognose mit der langfristigen Musterprognose zu einer Gesamtprognose verrechnet wird.

3.4.2.1 Implementierung linearer Einfachregression in SQL

Die kurzfristige Trendanalyse wird wie in Kapitel 3.3 erläutert als lineare Regression umgesetzt. Die theoretischen Grundlagen der linearen Regression wurden bereits in Kapitel 2.3.3.1 erläutert. In Gl. (3.6) wurden in die Regressionsgleichung (2.7) die Schätzer α und β aus Gl. (2.9) bzw. Gl. (2.10) eingesetzt und leicht umgeformt.

$$\hat{y}_i = \alpha + \beta x_i = \bar{y} - \beta\bar{x} + \beta x_i = \bar{y} + \beta(x_i - \bar{x}) = \bar{y} + \frac{S_{xy}}{S_x^2}(x_i - \bar{x}) \tag{3.6}$$

Zur Implementierung der kurzfristigen Trendanalyse wird diese Gleichung nun in SQL ausgedrückt. Da eine kurzfristige Prognose nicht alle, sondern nur die letzten k Werte einbeziehen soll (lookback), müssen diese zunächst wie in Listing 3.1 gezeigt ausgewählt werden.

```
SELECT ... FROM monitoredData
WHERE id > (SELECT MAX(id) FROM monitoredData) - k
```

Listing 3.1: WHERE-Klausel für die kurzfristige Trendanalyse

Welches k sinnvoll ist, soll zunächst nicht festgelegt, sondern im Rahmen der Evaluation in Kapitel 3.5.2 erarbeitet werden. Das dort gefundene k wird dann in die SQL-Query als Konstante eingesetzt.
Nun wird Gl. (3.6) in Listing 3.2 als SQL formuliert.

```
AVG(y) + COVARIANCE(x,y) / VARIANCE(x) * (MAX(id)+s - AVG(x))
```

Listing 3.2: Lineare Regressionsgleichung in SQL

Hier wurde als x_i die ID des zu prognostizierenden Zeitpunkts **MAX**(id)+s eingesetzt. Die Variable s gibt an, wie viele Schritte die Prognose in die Zukunft reichen soll (lookahead). Wenn man davon ausgeht, dass alle 10 Sekunden ein neues Tupel geschrieben wird, würde $s = 1$ bedeuten, dass der Prognosezeitpunkt „in 10 Sekunden“ entspricht. Wenn man die Prognose für „in 20 Sekunden“ benötigt, etwa weil ein Knoten so lange zum Starten braucht, würde man $s = 2$ einsetzen.
Die Varianz wird durch die eingesetzte SQLite-Erweiterung [5] implementiert und kann daher direkt genutzt werden. Die Kovarianz ist dagegen nicht in SQLite oder der SQLite-Erweiterung implementiert und muss durch Standard-Funktionen ersetzt werden. Dazu betrachten wir zunächst die Definition der Kovarianz, die wie in Gl. (3.7) gezeigt mit Hilfe des *Verschiebungssatzes* [10] umgeformt werden kann, um eine einfachere und schnellere Berechnung zu ermöglichen.

$$S_{xy} = \frac{1}{k}\sum_{i=1}^{k}(x_i - \bar{x})(y_i - \bar{y}) = -\bar{x}\bar{y} + \frac{1}{k}\sum_{i=1}^{k} x_i y_i \tag{3.7}$$

Darauf basierend zeigt Listing 3.3, wie die Kovarianz mit Standard-Funktionen von SQL ausgedrückt werden kann.

```
SUM(x * y) / k - AVG(x) * AVG(y)
```

Listing 3.3: Kovarianz ausgedrückt durch Standard-SQL Funktionen

Nun wird in Listing 3.4 die Funktion **COVARIANCE** aus Listing 3.2 durch Listing 3.3 ersetzt.

```
AVG(y) + ( SUM(x * y) / k - AVG(x) * AVG(y) ) /
VARIANCE(x) * (MAX(id)+s - AVG(x))
```

Listing 3.4: Lineare Regressionsgleichung mit ersetzter Kovarianz

Da im Datenbankschema die Spalten nicht x und y heißen, müssen diese noch ersetzt werden. Der Regressor x ist im Datenbankschema die Spalte id und der Regressand y die zu prognostizierende Spalte, also z.B. cpuLoad. In Listing 3.5 wurden die Spalten entsprechend ersetzt.

```
AVG(cpuLoad) + ( SUM(id * cpuLoad) / k - AVG(id) * AVG(cpuLoad) ) /
VARIANCE(id) * (MAX(id)+s - AVG(id))
```

Listing 3.5: Lineare Regressionsgleichung mit Spalten des Datenbankschemas

Nun kann Listing 3.5 in Listing 3.1 eingesetzt werden:

```
SELECT  AVG(cpuLoad) + (SUM(id*cpuLoad)/k - AVG(id)*AVG(cpuLoad))/
        VARIANCE(id) * (MAX(id)+s - AVG(id))
FROM monitoredData
WHERE id > (SELECT MAX(id) FROM monitoredData) - k
```

Listing 3.6: WHERE-Klausel für die kurzfristige Trendanalyse

3.4.2.2 Implementierung der pessimistischen linearen Einfachregression in SQL

Die lineare Einfachregression ist nun implementiert, muss aber noch angepasst werden, um die pessimistische Zielsetzung zu erfüllen. Wie in Kapitel 3.3.2 besprochen wurde, sollen dazu zunächst die Abweichungen ε_i der vergangenen Werte zu der Regressionsgeraden bestimmt werden. Die Regressionsgerade soll dann nach oben geschoben werden, sodass weniger vergangene Werte unter der Geraden liegen. Zur Verschiebung wurden drei Möglichkeiten diskutiert: ε_{max}, ε_{Q3} sowie $\overline{|\varepsilon_i|}$. Für alle drei Möglichkeiten muss zunächst die Berechnung der ε_i implementiert werden. Da $\varepsilon_i = \hat{y}_i - y_i$ gilt, sind zunächst die $\hat{y}_i$ durch Einsetzen in die Regressionsgleichung zu berechnen. Diese Berechnung wurde bereits für $\hat{y}_{t+s}$ durchgeführt, die Berechnung der $\hat{y}_i$ mit $y - k < i \leq t$ erfolgt analog.
In Listing 3.7 ist die Berechnung der ε_i in SQL ausgedrückt. Da jetzt nicht mehr eine Aggregation über alle Zeilen erfolgen soll, sondern k Zeilen mit jeweils einem ε_i berechnet werden sollen, müssen die Aggregat-Funktionen in Subselects verschoben werden. Zunächst wird $\bar{y}$, also **AVG**(cpuLoad) in einem Subselect berechnet (Zeilen 1 und 2). In einem zweiten Subselect wird β berechnet (Zeilen 3-6). Das dritte Subselect in Zeilen 9-10 berechnet $\bar{x}$, also **AVG**(id). Alle Subselects haben die selbe **WHERE**-Klausel wie der umgebende **SELECT**-Ausdruck, der die letzten k Tupel selektiert. Der wesentliche Unterschied zu Listing 3.6 ist, dass anstatt eines x-Wertes (dort **MAX**(id)+s) k unterschiedliche x-Werte id eingesetzt werden (Zeile 7). Darüber hinaus wird vom Wert der Regressionsgeraden $\hat{y}_i$ direkt noch der tatsächliche Wert y_i (bzw. hier cpuLoad) abgezogen, sodass $\varepsilon_i = \hat{y}_i - y_i$ berechnet wird (Zeile 12).

```
1  SELECT   (          SELECT AVG(cpuLoad) FROM monitoredData
2                      WHERE id>(SELECT MAX(id) FROM monitoredData)-k
3           ) + (      SELECT ( SUM(id*cpuLoad)/k - AVG(cpuLoad)*AVG(id))
4                                          / VARIANCE(id)
5                      FROM monitoredData
6                      WHERE id>(SELECT MAX(id) FROM monitoredData)-k
7           ) * ( id -
8                      (
9                      SELECT AVG(id) FROM monitoredData
10                     WHERE id>(SELECT MAX(id) FROM monitoredData)-k
11                     )
12          ) - cpuLoad
13 FROM monitoredData WHERE id>(SELECT MAX(id) FROM monitoredData)-k
```

Listing 3.7: Berechnung der ε_i in SQL

Je nachdem, welcher pessimistische Ansatz gewählt werden soll, muss nun das Maximum, das obere Quartil oder das arithmetische Mittel der Absolutbeträge berechnet werden. Dies wiederum wird dann auf den Prognosewert aufaddiert. Die so entstehende vollständige SQL-Query ist in Listing 3.8 zu sehen. Die Zeilen 4 bis 18 sind mit Listing 3.7 identisch bis auf die Tatsache, dass die Aggregat-Funktion **MAX** zur Berechnung von ε_{max} eingefügt wurde (Zeile 4). Das Ergebnis dieses **SELECT**-Ausdrucks ε_{max} wird auf den Prognosewert $\hat{y}_{t+s}$ addiert, der in Zeilen 1-2 genau wie in Listing 3.6 berechnet wird.

```
1  SELECT AVG(cpuLoad) + ( SUM(id*cpuLoad)/k - AVG(id)*AVG(cpuLoad))
2        / VARIANCE(id) * (MAX(id) +s - AVG(id))
3  + (
4        SELECT MAX(
5              (     SELECT AVG(cpuLoad) FROM monitoredData
6                    WHERE id>(SELECT MAX(id) FROM monitoredData)-k
7              ) + (       SELECT  ( SUM(id*cpuLoad)/k - AVG(cpuLoad)*AVG(id))
8                                  / VARIANCE(id)
9                    FROM monitoredData
10                   WHERE id>(SELECT MAX(id) FROM monitoredData)-k
11             ) * ( id -
12                   (
13                   SELECT AVG(id) FROM monitoredData
14                   WHERE id>(SELECT MAX(id) FROM monitoredData)-k
15                   )
16             ) - cpuLoad
17       )
18       FROM monitoredData WHERE id>(SELECT MAX(id) FROM monitoredData)-k
19 )
20 FROM monitoredData WHERE id>(SELECT MAX(id) FROM monitoredData)-k;
```

Listing 3.8: Pessimistische Trendberechnung in SQL

In diesem Listing kann **MAX** in Zeile 4 durch eine andere Aggregatfunktion wie z.B. **upper_quartile**(**ABS**) oder **AVG**(**ABS**) ersetzt sowie s durch den gewünschten Prognosezeitpunkt ersetzt werden. Außerdem muss in k eingesetzt werden, wie viele vergangene Datenpunkte in die Prognose einbezogen werden sollen. Welche Werte bzw. welche Aggregatfunktion hier am besten geeignet sind, wird im Rahmen der Evaluation in Kapitel 3.5.2 festgelegt.

3.4.3 Verrechnung der kurzfristigen Trendprognose mit der langfristigen Musterprognose

Nachdem sowohl die Prognose auf Basis langfristiger Muster als auch jene auf Basis kurzfristiger Trends implementiert wurde, ist noch zu klären, wie beide zu einer Endprognose verrechnet werden. Dabei sollen zwei Fälle unterschieden werden: Die kurzfristige Prognose kann höher als die langfristige oder niedriger ausfallen.
Fällt sie höher aus, so muss auf jeden Fall die kurzfristige Prognose als Endprognose angesetzt werden. Zum einen verlangt unsere pessimistische Zielsetzung dies, zum anderen können spontane Abweichungen von langfristigen Mustern immer auftreten.
Nicht so klar ist es dagegen, was die Endprognose sein soll, falls die langfristige Prognose höher ausfällt. Würde man hier trotzdem die kurzfristige Prognose ansetzen, würde die langfristige Prognose nie verwendet und langfristige Muster somit komplett ignoriert, was der Zielsetzung widerspricht. Ganz pessimistisch könnte man in diesem Fall die langfristige Prognose ansetzen, was bedeuten würde, dass stets die höhere beider Prognosen genutzt wird.
Angenommen die langfristige Prognose kennt z.B. einen Feiertag noch nicht und prognos-

tiziert daher einen deutlich zu hohen Wert, so würde dies aber bedeuten, dass den ganzen Tag über deutlich mehr Knoten aktiv sind als benötigt, was zu einem deutlich höheren Energieverbrauch führen würde. Um den Energieverbrauch möglichst gering zu halten, ohne langfristige Muster komplett zu ignorieren, sollte in diesem Fall daher das arithmetische Mittel der beiden Prognosewerte die Endprognose bilden. Dies würde bedeuten, dass wegen der langfristigen Muster eine gewisse Reserve an Knoten bereitgehalten wird, immerhin könnte es sein, dass das Muster etwas verspätet noch eintritt. Andererseits wird nicht so viel Kapazität bereitgestellt, wie die Muster prognostizieren, um den Energiebedarf gering zu halten.

3.5 Evaluation

Nachdem in Kapitel 3.1 klare Ziele für die Lastprognose definiert wurden und die Implementierung in Kapitel 3.4 dargelegt wurde, soll nun im Rahmen einer Evaluation überprüft werden, ob die Implementierung den gestellten Zielen gerecht wird. Dazu wird zunächst die langfristige Mustererkennung und anschließend die kurzfristige Trendanalyse untersucht. Das Kapitel schließt mit einer Gesamtevaluation, welche die langfristige Mustererkennung und die kurzfristige Trendanalyse verknüpft betrachtet.

3.5.1 Evaluation der langfristigen Mustererkennung

Die Evaluation der Prognose auf Basis langfristiger Mustererkennung ist äußerst schwierig, weil langfristige Echtdaten über mehrere Jahre benötigt werden. Das WattDB-Monitoring hat bisher noch keine Daten eines produktiven Systems erfasst, daher sind hier keine Daten zur Evaluation verfügbar. Lastverläufe anderer Systeme wären prinzipiell auch bestens geeignet, allerdings konnte kein Zugang zu jahrelangen Lastverläufen von Serversystemen gefunden werden. Aus diesem Grund musste für die Evaluation der langfristigen Mustererkennung auf andere Daten zurückgegriffen werden. Anspruch an die Daten war, dass sie sich über mindestens drei Jahre erstrecken und langfristige Muster wie die in Kapitel 3.2 besprochenen enthalten.
Als brauchbare Daten stellte sich schließlich der Gesamt-Energiebedarf von Großbritannien heraus [12]. Diese Daten sind ab April 2001 bis heute verfügbar, wobei alle halbe Stunde der Gesamtbedarf während dieser halben Stunde erfasst wird. Die Daten enthalten erwartungsgemäß diverse Muster, die den hier besprochenen Mustertypen entsprechen. So ist der Energiebedarf stark von täglichen Mustern geprägt, da offenbar tagsüber durch Industrie ein deutlich höherer Stromverbrauch verzeichnet wird. Dies äußert sich auch in wöchentlichen Mustern, insbesondere einem geringeren Bedarf am Wochenende. Als jährliches Muster zeigt sich unter anderem ein höherer Energiebedarf in den Wintermonaten sowie ein geringerer Energiebedarf an Feiertagen. Insbesondere eignen sich diese Daten daher auch, um zu überprüfen, ob bewegliche jährliche Muster wie Ostern zuverlässig erkannt werden.
Zur Evaluation wurden die Daten der Jahre 2007 bis 2011 in die SQLite-Datenbank importiert. Die SQL-Query wurde zunächst stichprobenartig auf ca. 100 Prognosezeitpunkte

ausgeführt, um eine gute Obergrenze für den Variationskoeffizienten zu bestimmen, wobei sich 0,18 als geeigneter Wert herausstellte (vgl. Kapitel 3.4.1.4). Daraufhin wurde die Prognose auf Basis langfristiger Muster wie in Kapitel 3.4.1 beschrieben für die Jahre 2007 bis 2011 berechnet. Dabei wurde die SQL-Query derart angepasst, dass nur Daten benutzt werden, die vor dem zu prognostizierenden Zeitpunkt liegen. Würde die Prognose auch Datenpunkte miteinbeziehen, die nach dem Prognosezeitpunkt erfasst wurden, würde dies eine unrealistische Evaluation ergeben, da in der Realität nie Datenpunkte aus der Zukunft existieren. Stattdessen soll die Prognose im Jahr 2007 ohne Datenmaterial beginnen und mit immer mehr Daten fortlaufend berechnet werden, sodass das Lernverhalten beobachtet werden kann. Außerdem wurde das Muster „hour + minute“ entfernt, da keine Daten auf Minutenbasis vorhanden sind.
Nun sollen die so prognostizierten Werte mit den tatsächlichen verglichen werden und überprüft werden, ob die Prognose die gegebenen Ziele erreicht.

3.5.1.1 Pessimismus

Zunächst soll untersucht werden, in wie viel Prozent der Fälle die langfristige Prognose den tatsächlichen Wert unterschätzt. Nach der Zielsetzung ist dies in maximal 10% der Fälle erlaubt. Da die Zielsetzung allerdings für die Gesamtprognose gilt und die kurzfristige Trendanalyse die Gesamtprognose bestimmt, falls sie höher ist, sind hier noch deutlich mehr Unterschätzungen erlaubt.

Tabelle 3.3: Evaluation der langfristigen Mustererkennung: Pessimismus

Jahr	Anteil Unterschätzung	Unterschätzung um	Mehr als 10% unterschätzt
2007	45,41%	3,12%	0,43%
2008	42,76%	3,31%	1,44%
2009	12,23%	2,66%	0,29%
2010	28,44%	3,78%	1,51%
2011	5,02%	1,85%	0,02%

In Tab. 3.3 sind Ergebnisse der Evaluation in Hinblick auf Pessimismus zusammengefasst. Der Anteil Unterschätzungen ist im ersten Jahr 2007 mit ca. 45% noch sehr hoch. Hier zeigt sich, dass eine kurzfristige Prognose im ersten Jahr zwingend nötig ist, um zu vermeiden, dass so viele Prognosen die Last unterschätzen. Der Anteil unterschätzender Prognosen sinkt allerdings in den folgenden Jahren deutlich und erreicht im fünften Jahr 2011 5% und liegt damit unter den geforderten 10%. Offenbar wird hier ab dem fünften Jahr die Forderung, dass maximal 10% der Prognosen unterschätzen dürfen, schon allein mit der langfristigen Prognose erreicht.
Die dritte Spalte in Tab. 3.3 gibt Aufschluss darüber, wie stark die vorhandenen Unterschätzungen durchschnittlich sind. Im Jahr 2007 weichen die vorhandenen Unterschätzungen um durchschnittlich 3,12% vom Maximalwert ab. In den folgenden Jahren ist kein klarer Trend erkennbar, allerdings ist auffällig, dass im letzten Jahr 2011 die wenigen Unterschätzungen auch noch gering waren, nämlich durchschnittlich nur 1,85% vom

Maximalwert.
Es wurde noch gefordert, dass Prognosen die Last nie mehr als um 10% des Maximalwerts unterschätzen dürfen. Wie man in der letzten Spalte ablesen kann, erreicht die langfristige Prognose dieses Ziel noch nicht allein, aber auch hier ist auffällig, dass im letzten Jahr nur noch äußerst selten derart starke Unterschätzungen vorkommen.

Insgesamt zeigt sich, dass die langfristige Prognose umso seltener unterschätzt, je mehr Daten sie zur Verfügung hat. Um in den ersten Jahren trotzdem gute Ergebnisse zu erzielen, ist eine zusätzliche kurzfristige Trenderkennung aber zwingend erforderlich.

3.5.1.2 Geringer Prognosefehler

Das Ziel der Gesamtprognose ist es, einen durchschnittlichen Prognosefehler unter 10% zu erreichen. Hier soll zunächst untersucht werden, ob die Prognose auf Basis langfristiger Muster dieses Ziel trotz Pessimismus erreicht. In Tab. 3.4 wird für die Jahre 2007 bis 2011 der durchschnittliche Prognosefehler in Bezug auf das Jahresmaximum aufgeführt. Zum Vergleich wurde daneben der durchschnittliche Prognosefehler der same-change-Prognose (s. Kapitel 2.3.1) ermittelt. Es ist deutlich sichtbar, dass der Prognosefehler eine leicht steigende Tendenz hat, während der Prognosefehler der same-change-Prognose annähernd konstant bleibt. Der wachsende Prognosefehler ist mit dem steigenden Pessimismus erklärbar. Trotz Pessimismus bleibt der Prognosefehler aber unter den geforderten 10%.

Tabelle 3.4: Evaluation der langfristigen Mustererkennung: Prognosefehler in Bezug auf das Jahresmaximum

Jahr	Prognosefehler	Prognosefehler same-change-Prognose
2007	3,66%	2,15%
2008	4,15%	2,23%
2009	5,22%	2,15%
2010	4,47%	2,10%
2011	6,05%	2,19%

3.5.1.3 Vertretbarer Overhead

Der Overhead für die Berechnung der Prognose auf Basis langfristiger Muster spielt eine geringere Rolle, da diese Berechnung höchstens jede Minute, eher sogar nur jede Stunde nötig ist. Das einzige Muster, dass die aktuelle Minute mit einbezieht, ist das tägliche Muster „hour + minute“. Aus Effizienzgründen erscheint es sinnvoll, dieses Muster außer Acht zu lassen und die langfristige Prognose nur stündlich auszuführen und ausschließlich die kurzfristige Trendanalyse alle zehn Sekunden neu zu berechnen. Da die Berechnung selbst auf schwachen Knoten innerhalb von wenigen Sekunden ausgeführt werden kann, fällt dies im Vergleich zur kurzfristigen Trendanalyse weniger ins Gewicht.
Ein weiterer zu beachtender Overhead stellt die für die langfristige Mustererkennung nötige Speicherung von Monitoring-Daten über einen längeren Zeitpunkt dar. Wie in [11]

abgeschätzt ist, beläuft sich die Datenmenge aller vom Monitoring-System gespeicherten Daten auf weniger als 2 GB pro Jahr. Die vom Prognoseverfahren hinzugefügten Tabellen dayInfo und specialDay benötigen nicht mehr als einige KB und können daher vernachlässigt werden. Selbst wenn Daten über zehn Jahre gesammelt werden, wären dies weniger als 20 GB. Es erscheint nicht sinnvoll, Daten über noch längere Zeiträume in die Prognose einfließen zu lassen, da keine Muster über Jahrzehnte zu erwarten sind. Diese Datenmenge kann problemlos auf der Festplatte des Masterknotens gespeichert werden. Es wird also nicht etwa eine weitere Festplatte benötigt, die zusätzliche Energie benötigen würde. Daher entsteht durch die Speicherung kein bedeutender Overhead.

3.5.1.4 Berücksichtigung längerfristiger Muster

Der interessanteste Teil der Evaluation ist es nun herauszufinden, ob die Prognose langfristige Muster zuverlässig erkennt. Dazu sollen beispielhaft einige Graphen untersucht werden, die den Verlauf der Prognose auf Basis langfristiger Muster mit dem der tatsächlichen Werte vergleicht.

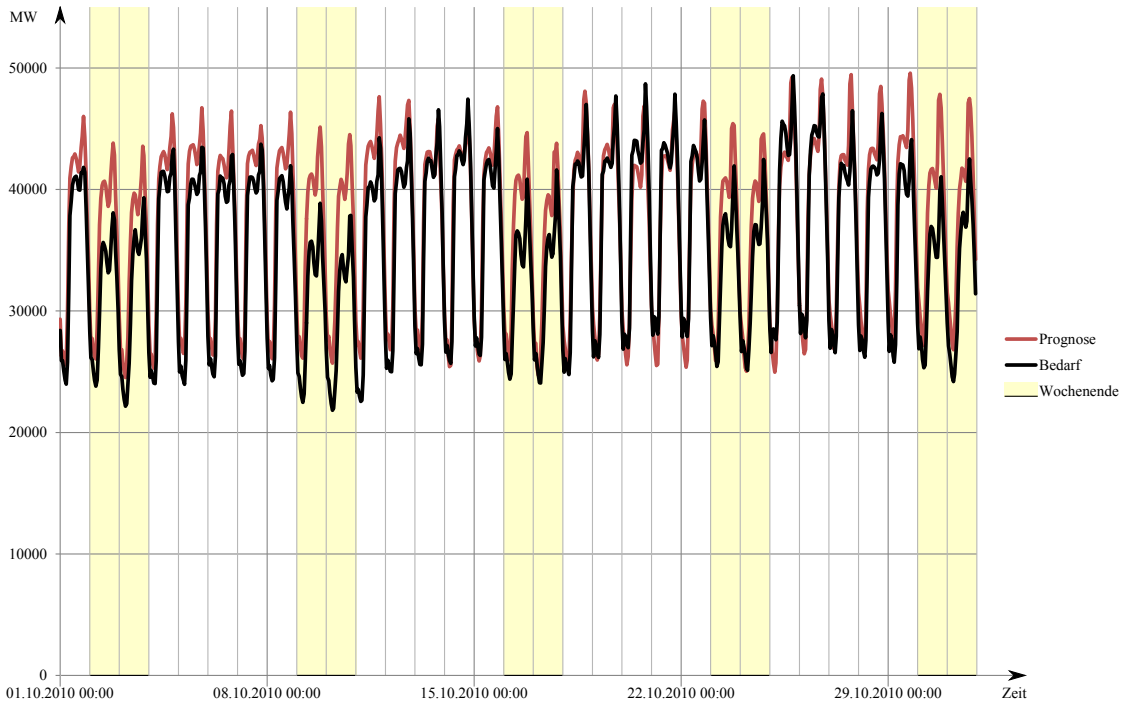

Abbildung 3.2: Prognose für Oktober 2010 auf Basis langfristiger Muster

In Abb. 3.2 ist der Verlauf des Energiebedarfs von Großbritannien sowie der Prognose für Oktober 2010 eingezeichnet. Man erkennt, dass die Prognose nicht stark von dem tatsächlichen Verbrauch abweicht und, wie es Zielsetzung war, eher zum Überschätzen als zum Unterschätzen tendiert.
Mehrere Muster sind in der Grafik erkennbar. Zunächst ist ein tägliches Muster sehr stark

ausgeprägt: Nachts ist der Energiebedarf auf einem Minimum, steigt morgens steil an und erreicht ein lokales Maximum, sinkt am frühen Nachmittag leicht ab, erreicht am frühen Abend das absolute Maximum und fällt zur Nacht wieder steil ab. Dieses Muster ist sowohl im Verlauf des tatsächlichen Bedarfs als auch im Verlauf der Prognose erkennbar. Dies lässt darauf schließen, dass die Prognose dieses Muster erkennt und entsprechend stark gewichtet.
Neben dem täglichen Muster ist ein wöchentliches Muster deutlich erkennbar: An Wochenenden (in der Grafik hell hinterlegt) fällt der Energiebedarf deutlich niedriger aus als unter der Woche. Die Prognose ist an Wochenenden auch niedriger als unter der Woche, wenn auch an einigen Wochenenden die Prognose den Bedarf auffällig überschätzt.
Neben diesem wöchentlichen Muster scheint auch ein monatliches oder jährliches Muster die Wochenenden zu beeinflussen, denn die ersten beiden Wochenenden erreichen Maximalwerte von unter 40.000 MW, während die letzten drei Wochenenden höher ausfallen. Insbesondere der 31. Oktober fällt recht hoch aus, wird aber auch von der Prognose höher als die anderen Wochenenden prognostiziert. Hier lässt sich erahnen, dass monatliche oder jährliche Muster eine Rolle spielen und erkannt werden.

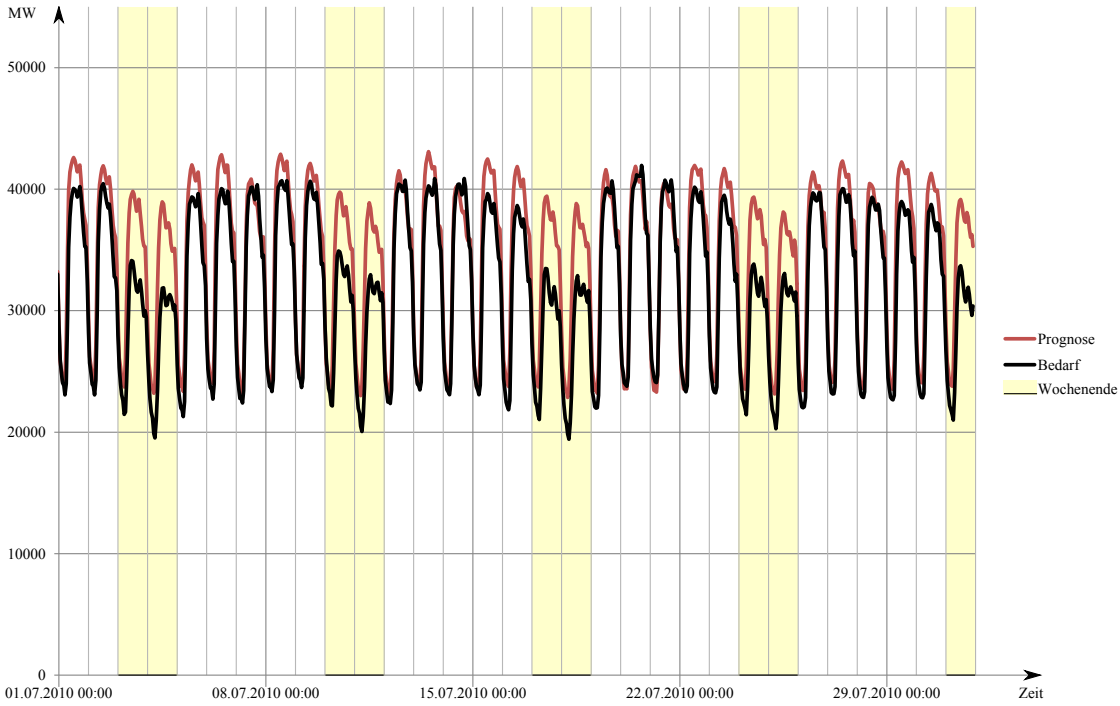

Abbildung 3.3: Prognose für Juli 2010 auf Basis langfristiger Muster

Um dies genauer zu untersuchen, ist zum Vergleich der Monat Juli 2010 in Abb. 3.3 dargestellt. Es fällt zunächst auf, dass der Energiebedarf im Juli deutlich geringer ist als im Oktober: Ende Oktober erreichte der Bedarf fast 50.000 MW während der Bedarf im Juli 40.000 MW kaum überschreitet. Die Prognose hat diesen jährlichen Trend offenbar erkannt, denn auch sie überschreitet die 40.000 MW kaum.

Das tägliche Muster ist im Juli offenbar auch verändert: Das absolute Maximum liegt hier am Mittag, während ein leichtes lokales Maximum am Nachmittag erkennbar ist. Diese Veränderung spiegelt sich allerdings auch in der Kurve der Prognose wieder, was darauf schließen lässt, dass die Prognose das Muster hier korrekt erkennt.
Auch das wöchentliche Muster, dass an Wochenenden der Bedarf deutlich geringer ist, ist im Juli stark ausgeprägt und wird von der Prognose erkannt, wenn auch nicht in vollem Maße.
Es bleibt noch zu untersuchen, ob bewegliche jährliche Muster wie etwa Ostern von der Prognose erkannt und berücksichtigt werden. In Abb. 3.4 ist der Verlauf von Prognose und Bedarf für April 2010 dargestellt, wobei hier die Ostertage von Karfreitag bis Ostermontag hell unterlegt wurden. Man kann erkennen, dass der Energiebedarf zu Ostern niedriger als gewöhnlich ist. Dies trifft weniger auf Ostersamstag und Ostersonntag zu, da der Bedarf am Wochenende ohnehin für gewöhnlich niedriger ist. An Karfreitag und Ostermontag ist der Bedarf dagegen ungefähr so hoch wie an einem normalen Wochenende – man sieht dem Bedarf das „verlängerte Wochenende“ regelrecht an. Die Prognose an diesen Tagen fällt wie gewünscht niedriger aus als an normalen Wochentagen, was dafür spricht, dass die Prognose auch dieses Muster korrekt erkennt.

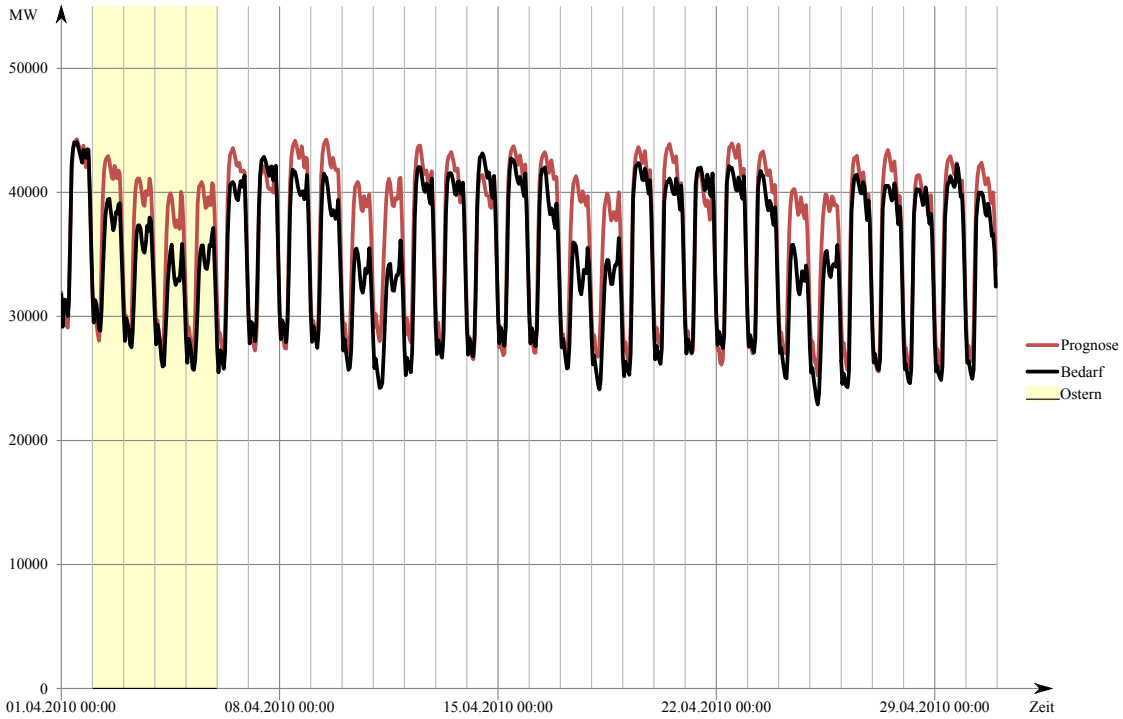

Abbildung 3.4: Prognose für April 2010 auf Basis langfr. Muster

3.5.1.5 Schnelle Erkennung langfristiger Muster

Es wurde beispielhaft am Jahr 2010 gezeigt, dass die Prognose Muster wie gewünscht erkannt hat. Dies zeigt, dass bereits im vierten Jahr Muster zuverlässig erkannt werden. Es bleibt noch zu untersuchen, wie gut Muster in den ersten drei Jahren und insbesondere im ersten Jahr erkannt wurden.

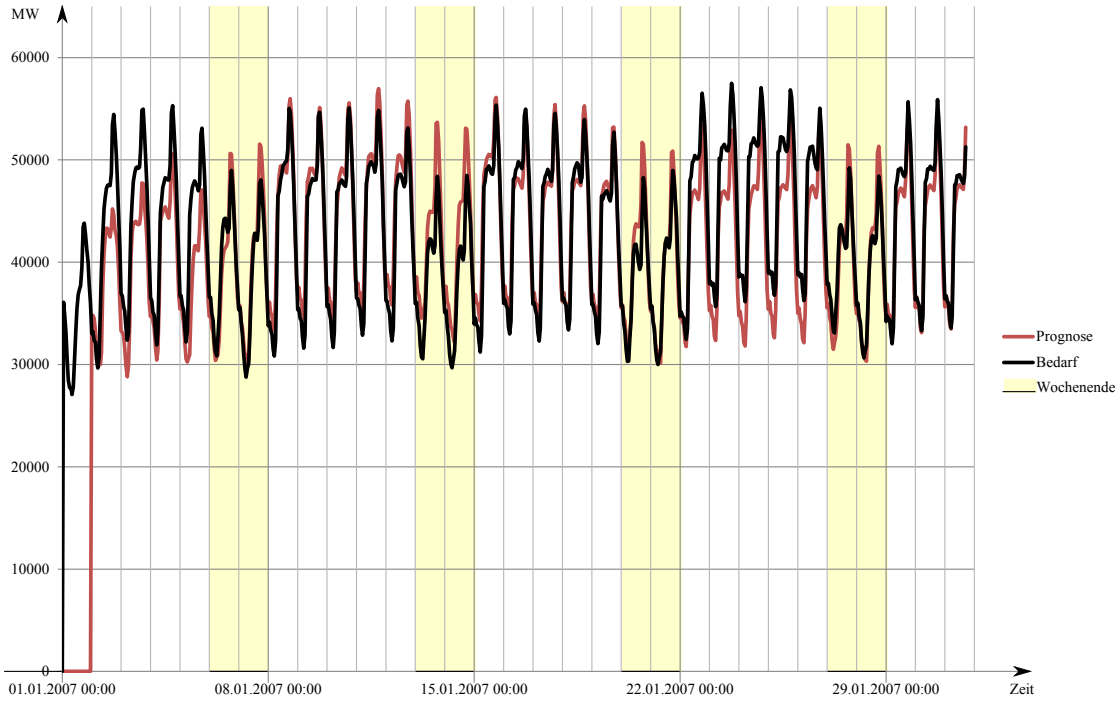

Abbildung 3.5: Prognose für Januar 2007 durch Mustererkennung

Zunächst soll der erste Monat, Januar 2007, betrachtet werden (s. Abb. 3.5). Der erste Tag kann von der langfristigen Prognose überhaupt nicht prognostiziert werden, da für keines der Muster passende Tupel in der Datenbank vorhanden sind. Am ersten Tag ist daher eine kurzfristige Trendanalyse erforderlich, um überhaupt etwas prognostizieren zu können. Da der erste Januar ein Feiertag mit vergleichsweise niedrigem Energiebedarf ist, lernt die Prognose zunächst eine Ausnahme, was dazu führt, dass der Bedarf der ersten Woche unterschätzt wird. Schon in der zweiten und dritten Woche weicht die Prognose aber nicht mehr stark vom tatsächlichen Energiebedarf ab. Hier zeigt sich, dass tägliche und wöchentliche Muster sehr schnell gelernt werden. In der vierten Woche ist dagegen der Energiebedarf entgegen der Prognose etwas höher, was wahrscheinlich durch monatliche oder jährliche Muster prognostizierbar wäre.
Insgesamt ist der Januar 2007 zwar bereits erstaunlich gut prognostiziert, die wenigen Daten drücken sich aber vor allem in einem schwach ausgeprägten Pessimismus und daher recht vielen Unterschätzungen aus.
In Abb. 3.6 ist der April 2007 dargestellt. Hier ist ein sehr deutliches Beispiel eines

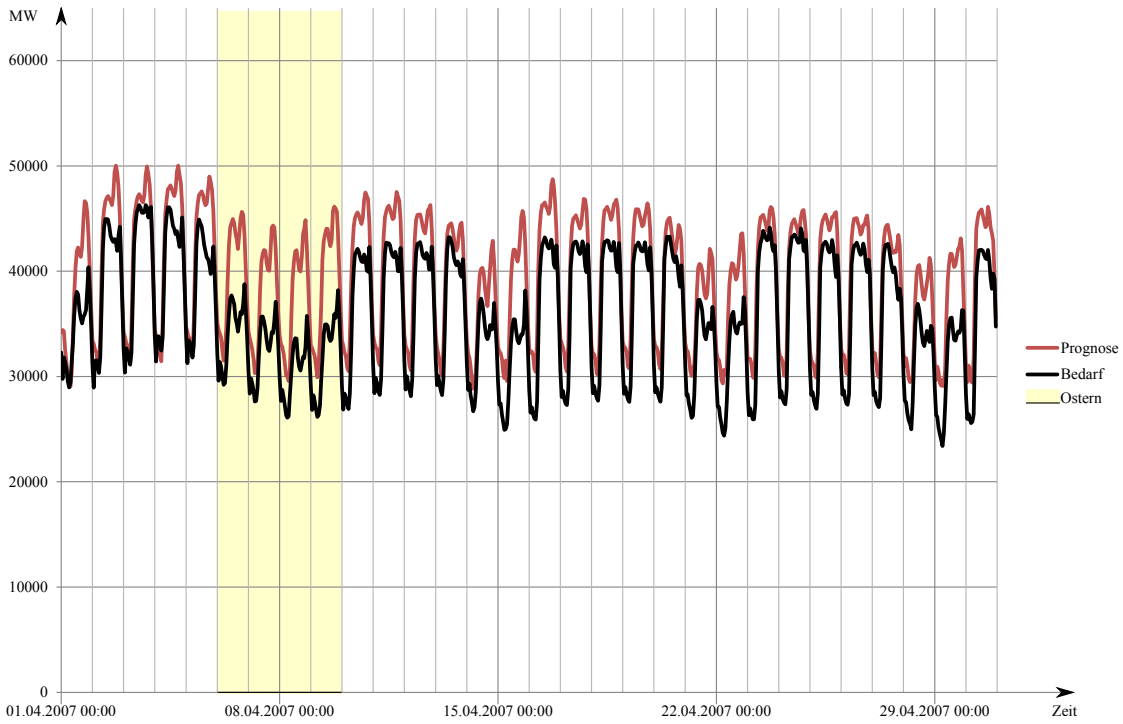

Abbildung 3.6: Prognose für April 2007 auf Basis langfr. Muster

nicht erkannten jährlichen Musters erkennbar: Die Ostertage (wieder hell unterlegt) werden deutlich überschätzt. Da dieses Muster im April 2010 recht gut erkannt wurde (s. Abb. 3.4) zeigt sich hier, dass das Erkennen jährlicher Muster einige Jahre, aber nicht Jahrzehnte, benötigt.

3.5.2 Evaluation der kurzfristigen Trendanalyse

Die Evaluation der kurzfristigen Trendanalyse konnte mit Echtdaten eines produktiven Datenbanksystems durchgeführt werden. Für die Evaluation wurde die Prozessorlast eines Datenbankservers mit installiertem Microsoft SQL Server der SPH AG über acht Tage genutzt, wobei alle zehn Sekunden ein Datenpunkt genommen wurde. Das genaue Setup unter dem diese Daten erhoben wurden ist in [13] nachlesbar. Für alle hier durchgeführten Evaluationen wurde der lookahead $s = 2$ gewählt, d.h. es soll die Last in zwanzig Sekunden prognostiziert werden. Da dies in etwa der Zeit entspricht, die ein WattDB-Knoten benötigt, um aus dem ausgeschalteten Zustand zu starten, wird dies in der Praxis die häufigste Anwendung finden.

3.5.2.1 Bestimmung des lookback Parameters k

Ziel der Evaluation der kurzfristigen Trendanalyse ist es zunächst, einen geeigneten Parameter k zu finden sowie einen der diskutierten Pessimismus-Ansätze auszuwählen. Da

diese Ansätze lediglich eine Anhebung der Prognose zu Folge haben, sind sie für die Suche des k irrelevant. Daher soll zuerst ein geeignetes k für die nicht-pessimistische Trendanalyse gefunden werden und dann anhand dieses k die unterschiedlichen Pessimismus-Ansätze untersucht werden.
Wenn alle zehn Sekunden ein Datenpunkt genommen wird, bedeutet $k = 6$ beispielsweise, dass die Datenpunkte der letzten Minute verwendet werden. Was passiert, wenn man z.B. $k = 36$ wählt und damit Daten der letzten sechs Minuten einbezieht, ist in Abb. 3.7 zu sehen.

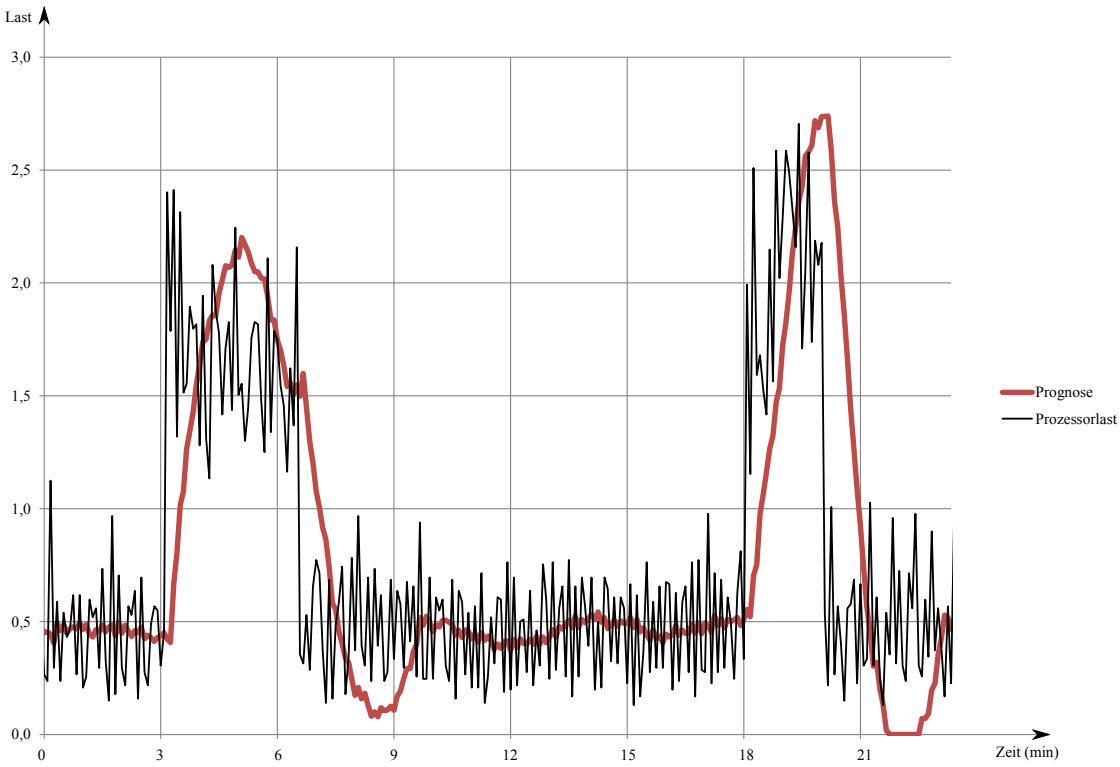

Abbildung 3.7: Kurzfristige Trendanalyse mit k=36 (6 Minuten)

Die lineare Regression über 36 Datenpunkte führt zu einer starken Glättung der Prognose, sodass diese kleinere Schwankungen völlig ignoriert. Dies ist prinzipiell wünschenswert, allerdings wird ein steiler Anstieg wie bei Minute 3 oder Minute 18 so nur verspätet und vermindert prognostiziert. Darüber hinaus führt ein starker Abfall wie bei Minute 20 dazu, dass der fallende Trend noch über Minuten hinaus in die Prognose einfließt obwohl die Last wieder stabil ist. In diesem Fall führt dies sogar dazu, dass die Prognose bei Minute 22 für einige Minuten keinerlei Last prognostiziert und damit die Last deutlich unterschätzt.
Eine lineare Regression über so viele zurückliegende Datenpunkte ist also genau dann besonders schlecht, wenn die kurzfristige Prognose wichtig ist, nämlich wenn kurzfristig starke Lastspitzen auftreten. Darüber hinaus bedeutet das Berechnen der Regression

über viele Datenpunkte auch einen höheren Overhead, da die Berechnung von Durchschnitten, Varianz und Kovarianz umso rechenaufwändiger wird, je mehr Datenpunkte sie umfasst.

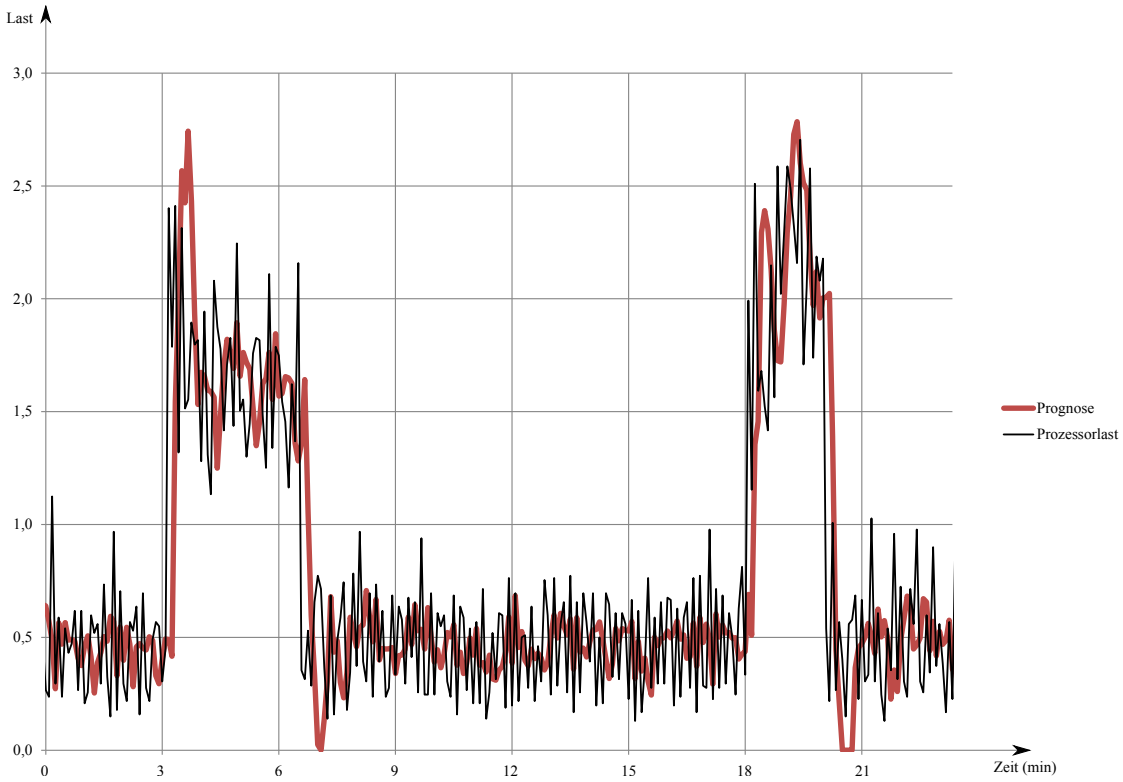

Abbildung 3.8: Kurzfristige Trendanalyse mit k=8 (1 Minute 20 Sekunden)

In Abb. 3.8 ist zum Vergleich die Prognose mit $k = 8$ eingezeichnet. Man erkennt klar, dass die Prognose hier weniger stark geglättet ist und auch den Schwankungen der Last folgt. Kleinere Schwankungen der Prognose haben allerdings ohnehin keine großen Auswirkungen auf das Verhalten des DBMS: Da nur ganze Knoten zu- oder abgeschaltet werden können, führen sie zu keiner Veränderung der Clustergröße. An den Stellen, an denen ein starker Anstieg oder starker Abfall erkennbar ist, reagiert die Prognose hier frühzeitiger. Natürlich kann die kurzfristige Prognose einen Anstieg erst prognostizieren, wenn er beginnt, wodurch sie stets leicht verspätet reagiert. Dies ist auch der Grund, warum eine zusätzliche langfristige Mustererkennung erforderlich ist.
Für eine kurzfristige Trendanalyse ist es also sinnlos, Datenpunkte mit einzubeziehen, die mehr als zwei Minuten alt sind. Um das optimale k zwischen 2 und 12 zu finden, wurden die durchschnittlichen Prognosefehler (in Bezug zum Maximum) sowie die durchschnittliche Stärke der Unterschätzungen für diese k berechnet und in Abb. 3.9 eingetragen. Man sieht deutlich, dass sehr kleine k unter 5 sowohl zu hohen Prognosefehlern als auch stärkerer Unterschätzung führen, während höhere k bessere durchschnittliche Ergebnisse liefern. Das erste lokale Minimum liegt bei $k = 8$, daher soll dies für die kurzfristige

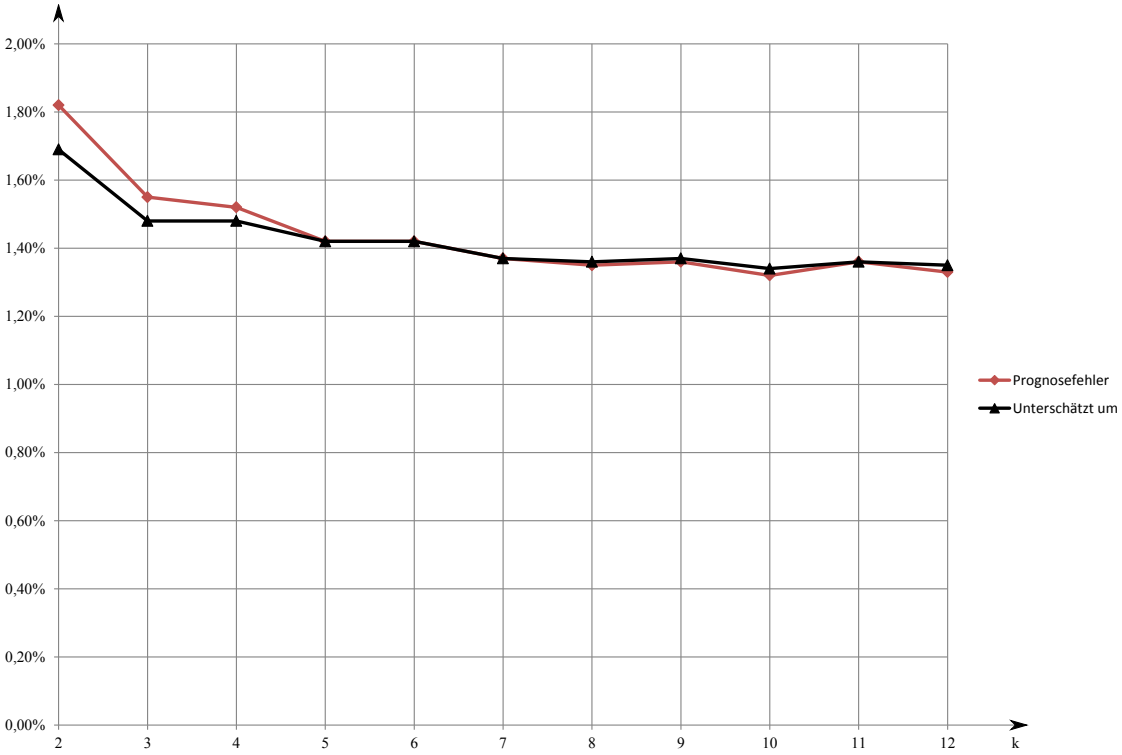

Abbildung 3.9: Prognosefehler und durchschnittliche Unterschätzung bei unterschiedlichem k

Trendanalyse gewählt werden. Die Datenpunkte der letzten 80 Sekunden einzubeziehen erscheint (wie man in Abb. 3.8 sieht) auch angemessen, wenn eine Prognose für die nächsten 20 Sekunden berechnet werden soll.

3.5.2.2 Auswahl der Methode zur pessimistischen Trendanalyse

Die bisher evaluierte kurzfristige Trendanalyse ist noch nicht pessimistisch genug, da sie ca. 50% der Datenpunkte unterschätzt (unabhängig von k). In Kapitel 3.3.2 wurden drei Ansätze für eine pessimistischere Trendanalyse diskutiert, die nun hier gegenübergestellt werden sollen.
In Tab. 3.5 wurde für jeden Ansatz zunächst der mittlere Prognosefehler in Bezug auf das Maximum bestimmt. Hier zeigt sich, dass die Verwendung des Durchschnitts der absoluten Abweichungen AVG(ABS) im Schnitt den geringsten Fehler erzeugt, die Verwendung des Maximums MAX erwartungsgemäß den größten. Allerdings sind alle Prognosen mit einem durchschnittlichen Fehler von unter 2% hinreichend genau, sodass dieses Kriterium nicht ausschlaggebend für die Entscheidung ist.
Neben dem Prognosefehler wurde berechnet, welcher Anteil der Datenpunkte unterschätzt wird. Der Durchschnitt der absoluten Abweichungen unterschätzt mit gut 25%

Tabelle 3.5: Evaluation der kurzfristigen Trendanalyse mit unterschiedlichen Pessimismus-Ansätzen

	AVG(ABS)	upper_quartile(ABS)	MAX
Prognosefehler	1,64%	1,80%	1,83%
Unterschätzt	25,94%	21,76%	19,19%
Unterschätzt um	2,24%	2,46%	2,61%
Unterschätzt um $>$ 10%	0,81%	0,84%	0,76%

deutlich zu viele Datenpunkte. Selbst das sehr pessimistische Maximum unterschätzt noch knapp jeden fünften Wert. Um die Zielsetzung, dass die Gesamtprognose maximal 10% der Datenpunkte unterschätzen darf, erreichen zu können, muss die Wahl auf das Maximum fallen. Bei dem anderen beiden Ansätzen müsste die langfristige Prognose in mindestens 15 bzw. 12% der Fälle deutlich über der kurzfristigen Prognose liegen, um eine Unterschätzung zu vermeiden. Beim Maximum-Ansatz ist das in nur 10% der Fälle nötig.
In der dritten Zeile in Tab. 3.5 wurde berechnet, um wie viel die Unterschätzungen durchschnittlich abweichen (wieder bezogen auf den Maximalwert). Beim oberen Quartil upper_quartile(ABS) und dem Durchschnitt der absoluten Abweichungen werden offenbar sehr viele Werte leicht unterschätzt, daher ist die durchschnittliche Abweichung geringer. Dies bedeutet aber nicht, dass dies bessere Prognosen sind: Da das Maximum am pessimistischsten ist, werden alle Werte, die vom Maximum unterschätzt werden, mindestens in diesem Maße auch von den beiden anderen Ansätzen unterschätzt.
Dies drückt sich auch in der letzten Zeile in Tab. 3.5 aus: Hier wurde berechnet, wie viele Prognosen die Last um mehr als 10% unterschätzen. Hier sieht man deutlich, dass der Maximum-Ansatz mit 0,76% die wenigsten starken Unterschätzungen hervorruft.

Aufgrund der vielen unterschätzten Datenpunkte der anderen beiden Ansätze bleibt nur der pessimistischste Ansatz, das Maximum zu wählen. Da der Prognosefehler dieses Ansatzes noch deutlich im Rahmen liegt, soll er für die pessimistische Prognose verwendet werden.

3.5.2.3 Einhaltung der Zielsetzung

Abschließend soll dargestellt werden, inwieweit die kurzfristige Trendanalyse zum Erreichen der gesteckten Ziele beiträgt.

Pessimismus Mit fast 20% werden noch deutlich zu viele Datenpunkte unterschätzt, hier zeigt sich, dass eine ergänzende langfristige Prognose nötig ist. Der Grad der Unterschätzung von durchschnittlich weniger als 3% stellt dabei noch einen recht guten Wert dar, wenn man bedenkt, dass die Zielsetzung Unterschätzungen bis 10% erlaubt. Derart starke Unterschätzungen treten hier noch in 0,67% der Fälle auf. Dass diese Fälle von der langfristigen Prognose korrigiert werden können, erscheint realistisch und wird in der

Gesamtevaluation untersucht werden.

Geringer Prognosefehler Mit unter 2% erreicht die kurzfristige Trendprognose einen ausgezeichneten mittleren Prognosefehler. Dieser liegt noch weit unter den geforderten 10%, sodass die Einhaltung dieses Ziels in der Gesamtprognose realistisch scheint.

Vertretbarer Overhead Der Overhead der kurzfristigen Trendanalyse ist stark von k abhängig: je mehr Datenpunkte mit einbezogen werden, umso aufwändiger werden die Berechnungen für Mittelwert, Varianz und Kovarianz und damit die Berechnung der kurzfristigen Trendanalyse. Mit $k = 8$ wurde k recht klein gewählt. Selbst für einen schwachen Knoten dürfte es keinerlei Problem darstellen, über 8 Gleitkommazahlen einen Durchschnitt, die Varianz oder die Kovarianz zu berechnen. Dies soll mit einem kurzen Benchmark belegt werden.
Die SQL-Query wurde auf einer Datenbank mit ca. 70.000 Datenpunkten ausgeführt. Die Parameter wurden so gewählt, wie in der Evaluation empfohlen ($k = 8$, $s = 2$, Maximum als Pessimismus-Ansatz). Aus einem einfachen C++-Programm wurde die Query 100.000 mal ausgeführt und die benötigte Zeit gemessen. In jedem Durchlauf wurde die Verbindung zur Datenbank neu geöffnet, die Query ausgeführt, das Ergebnis in eine Variable geschrieben und die Verbindung wieder geschlossen. Auf einem Laptop mit Intel® Core™2 Duo T5870 CPU mit 2.00 GHz und 3 GB Arbeitsspeicher benötigte dieser Benchmark 146,604 Sekunden, also weniger als 1,5 ms pro Berechnung[5]. Die WattDB-Knoten sind aktuell mit einer Intel Atom D510 CPU mit 1,66 GHz sowie 2 GB RAM ausgestattet und damit zwar etwas schwächer, sollten die Berechnung aber trotzdem innerhalb weniger Millisekunden abschließen.
Im Produktivbetrieb ist langfristig selbstverständlich mit einer deutlich größeren Datenbank zu rechnen. Da aber die Query nur $k = 8$ Tupel betrachtet und diese über den Primärschlüssel id auswählt[6], ist nicht davon auszugehen, dass die Ausführzeit der Query durch eine größere Datenbank bedeutend langsamer wird. Da nur alle zehn Sekunden neue Monitoring-Daten eintreffen, muss die Berechnung einer neuen kurzfristigen Prognose auch nur alle zehn Sekunden erfolgen. Eine Berechnungszeit von unter zwei Millisekunden ist daher so unbedeutend, dass dieser Overhead die möglichen Energieeinsparungen rechtfertigt. Selbst wenn man Einzelprognosen für alle ca. 50 vom Monitoring-System erfassten Werte berechnen wollte (was kaum sinnvoll erscheint), so würde dies den Masterknoten nur alle 10 Sekunden für eine Zehntelsekunde beanspruchen.

Berücksichtigung kurzfristiger Trends Die Zielsetzung, dass kurzfristige Trends erkannt und die Last nie häufiger als von der linearen Regression unterschätzt wird, ist dadurch erfüllt, dass eine pessimistische lineare Regression zum Einsatz kommt, die nie niedriger als die lineare Regression ist.

[5] Der Benchmark wurde auf nur einem Prozessorkern ausgeführt.
[6] SQLite unterstützt Indizes.

3.5.3 Gesamtevaluation

Eine Gesamtevaluation ist schwierig, da keine Echtdaten eines Datenbankservers über mehrere Jahre vorliegen. Trotzdem soll anhand der Daten, die auch für die Evaluation der kurzfristigen Trendanalyse genutzt wurden, untersucht werden, wie gut die Gesamtprognose in der ersten Woche ist. Da die langfristige Mustererkennung, wie in Kapitel 3.5.1 gezeigt, mit der Zeit besser wird, bedeutet dies, dass langfristig mit besseren Ergebnissen gerechnet werden kann.
Für die Gesamtevaluation wurde sowohl die langfristige Mustererkennung als auch die kurzfristige Trendanalyse für die Daten des SQL Servers der SPH AG berechnet. Auf Basis der langfristigen Mustererkennung wurde für jede Minute eine Prognose berechnet, wobei (im Gegensatz zur Evaluation in Kapitel 3.5.1) das Muster „hour + minute“ mit in die Berechnung eingeschlossen wurde. Da die langfristige Prognose am ersten Tag keine Ergebnisse liefern kann, wird die Gesamtevaluation ab dem zweiten Tag über die verbleibenden sieben Tage durchgeführt. Die kurzfristige Prognose wurde mit dem pessimistischen Verfahren auf Basis des Maximums, $k = 8$ und $s = 2$ für alle zehn Sekunden berechnet. Die beiden Prognosen wurden dann wie in Kapitel 3.4.3 beschrieben zu einer Gesamtprognose verrechnet.

Tabelle 3.6: Evaluation der Gesamtprognose

	Nur kurzfristig	Gesamtprognose
Prognosefehler	1,83%	3,42%
Unterschätzt	19,19%	11,75%
Unterschätzt um	2,61%	0,000028%
Unterschätzt um $>$ 10%	0,76%	0,37%

Die Ergebnisse der so durchgeführten Gesamtevaluation sind in Tab. 3.6 den Ergebnissen der kurzfristigen Evaluation gegenübergestellt. Wie erwartet, erhöht sich der Prognosefehler durch die Hinzunahme der langfristigen Muster, bleibt aber mit gut 3% deutlich unter den geforderten 10%.
Die Gesamtprognose unterschätzt mit 11,75% deutlich weniger Datenpunkte als die kurzfristige Evaluation. Die Zielsetzung, nicht mehr als 10% der Datenpunkte zu unterschätzen, wird zwar knapp verfehlt, aber mit zunehmendem Pessimismus der langfristigen Mustererkennung ist davon auszugehen, dass dieser Wert schon nach wenigen Monaten unter 10% fällt.
Beachtenswert ist, dass die Unterschätzungen der Gesamtprognose im Schnitt weniger als ein Promille der Maximallast ausmachen und daher meist bedeutungslos werden.
Wichtiger ist, dass trotzdem noch 0,37% der Prognosen die Last um mehr als 10% unterschätzen. Dies sollte nach Zielsetzung zwar nie der Fall sein, allerdings ist auch hier davon auszugehen, dass mit der Zeit die langfristige Mustererkennung diesen Wert noch deutlich senken wird. Immerhin werden dank langfristiger Mustererkennung schon in der ersten Woche nur noch halb so viele Werte so stark unterschätzt.

Kapitel 4

Ausblick

Nachdem die Lastprognose für WattDB vorgestellt wurde, sollen im Folgenden verbleibende Optimierungsmöglichkeiten diskutiert, sowie ein Ausblick auf das weitere Vorgehen zur Nutzung der Prognose in WattDB gegeben werden.

4.1 Evaluation an langfristigen Echtdaten

Aufgrund fehlender langfristiger Daten konnte die langfristige Mustererkennung leider nur an Daten evaluiert werden, die nicht an einem Datenbanksystem erhoben wurden. Wünschenswert wäre es, eine Gesamtevaluation an Echtdaten eines Datenbanksystems durchzuführen, die über mindestens drei Jahre gesammelt wurden. Idealerweise sollten die Daten mit dem Monitoring-System von WattDB erhoben worden sein.
Anhand solcher Daten ließe sich ähnlich wie in Kapitel 3.5.2 die Prognose mit unterschiedlichen Parametern durchrechnen, um das Verfahren weiter zu optimieren. Im Folgenden werden einige solcher Optimierungsmöglichkeiten genannt, die von einer solchen Evaluation profitieren würden.

4.2 Weitere Optimierungen

Wie in der Evaluation gezeigt liefert das entwickelte Prognoseverfahren bereits gute Ergebnisse. Trotzdem sind weitere Optimierungen denkbar: Zum einen kann versucht werden, bessere Prognosen mit einem geringeren Prognosefehler oder geringeren Unterschätzungen zu erreichen. Zum anderen kann man versuchen, den Overhead der Prognose weiter zu minimieren.

4.2.1 Optimierungen für bessere Prognosen

Das Prognoseverfahren wird durch eine ganze Reihe von unterschiedlichen Parametern bestimmt, welche die Güte der Prognose beeinflussen können. Bei der langfristigen Prognose beginnt dies mit der Definition der Mustertypen: Hier stellt sich die Frage, ob es weitere Muster gibt oder ob manche Muster die Prognose eher verschlechtern und daher

entfernt werden sollten. Prinzipiell könnte man z.B. jedem Muster neben der Stunde noch die Minute hinzufügen. Da die Definition der Muster einen offensichtlichen Ansatzpunkt für Optimierungen und Anpassungen darstellt, wurden die SQL-Queries der Muster nicht fest in das C++-Programm der Prognose integriert, sondern in eine Datei ausgelagert, sodass eine Anpassung ohne Neukompilierung möglich ist.
Neben den Mustern selbst könnte man untersuchen, ob die Gewichtung der Muster noch verbessert werden kann. So wäre es denkbar, dass der hier genutzte maximale Variationskoeffizient von $0,18$ eventuell zu niedrig gewählt ist: Angenommen, die Last eines Musters beträgt durchschnittlich 5% und schwankt sehr stark zwischen 0% und 10%, die Standardabweichung betrage also 5% und der Variationskoeffizient damit 1. Ein solches Muster wird nach aktueller Gewichtung gar nicht berücksichtigt. Dies kann dazu führen, dass die langfristige Prognose in solchen Fällen sämtliche Muster verwirft und keine Prognose abgeben kann (bzw. 0 prognostiziert). Hier wäre zu untersuchen, ob es tatsächlich gut ist, in solchen Fällen auf die langfristige Prognose zu verzichten und nur die kurzfristige einzusetzen, oder ob solche Muster trotzdem berücksichtigt werden sollen.
Eine weitere Möglichkeit, Prognoseergebnisse zu verbessern, stellt die höhere Gewichtung jüngerer Daten dar: Da sich Muster in Lastverläufen ändern können, nützt es nichts, ein jahrelang stark ausgeprägtes Muster auch noch über Jahre stark zu gewichten, obwohl es gar nicht mehr auftritt. Bei der Lastprognose sind neuere Daten i.d.R. bessere Daten und könnten daher höher gewichtet werden. Die einfachste Möglichkeit, neuere Daten stärker zu gewichten, wäre eine Aggregation alter Daten: Man könnte beispielsweise die Daten des letzten Jahres auf 20 Sekunden aggregieren (also von je zwei Datenpunkten den Mittelwert bilden), Daten des vorletzten Jahres auf 30 Sekunden und so weiter. Ohne Veränderung der Prognose würde dies die älteren Daten schwächer gewichten als jüngere, da das obere Quartil dann über weniger ältere Daten gebildet würde.

4.2.2 Optimierungen für weniger Overhead

Obwohl die Prognose in ihrer derzeitigen Form einen vertretbaren Overhead erzeugt, bedeutet eine Minimierung von Overhead prinzipiell auch die Möglichkeit, das Datenbanksystem zu entlasten und Energie zu sparen.
Wie bereits erwähnt, wäre es wünschenswert, alte Daten zu aggregieren. Dies würde nicht nur wie erläutert die Prognose verbessern, indem neuere Werte stärker gewichtet werden, sondern auch den Overhead in zweierlei Hinsicht verringern: Zum einen würde die Aggregation den Speicherbedarf der Datenbank verringern und damit vermeiden, dass dieser stetig wächst. Zum anderen läuft die SQL-Query der langfristigen Mustererkennung schneller, wenn weniger Datenpunkte zu verarbeiten sind. Da mit dieser Optimierung sowohl der Prognosefehler als auch der Overhead optimiert werden kann, stellt dies die interessanteste Weiterentwicklung dar.
Eine weitere Möglichkeit, die langfristige Mustererkennung zu beschleunigen, wäre das Zwischenspeichern einzelner Musterergebnisse: So könnte man zwischen Mustern unterscheiden, welche die Minute verwenden und denen, die die Minute nicht enthalten. Muster, die die Minute nicht enthalten, müssen nur jede Stunde berechnet werden und können so lange zwischengespeichert werden. Muster, welche die Minute mit einfließen

lassen, könnten jede Minute berechnet werden und zusammen mit den zwischengespeicherten Mustern verrechnet werden. Dies würde es erlauben, Muster auf Minuten-Basis zu verwenden, ohne einen deutlich höheren Overhead zu erzeugen.

4.3 Nutzung der Prognose in WattDB

Für WattDB wurde zunächst das Monitoring-Framework und nun das Prognoseverfahren zur Lastprognose entwickelt. Doch so lange die prognostizierten Werte nicht zur Steuerung des DBMS genutzt werden, sind sie wertlos. Es ist daher als nächster Schritt nötig, dass WattDB die prognostizierten Werte interpretiert und basierend auf der Prognose Knoten zu- und abschaltet.

4.3.1 Starten und Stoppen von Knoten auf Basis der Lastprognose

Die Energieproportionalität des Systems verbessert sich erst, sobald das Starten und Stoppen von Knoten auf Basis der Lastprognose geschieht. In Gl. (4.1) ist ein sehr einfaches Verfahren zur Berechnung der benötigten Anzahl Knoten auf Basis der Prognose skizziert. Die prognostizierte Last wird durch die Maximalkapazität des Clusters geteilt, um den Anteil benötigter Knoten zu bestimmen. Dieser Anteil wird mit der Gesamtanzahl Knoten multipliziert und auf ganze Knoten aufgerundet.

$$Knoten_{benötigt} = \left\lceil \frac{Prognose}{Kapazität} * Knoten_{gesamt} \right\rceil \tag{4.1}$$

Nun geht dieser Ansatz aber davon aus, dass eine Prognose für nur genau eine Kennzahl berechnet wird. In Wirklichkeit werden aber zahlreiche Kennzahlen für Prozessor, Arbeitsspeicher, Netzwerk, Festplatten etc. ausgewertet und prognostiziert. Würde man alle diese Kennzahlen einzeln in die Gleichung einsetzen, würde man für jede Kennzahl eine Anzahl benötigter Knoten herausbekommen. Ob es nun sinnvoll ist, hier das Maximum zu verwenden, oder ob eine deutlich komplexere Entscheidungsfindung nötig ist, muss erst noch untersucht werden.

4.3.2 ECA-Regelwerk für WattDB

Um eine möglichst optimale Steuerung von WattDB, insbesondere der Anpassung der Clustergröße, zu erreichen, ist geplant, ein *Event-Condition-Action*-Regelwerk (ECA) zu entwerfen. Dies soll bei Eintreten bestimmter Ereignisse (*events*) unter Berücksichtigung der gegebenen Bedingungen (*conditions*) Entscheidungen treffen, um Aktionen (*actions*) auszuführen. Ein solches Ereignis könnte der Ablauf eines Timers (z.B. alle zehn Sekunden), das Eintreffen einer Nachricht (z.B. „Knoten 3 überlastet“) oder Datenbank-Events wie Commit oder Rollback sein. Die Bedingungen wären zum einen der aktuelle Zustand des Clusters, wie er durch das Monitoring-System dem Master stets verfügbar gemacht wird, zum anderen aber auch die Prognose des Monitoring-Systems. Die Aktionen wären in erster Linie das Starten oder Stoppen eines Knotens.

Tabelle 4.1: Beispiel ECA-Regeln für die Steuerung von WattDB

Event	Condition	Action
Timer	$Knoten_{aktiv} < Knoten_{benötigt}$ AND $Knoten_{aktiv} < Knoten_{gesamt}$	Starte Knoten (Anzahl: $Knoten_{benötigt} - Knoten_{aktiv}$)
Timer	seit 30 Sekunden $Knoten_{aktiv} > Knoten_{benötigt}$	Stoppe Knoten (Anzahl: $Knoten_{benötigt} - Knoten_{aktiv}$)

In Tab. 4.1 sind einfache Beispiele solcher Regeln definiert. Hier sieht man schon eine kleine Verfeinerung: Das Stoppen von Knoten soll nicht sofort geschehen, sondern erst, wenn seit 30 Sekunden zu viele Knoten aktiv sind. Dies würde verhindern, dass ein Knoten ständig zu- und abgeschaltet wird, weil die Prognose leicht schwankt. Da das Starten eines Knotens Energie benötigt, ohne dass der Knoten währenddessen Anfragen bearbeiten kann, würde ein ständiges An- und Abschalten eines Knotens keine Energie einsparen. Ein ECA-Regelwerk könnte aber deutlich komplexere Regeln enthalten, welche auch auf die unterschiedlichen Kennzahlen für CPU, RAM, Festplatte und Netzwerk eingehen könnten. Darüber hinaus könnte beispielsweise zwischen Knoten mit Festplatte sowie Knoten ohne eigene Festplatte unterschieden werden.

4.4 Nutzung des Prognoseverfahrens in anderen Systemen

Das im Rahmen dieser Arbeit entwickelte Prognoseverfahren wurde zwar in Hinblick auf die Prognose der Last eines DBMS entwickelt, ist aber nicht so speziell auf DBMS ausgerichtet, dass es sich nicht für andere verteilte Systeme einsetzen ließe. Langfristige Muster sind auch in Lastverläufen anderer Systeme verbreitet und daher könnte die Prognose auch (eventuell mit kleineren Anpassungen) in anderen verteilten Systemen zum Einsatz kommen. Geeignet wäre sie für jedes verteilte System, das die Clustergröße dynamisch anpassen kann und langfristige Muster in den Lastverläufen zeigt, welche den verwendeten Mustertypen entsprechen. Denkbar wäre beispielsweise der Einsatz in einem MapReduce-Cluster [1], wobei das verteilte Dateisystem hier eventuell für den gezielten Wegfall von Knoten angepasst werden müsste. Eine andere mögliche Anwendung wäre der Einsatz in einem Webserver-Cluster. Gerade in Hinblick auf den aktuellen Trend, Dienste „in der Cloud“ verfügbar zu machen und Software, Plattformen oder IT-Komponenten als Dienst anzubieten (*Software as a Service*), werden Cluster zukünftig immer mehr zum Einsatz kommen. Damit wird auch der Bedarf wachsen, die Last solcher Systeme prognostizieren zu können.

Kapitel 5

Fazit

In der Evaluation konnte gezeigt werden, dass sich die im Rahmen dieser Bachelorarbeit entwickelte Lastprognose wie erhofft verhält und den gesetzten Zielsetzungen gerecht wird. Durch die Aufteilung in eine kurzfristige Trendanalyse und eine langfristige Mustererkennung ist es gelungen, sowohl überraschend auftretende kurzfristige Lastspitzen und Lasteinbrüche, als auch regelmäßig wiederkehrende Muster zu berücksichtigen. Begonnen bei einfachen täglichen Mustern über wöchentliche und monatliche Muster kann die Prognose selbst komplizierte Muster wie bewegliche Feiertage zuverlässig erkennen und berücksichtigen.
Beide Teilprognosen wurden dabei konsequent pessimistisch konzipiert und tendieren deshalb eher zum Überschätzen als zum Unterschätzen. Dies führt dazu, dass die prognostizierte Last die tatsächliche Last nur noch selten unterschätzt. Die trotzdem auftretenden Unterschätzungen sind äußerst gering und werden immer weniger, je besser die Prognose die langfristigen Muster erkennen kann.
Trotz Pessimismus ist es gelungen, einen geringen Prognosefehler zu erzielen. Der durchschnittliche Prognosefehler liegt deutlich unter der Kapazität eines Knotens, was dazu führt, dass die Prognose die benötigte Anzahl Knoten zuverlässig prognostiziert.
Obwohl die Prognose neben kurzfristigen Trends auch komplexe langfristige Muster erkennt, verursacht sie keinen besonders großen Overhead. Dies ist besonders wichtig, da der Overhead der Prognose durch die eingesparte Energie gerechtfertigt sein muss.

Dank der entwickelten Lastprognose kann ein verteiltes DBMS wie WattDB seine Clustergröße rechtzeitig dynamisch anpassen, ohne dass mit Überlastsituationen zu rechnen ist. So wird es möglich, Energieeffizienz und Energieproportionalität zu erreichen, ohne Leistungseinbußen und verzögerte Antwortzeiten hinnehmen zu müssen.

Kapitel 6

Anhang

6.1 Datenbankschema

Hinweis: Das Schema der Tabelle monitoringData in Tab. 6.1 stellt eine vereinfachte Sicht des Datenbankschemas des Monitoring-Frameworks von WattDB dar. In Wirklichkeit gibt es die Spalten cpuLoad, diskLoad, memoryLoad und networkLoad nicht. Stattdessen bildet das Datenbankschema des Monitoring-Frameworks detailliert diverse Kennzahlen zu allen Knoten des Clusters ab. Beispielweise werden für die CPU user-, iowait- und idle-Zeiten erfasst oder von den Festplatten Lese- und Schreibvorgänge gezählt. Die hier entwickelte Lastprognose kann auf jede dieser Kennzahlen angewendet werden. Um die Übersichtlichkeit, insbesondere der in dieser Arbeit aufgeführten Beispiele, zu gewährleisten, wurde darauf verzichtet, das vollständige Schema darzulegen. Stattdessen wird konsequent das hier abgedruckte vereinfachte Schema verwendet, welches alle für die Lastprognose relevanten Teile enthält. Das vollständige Schema des Monitoring-Systems ist in [11] nachzulesen.

Tabelle 6.1: Schema monitoredData

Spaltenname	Datentyp	Eigenschaften
id	integer	primary key
second	integer	not null
minute	integer	not null
hour	integer	not null
day	integer	not null
month	integer	not null
year	integer	not null
span	integer	not null
cpuLoad	integer	not null
diskLoad	integer	not null
memoryLoad	integer	not null
networkLoad	integer	not null

Tabelle 6.2: Schema dayInfo

Spaltenname	Datentyp	Eigenschaften
day	integer	primary key
month	integer	primary key
year	integer	primary key
weekday	integer	not null
dayOfYear	integer	not null
monthDaysLeft	integer	not null
weekdayThisMonth	integer	not null
weekdaysLeftThisMonth	integer	not null

Tabelle 6.3: Schema specialDay

Spaltenname	Datentyp	Eigenschaften
year	integer	primary key
name	text	primary key
dayOfYear	integer	not null

6.2 SQL-Query langfristige Mustererkennung

Diese Query gibt für alle Muster jeweils den Namen des Musters, den Prognosewert sowie den Variationskoeffizienten aus. Prognosezeitpunkt ist „now“, also der jetzige Zeitpunkt. Man kann in der Query leicht 'now' durch einen absoluten Zeitpunkt, wie z.B. '2012-03-31 13:27:00' oder einen anderen relativen Zeitpunkt wie 'now','+ 20 seconds', ersetzen. Außerdem kann außer der Spalte „cpuLoad“ jede Spalte des Datenbankschemas gewählt werden, um eine Prognose für diese Spalte zu berechnen.

```
SELECT 'hour + minute', upper_quartile(m.cpuLoad),
       STDEV(m.cpuLoad)/AVG(m.cpuLoad)
FROM   monitoredData m, dayInfo d
WHERE  d.day=m.day AND d.month=m.month AND d.year=m.year
       AND m.hour=strftime('%H',datetime('now'))
       AND m.minute=strftime('%M',datetime('now'))

UNION

SELECT 'weekday + hour', upper_quartile(m.cpuLoad),
       STDEV(m.cpuLoad)/AVG(m.cpuLoad)
FROM   monitoredData m, dayInfo d
WHERE  d.day=m.day AND d.month=m.month AND d.year=m.year
       AND d.weekday=strftime('%w',datetime('now'))
       AND m.hour=strftime('%H',datetime('now'))

UNION

SELECT 'day + hour', upper_quartile(m.cpuLoad),
       STDEV(m.cpuLoad)/AVG(m.cpuLoad)
FROM   monitoredData m
WHERE  m.day=strftime('%d',datetime('now'))
       AND m.hour=strftime('%H',datetime('now'))

UNION

SELECT 'monthDaysLeft + hour', upper_quartile(m.cpuLoad),
       STDEV(m.cpuLoad)/AVG(m.cpuLoad)
FROM   monitoredData m, dayInfo d
WHERE  d.day=m.day AND d.month=m.month AND d.year=m.year
       AND d.monthDaysLeft=strftime('%d',datetime('now'),
                'start of month','+1 month','-1 day')
               -strftime('%d',datetime('now'))
       AND m.hour=strftime('%H',datetime('now'))

UNION

SELECT 'xth weekday + hour', upper_quartile(m.cpuLoad),
       STDEV(m.cpuLoad)/AVG(m.cpuLoad)
FROM   monitoredData m, dayInfo d
```

```
WHERE   d.day=m.day AND d.month=m.month AND d.year=m.year
        AND d.weekday=strftime('%w',datetime('now'))
        AND d.weekdayThisMonth=CEIL(strftime('%d',datetime('now'))/7)
        AND m.hour=strftime('%H',datetime('now'))

UNION

SELECT 'x weekdays left + hour', upper_quartile(m.cpuLoad),
        STDEV(m.cpuLoad)/AVG(m.cpuLoad)
FROM    monitoredData m, dayInfo d
WHERE   d.day=m.day AND d.month=m.month AND d.year=m.year
        AND d.weekday=strftime('%w',datetime('now'))
        AND d.weekdaysLeftThisMonth=(strftime('%d',datetime('now'),
                'start of month','+1 month','-1 day')
                -strftime('%d',datetime('now')))/7
        AND m.hour=strftime('%H',datetime('now'))

UNION

SELECT 'day + month + hour', upper_quartile(m.cpuLoad),
        STDEV(m.cpuLoad)/AVG(m.cpuLoad)
FROM    monitoredData m
WHERE   m.day=strftime('%d',datetime('now'))
        AND m.month=strftime('%m',datetime('now'))
        AND m.hour=strftime('%H',datetime('now'))

UNION

SELECT 'month + hour', upper_quartile(m.cpuLoad),
        STDEV(m.cpuLoad)/AVG(m.cpuLoad)
FROM    monitoredData m
WHERE   m.month=strftime('%m',datetime('now'))
        AND m.hour=strftime('%H',datetime('now'))

UNION

SELECT 'specialDay_' || s.name || ' + hour',
        upper_quartile(m.cpuLoad), STDEV(m.cpuLoad)/AVG(m.cpuLoad)
FROM    monitoredData m, dayInfo d, specialDay s
WHERE   d.day=m.day AND d.month=m.month AND d.year=m.year
        AND s.year=m.year
        AND (d.dayOfYear-s.dayOfYear)=
                strftime('%j',datetime('now'))-(
                        SELECT dayOfYear FROM specialDay s2
                        WHERE s2.year=strftime('%Y',datetime('now'))
                        AND s2.name=s.name)
        AND m.hour=strftime('%H',datetime('now'))
GROUP BY s.name;
```

6.3 Verrechnung aller Muster zu einem Prognosewert

Die folgende C++-Funktion ruft die Einzelprognosewerte aller Muster von der Datenbank ab und verrechnet sie zu einer Gesamtprognose.

```
1  double predict(string predict_time) {
2          // die SQL-Query aus 6.1 wurde aus einer Datei nach patterns
3          // geladen
4          string query = patterns;
5          // der Prognosezeitpunkt wird in die Query eingesetzt
6          replace(query,"'now'",predict_time);
7          // Die Query wird ausgefuehrt
8          vector<vector<string>> result = db->query(query);
9          // Deklariere: Maximaler Variationskoeffizient
10         double maximumVarCoeff = 0.18;
11         // Initialisierung Summe aller Gewichte (Divisor gew. Mittel)
12         double totalWeight = 0;
13         // Initialisierung Divident des gew. Mittel
14         double predict = 0;
15         // durchlaufe alle Muster
16         for (vector<vector<string>>::iterator it = result.begin();
17                         it < result.end(); ++it) {
18                 vector<string> row = *it;
19                 // SQLite gibt alle Spalten als Strings zurueck
20                 string patternPredValStr = row.at(1);
21                 string patternVarCoeffStr = row.at(2);
22                 // konvertiere Prognosewert nach double
23                 std::istringstream stm;
24                 stm.str(patternPredValStr);
25                 double patternPredVal;
26                 stm >> patternPredVal;
27                 // konvertiere Variationskoeffizient nach double
28                 std::istringstream stm2;
29                 stm2.str(patternVarCoeffStr);
30                 double patternVarCoeff;
31                 stm2 >> patternVarCoeff;
32                 // Sortiere schwache Muster mit zu hohem
33                 // Variationskoeffizient sowie leere Muster aus
34                 if (patternVarCoeff < maximumVarCoeff
35                                 && patternVarCoeffStr!="") {
36                         // Berechne Gewicht
37                         double weight =
38                                 (maximumVarCoeff-patternVarCoeff) *
39                                 (maximumVarCoeff-patternVarCoeff);
40                         // Summiere das Gewicht auf
41                         // (Divisor gew. Mittel)
42                         totalWeight += weight;
43                         // Summiere Gewicht * Prognosewert auf
44                         // (Divident gew. Mittel)
```

```
45                          predict += weight * patternPredVal;
46                  }
47          }
48          if(totalWeight > 0) {
49                  // Berechne Gesamt-Prognose (Divident / Divisor)
50                  predict /= totalWeight;
51          }
52          // Falls alle Muster rausfallen ist die Prognose 0
53          // Rueckgabe Gesamtprognose
54          return predict;
55  }
```

Listing 6.1: Funktion zur Berechnung der langfristigen Prognose

Literaturverzeichnis

[1] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: simplified data processing on large clusters. In: *Commun. ACM* 51 (2008), Januar, Nr. 1, 107–113. http://dx.doi.org/10.1145/1327452.1327492. – DOI 10.1145/1327452.1327492. – ISSN 0001–0782

[2] Ecos Consulting: Energy-Efficient Computers Run With 80 PLUS™. http://www.plugloadsolutions.com/docs/broch/80PLUS_brochurepages.pdf

[3] Fichter, Dr. K.: Energieverbrauch und Energiekosten von Servern und Rechenzentren in Deutschland. (2008). http://www.bitkom.org/files/documents/Energieeinsparpotenziale_von_Rechenzentren_in_Deutschland.pdf

[4] Fichter, Prof. Dr. K. ; Clausen, Dr. J. ; Hintemann, Dr. R.: Roadmap "Ressourceneffiziente Arbeitsplatz-Computerlösungen 2020". (2011). http://www.bitkom.org/files/documents/Roadmap_ressourceneffizientearbeitsplatzcomputerloesungen_web(1).pdf

[5] Healy, Liam ; C, Mikey ; relicoder: *SQLite Extension-Functions*. https://www.sqlite.org/contrib/download/extension-functions.c?get=25

[6] Hoffmann, Norbert: *Kleines Handbuch neuronaler Netze*. Braunschweig/Wiesbaden : Vieweg+Teubner, 1993. – ISBN 9783528052392

[7] Härder, Theo ; Hudlet, Volker ; Ou, Yi ; Schall, Daniel: Energy Efficiency Is Not Enough, Energy Proportionality Is Needed! Version: 2011. http://dx.doi.org/10.1007/978-3-642-20244-5_22. In: Xu, Jianliang (Hrsg.) ; Yu, Ge (Hrsg.) ; Zhou, Shuigeng (Hrsg.) ; Unland, Rainer (Hrsg.): *Database Systems for Adanced Applications* Bd. 6637. Springer Berlin / Heidelberg, 2011. – ISBN 9783642202438, 226-239

[8] Hüttner, M.: *Prognoseverfahren und ihre Anwendung*. "de" Gruyter, 1986. – ISBN 9783110108262

[9] Intel Corporation: *Intel® Xeon® 5600er-Prozessoren*. http://www.intel.com/content/www/de/de/processors/xeon/xeon-5600-series-processors.html. Version: 01. März 2012

[10] Lippe, P.: *Deskriptive Statistik.* Oldenbourg, 2006. – ISBN 9783486578638

[11] Martins Dusso, Pedro: *A Monitoring System for WattDB: An Energy-Proportional Databsae Cluster.* Januar 2012

[12] National Grid: *Metered half-hourly electricity demands.* http://www.nationalgrid.com/uk/Electricity/Data/Demand+Data/. Version: 2012

[13] Schall, Daniel ; Höfner, Volker ; Kern, Manuel: Towards an Enhanced Benchmark Advocating Energy-Efficient Systems. In: *Proc. TPCTC 2011*, 2011

[14] Schall, Daniel ; Hudlet, Volker: WattDB: an energy-proportional cluster of wimpy nodes. In: Sellis, Timos K. (Hrsg.) ; Miller, Renée J. (Hrsg.) ; Kementsietsidis, Anastasios (Hrsg.) ; Velegrakis, Yannis (Hrsg.): *SIGMOD Conference*, ACM, 2011. – ISBN 9781450306614, 1229-1232

[15] Schall, Daniel ; Hudlet, Volker ; Härder, Theo: Enhancing energy efficiency of database applications using SSDs. In: *Proceedings of the Third C* Conference on Computer Science and Software Engineering.* New York, NY, USA : ACM, 2010 (C3S2E '10). – ISBN 9781605589015, 1–9

[16] Schulze, P.M.: *Beschreibende Statistik.* Oldenbourg, 2007. – ISBN 9783486582208

[17] Spahn, Günther: *Energieanalyse „Energy Outlook 2030": Globaler Energiebedarf steigt bis 2030 um 39 Prozent.* http://www.epochtimes.de/865922_globaler-energiebedarf-steigt-bis-2030-um-39-prozent.html. Version: 01. März 2012

[18] Spiegel Online: *Energiekosten: Wo Gas und Strom teurer werden.* http://www.spiegel.de/wirtschaft/service/0,1518,806074,00.html. Version: 01. März 2012

[19] SQLite-Team: *SQLite.* http://www.sqlite.org

[20] Tsirogiannis, Dimitris ; Harizopoulos, Stavros ; Shah, Mehul A.: Analyzing the energy efficiency of a database server. In: *Proceedings of the 2010 international conference on Management of data.* New York, NY, USA : ACM, 2010 (SIGMOD '10). – ISBN 9781450300322, 231–242

[21] Vockerodt, V.: *Einsatz neuronaler Netze zur Mustererkennung.* GRIN Verlag GmbH, 2007. – ISBN 9783638691147

Erklärung über die selbständige Anfertigung dieser Arbeit

Ich erkläre hiermit, diese Arbeit selbstständig verfasst und nur die im Literaturverzeichnis aufgeführten Quellen und Hilfsmittel verwendet zu haben.

Ort, Datum Unterschrift