Lastprognose für energieeffiziente verteilte DBMS

Christopher Kramer, Volker Höfner, Theo Härder

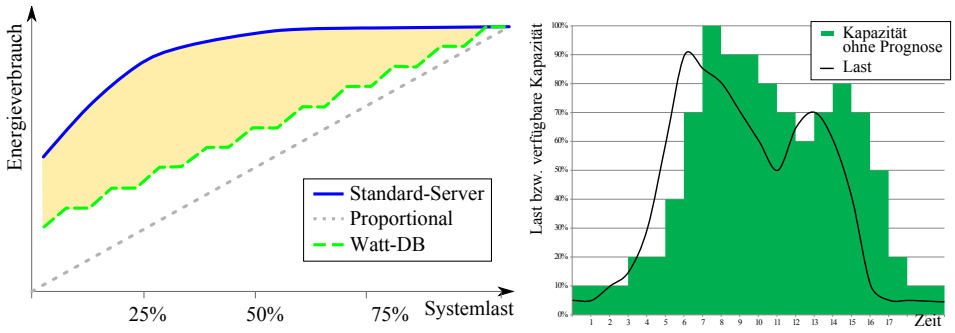
AG DBIS, TU Kaiserslautern
c_kramer, hoefner, haerder@cs.uni-kl.de

Abstract: Energieeffizienz gewinnt in der IT und auch im Bereich von 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 dazu ein Verfahren, 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.

1 Einleitung

Neben effizienter, energiesparender Hardware können auch Softwarelösungen zur Realisierung möglichst energieeffizienter IT-Systeme beitragen, indem sie Hardware je nach Last steuern und gerade nicht benötigte Hardware abschalten. Das gilt insbesondere auch für DBMS, die als verteilte Systeme für immer größere Datenvolumina steigende Leistungsanforderungen und zugleich Energieeffizienz gewährleisten sollen. In der Regel zielte der Entwurf solcher Systeme bisher nur auf Leistungssteigerung durch Parallelisierung ab. 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 TU Kaiserslautern entwickelte verteilte DBMS WattDB [SH11] 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 bei Überlast startet, entstehen deutliche 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 abzuschätzen, sodass Knoten genau so gestartet werden, dass sie bei Bedarf einsatzbereit sind. Dieses Ziel soll dadurch angenähert werden, dass auf Basis vergangener Lastverläufe Prognosen über die zukünftige Last eines solchen DBMS gemacht werden. Dabei sollen sowohl Muster in den Lastverläufen der Vergangenheit, als auch kurzfristige Trends in die Prognose einfließen.



(a) Annäherung der Energieproportionalität **(b)** Last- und Kapazitätsverlauf ohne Prognose

Abb. 1: Verbesserte Energieeffizienz durch ein Cluster leichtgewichtiger Rechnerknoten

Im Folgenden wird zunächst erörtert, warum Lastprognosen Energieeffizienz und Energieproportionalität für DBMS verbessern können, bevor ihre Umsetzung in WattDB skizziert wird. In Kapitel 3 werden Methoden für lang- und kurzfristige Lastprognosen diskutiert, ein Lösungsansatz mittels Mustererkennung und linearer Regression vorgestellt sowie die Implementierung für WattDB beschrieben. Kapitel 4 führt eine Evaluation der vorgestellten Lastprognose durch. In Kapitel 5 folgt ein Ausblick auf weitere Optimierungen und Nutzungen der Lastprognose, bevor ein Fazit gezogen wird.

2 Energieproportionalität von DBMS

Wie die meisten Server-Systeme sind auch DBMS [SHK11] 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 der Arbeitsspeicher, einen hohen lastunabhängigen Grundverbrauch haben, verhält sich der Energiebedarf dieser Server nicht proportional zur anfallenden Last [THS10]. 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. 1a 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. Perfekte Energieproportionalität lässt sich nicht erreichen, da immer nur ganze Knoten zugeschaltet werden können und mindestens ein Knoten immer betrieben werden muss. Jedoch ist an den jeweiligen Energieverläufen zu erkennen, dass WattDB deutliche Energieeinsparungen erreichen kann. Außerdem kann sich sich der Energieverbrauch von WattDB dem perfekten Verlauf umso mehr annähern, je mehr Knoten im Cluster vorhanden sind und benötigt werden.

2.1 Lastprognose für bessere Energieproportionalität

Das Zuschalten von Knoten benötigt einige Sekunden, bis sie in WattDB einsatzbereit sind. Würden sie erst gestartet, sobald eine erhöhte Last anfällt, würden zeitweise erhebliche Verzögerungen durch Überlast auftreten. In Abb. 1b 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 Lasterhöhungen abzufangen. Dadurch würde allerdings die Energieeinsparung des Systems erheblich geschmälert. Außerdem ist die Abschätzung der Anzahl von Reserveknoten schwierig, die eine plötzlich eintretende Lastspitze erfordert. Für eine bessere Annäherung der Energieproportionalität ist es daher notwendig, zusätzliche Knoten genau dann zu starten, wenn in wenigen Sekunden eine erhöhte Last zu erwarten ist, sodass neue Knoten schon einsatzbereit sind, wenn die Last auftritt. Dies ist nur möglich, wenn eine Prognose über die zukünftige Last getroffen werden kann.

2.2 WattDB

WattDB ist ein verteiltes DBMS, welches als Cluster leichtgewichtiger Knoten geringen Energieverbrauchs mit dem Ziel entworfen wurde, Energieproportionalität anzunähern [HHOS11]. Das aktuelle WattDB-Demo-Cluster sieht wie folgt aus [SH11]:
Zehn Knoten sind mit je 2 GB RAM und einer Intel® Atom D510 CPU ohne Frequenz-Skalierung ausgestattet. Die Netzteile der Knoten sind äußerst effizient und 80 Plus® gold zertifiziert. Im Leerlauf benötigen diese Knoten ca. 23 Watt, unter Volllast steigt der Verbrauch auf etwa 26 Watt. Aktuell sind zwei der zehn Knoten mit je vier SATA-Festplatten mit jeweils 250 GB ausgestattet. Datenknoten mit Festplatten (mit möglichst geringem Energiebedarf) verbrauchen bis zu 41 Watt. 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.

Energieproportionalität in WattDB Wie in Abb. 1a skizziert, wird sie durch das Zu- und Abschalten von Knoten erreicht. Obwohl verteilte DBMS schon lange verbreitet sind, unterstützen kommerzielle DBMS bisher nicht das Abschalten von Knoten, 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 oder komplett heruntergefahren werden. Knoten im Standby benötigen noch ca. 3 Watt und lassen sich per *wake-on-LAN* in weniger als fünf Sekunden starten [SH11]. 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, indem z. B. nicht benötigte Kernel-Module deaktiviert wurden, dauert ein Knotenstart noch ungefähr zwanzig Sekunden.

Monitoring in WattDB Bei solchen Startzeiten kann bei steigender Last eine Überlast von mehreren Sekunden auftreten, weil Knoten nicht rechtzeitig bereit sind. Um solche Situationen zu reduzieren, wollen wir Lastprognosen aus Daten früherer Lastverläufe gewinnen. Deshalb wurde für WattDB ein Monitoring-System entworfen [MD12], das auf allen Knoten Clients einsetzt, die alle zehn Sekunden verschiedene Lastwerte sammeln und an einen Server auf dem Masterknoten senden. Dieser Server empfängt die protokollierten Daten und speichert sie in einer SQLite-Datenbank. Ermittelt wird eine Vielzahl von Werten zur Auslastung von Prozessor, RAM, Netzwerk und Festplatte.

3 Lastprognose für WattDB

Bevor ein neues Prognoseverfahren für WattDB entwickelt wird, soll im Folgenden klar definiert werden, welche Charakteristiken dieses Verfahren aufweisen soll. Dazu definieren wir zunächst die Ziele und skizzieren dann die grundlegenden Ideen hinter der langfristigen Mustererkennung sowie der kurzfristigen Trendanalyse.

Pessimismus Eine Prognose soll die Last eher überschätzen als unterschätzen. Andernfalls führt dies dazu, dass nicht genügend Knoten für die tatsächliche Last aktiv sind und so verlängerte Antwortzeiten entstehen. Bei Lastüberschätzung sind zwar mehr Knoten als nötig aktiv, dies hat aber keine negativen Auswirkungen auf die Systemleistung, es schadet nur der Energieproportionalität. Trotz pessimistischer Prognose bleibt die Energieproportionalität von WattDB jedoch weitaus besser als die eines vergleichbaren Einzel-Servers. Konkret soll die Prognose nur in maximal 10% der Fälle die tatsächliche Last (jedoch nur in geringem Maße) unterschätzen. 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.

Geringe Prognosefehler 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 mittleren Prognosefehler von mehr als 10% führen, so würde meist mindestens ein Knoten zu viel prognostiziert. In diesem Fall wäre es aber sinnvoller, diesen Knoten stets als Reserveknoten zu betreiben und keine Prognose durchzuführen.

Vertretbarer Overhead Ein vertretbarer Overhead ist 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.

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. Daher soll es Ziel sein, dass die entwickelte Lastprognose nicht häufiger als die lineare Regression die Last unterschätzt.

Berücksichtigung langfristiger Muster Da Lastverläufe 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 und das Starten zusätzlicher Knoten rechtzeitig veranlassen.
Zur Mustererkennung benötigen Systeme eine gewisse Zeit zur Datensammlung. Ein allgemeines Prognoseverfahren braucht einige Jahre, um wichtige Konzepte wie Wochen, Monate, Jahre, Schaltjahre oder Sommerzeit zu lernen und darauf basierende Muster zu erkennen. Um eine deutliche Energieersparnis sofort zu erreichen, muss das Verfahren langfristige Muster deutlich schneller erkennen. Dazu sind sie mit genügend Startwissen zu versorgen, um langfristige Muster bereits nach dreimaligem Auftritt zu erkennen.

3.1 Längerfristige Mustererkennung

Die größte Herausforderung, die sich aus der genannten Problemstellung ergibt, ist die schnelle Erkennung langfristiger Muster bei gleichzeitig geringem Overhead. Deshalb kommen viele Verfahren [Sch07], z. B. der Einsatz eines künstlichen neuronalen Netzes zur Mustererkennung, nicht in Frage. Das hier entwickelte Prognoseverfahren kennt häufig auftretende Mustertypen, wie *tägliche, wöchentliche, monatliche* oder *jährliche* Muster, sowie *spezielle Ereignisse* (siehe Tab. 1). Ist eine Prognose für einen Zeitpunkt erforderlich, prüft es, ob für diesen Zeitpunkt in der Vergangenheit solche Muster erkennbar sind. Wird ein Muster erkannt, soll es entsprechend in die Prognose einfließen.
Die Annahme solcher 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 stärker ausgeprägten Mustern, als es Benutzeranfragen tun. Da auch solche Systeme oft stündliche, tägliche, wöchentliche, monatliche oder jährliche Berichte erzeugen, folgen auch sie den gleichen Mustertypen wie OLTP-Systeme.

3.2 Kurzfristige Trendanalyse

Die Erkennung langfristiger Muster reicht für eine zuverlässige Lastprognose nicht aus. Es sind stets gravierende Abweichungen auch von stark ausgeprägten Mustern zu erwarten. 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.

Lineare Regression zur kurzfristigen Trendanalyse Da es bereits einige geeignete Verfahren zur kurzfristigen Trendprognose gibt, wollen wir ein geeignetes Prognoseverfahren an die Problemstellung anpassen. In [Sch07] wird die Regressionsanalyse als für kurzfristige Trendanalyse geeignet herausgestellt. Da ein linearer Lastverlauf am häufigsten anzutreffen ist, wird vereinfachend die lineare Regression genutzt und darauf verzichtet,

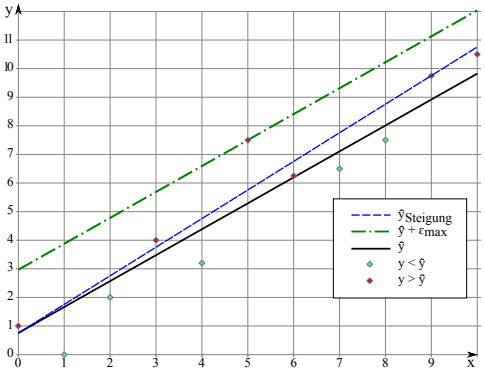


Abb. 2: Pessimistische lineare Einfachregression

zusätzlich eine exponentielle Regressionsanalyse durchzuführen. 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 4.2 auch evaluiert, wie viele vergangene Datenwerte einbezogen werden sollen, um möglichst gute Ergebnisse zu erhalten.

Pessimistische kurzfristige Trendanalyse Ziel der Methode der kleinsten Quadrate ist es, einen möglichst geringen Abstand zwischen der Regressionsgeraden und den Messpunkten 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 $\hat{y}$, die sich durch die Methode der kleinsten Quadrate ergibt, angepasst wird. Im Beispiel (siehe Abb. 2) 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. Zwei mögliche Anpassungen sind in Abb. 2 skizziert. Beispielsweise könnte man die Steigung so anpassen, dass bei positiver Steigung diese pauschal um 10% höher und bei negativer Steigung pauschal um 10% niedriger angesetzt wird. In Abb. 2 ist dies als $\hat{y}_{Steigung}$ eingezeichnet. Bei einem streng linearen Lastverlauf führt eine solche Anpassung zur Einrechnung einer gewissen Reserve. Wünschenswert wäre es aber, die Prognose umso pessimistischer zu gestalten, je stärker die ursprüngliche Regressionsgerade die vergangenen Werte unterschätzt hat, was durch Anpassung der Steigung kaum zu erreichen ist.
Bei Erhöhung des Ordinatenabschnitts dagegen kann einfach festgelegt werden, wie viele der vergangenen Werte unterschätzt werden dürfen. Verschiebt man die Gerade um die maximale Abweichung ε_{max} nach oben, führt dies dazu, dass die Gerade den am stärksten unterschätzten Punkt schneidet und alle anderen Punkte darunter liegen (siehe Abb. 2). Mit dieser Anpassung wird also kein vergangener Wert unterschätzt. Da das Maximum oft von einem statistischen Ausreißer bestimmt wird, ergibt sich eine sehr pessimistische Prognose. Andererseits soll das System ja ebensolche Lastspitzen bewältigen können.

3.3 Implementierung

Wie in Kapitel 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.

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** verknüpft sind. Jede der **SELECT**-Queries repräsentiert einen Mustertypen. Als Beispiel soll hier eine solche **SELECT**-Query genügen, deren **WHERE**-Klausel für wöchentliche Muster alle Tupel auswählt, deren Wochentag dem des zu prognostizierenden Tages entspricht, also vereinfacht wie folgt:

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

Pessimistische Berechnung des Prognosewertes für die Muster Durch dieses Verfahren werden Tupel durch **WHERE**-Klauseln ausgewählt, es fehlt aber noch die Aggregation der Werte zu einem Prognosewert. Da die Zielsetzung 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 Anfrage sähe dann beispielsweise wie folgt aus:

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

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 Anfragen wird dazu einfach das obere Quartil über die zu untersuchende Spalte (hier cpuLoad) eingesetzt.

Bewertung der Muster Bei dieser Verfahrensweise gibt jedes Muster 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. (1) als der durchschnittliche quadratische Abstand zum arithmetischen Mittel [Sch07].

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

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. (2) als Wurzel der Varianz (auch *Standardabweichung* s genannt) geteilt durch das arithmetische Mittel definiert [Sch07].

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

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, wie z. B.:

```
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 genauso wie **upper_quartile** in einer SQLite-Erweiterung enthalten ist, sowie die Funktion **avg** zurückgegriffen (s. Gl. (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. Ein beispielhaftes Ergebnis dieser Query ist in Tab. 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“ und „islamNewYear“. Die SQL-Query generiert also für jeden speziellen Tag ein eigenes Muster. Am Variationskoeffizienten V_i kann nun abgeschätzt werden, wie stark das jeweilige Muster ausgeprägt ist. Den kleinsten Variationskoeffizienten hat hier das Ostersonntag-Muster, wohl weil die Query einen Monat vor Ostern ausgeführt wurde. Danach folgen die jährlichen Muster mit Tag, Monat und Stunde bzw. nur Monat und Stunde. Diese sind dicht gefolgt vom monatlichen Muster des n-ten Wochentages (der 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.

Verrechnung aller Muster zu einem Prognosewert Jedes Muster liefert einen eigenen Prognosewert P_i (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. 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 $0,18$ stellte sich im Verlauf der Evaluation als gute Obergrenze heraus (s. Kapitel 4.1). Die Idee zur Berechnung eines Gesamt-Prognosewertes ist es nun, einen *gewichteten Durchschnitt* über die Prognosewerte der einzelnen Muster zu errechnen (Gl. (3)).

$$P = \frac{\sum_{i=1}^{n}(W_i P_i)}{\sum_{i=1}^{n} W_i} \tag{3}$$

Die Berechnung der Gewichte W_i erfolgt anhand des Variationskoeffizienten (Gl. (4)).

Tabelle 1: 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	45,374	0,119	0,00372	0,169	5%
day + month + hour	44,877	0,055	0,01562	0,701	20%
hour + minute	43,245	0,122	0,00336	0,145	4%
month + hour	45,938	0,080	0,01000	0,459	13%
monthDaysLeft + hour	44,486	0,116	0,00409	0,182	5%
specialDay_easterSunday + hour	45,114	0,050	0,01690	0,762	22%
specialDay_islamNewYear + hour	43,479	0,126	0,00291	0,127	4%
weekday + hour	44,109	0,093	0,00756	0,334	10%
x weekdays left + hour	45,898	0,103	0,00592	0,272	8%
xth weekday + hour	44,604	0,093	0,00756	0,338	10%
Summe			0,07768	3,489	

$$W_i = (0,18 - V_i)^2 \tag{4}$$

Die Gewichte sollen umso größer sein, je stärker das Muster ist, also je kleiner der Variationskoeffizient ist. Damit also Muster mit einem kleinen Variationskoeffizienten 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.

$$P = \frac{\sum_{i=1}^{n}(W_i P_i)}{\sum_{i=1}^{n} W_i} = \frac{3,489}{0,07768} \approx 44,910 \tag{5}$$

Zum besseren Vergleich der Gewichte wurde zusätzlich in der letzten Spalte von Tab. 1 der Anteil berechnet, den das einzelne Gewicht W_i an der Summe aller Gewichte ausmacht.

Implementierung der kurzfristigen Trendanalyse In ähnlicher Weise ist die kurzfristige Trendanalyse in SQL implementiert. Eine einzelne SQL-Anfrage wird auf die Daten angewendet und gibt den gewünschten Prognosewert zurück. Bei der linearen Einfachregression in SQL wird darauf geachtet, dass die Prognose pessimistisch wird.

Verrechnung kurzfristiger Trendanalysen mit langfristigen Musterprognosen Nachdem sowohl die Prognose auf Basis langfristiger Muster als auch jene auf Basis kurzfristiger Trends verfügbar sind, ist noch zu klären, wie beide zu einer Endprognose verrechnet werden. Fällt die kurzfristige Prognose höher aus, so muss sie auf jeden Fall als Endprognose angesetzt werden. Das verlangt unsere pessimistische Zielsetzung, außerdem können spontane Abweichungen von langfristigen Mustern immer auftreten.
Nicht so klar ist dagegen die Endprognose, falls die langfristige Prognose höher ausfällt. Wird hier die kurzfristige Prognose angesetzt, wird die langfristige Prognose nie verwendet und langfristige Muster werden somit komplett ignoriert. Ganz pessimistisch könnte man in diesem Fall die langfristige Prognose ansetzen, was bedeuten würde, dass stets die höhere beider Prognosen genutzt wird. Um den Energieverbrauch möglichst gering zu halten, ohne langfristige Muster komplett zu ignorieren, sollte in diesem Fall jedoch das arithmetische Mittel der beiden Prognosewerte die Endprognose bilden.

4 Evaluation

Im Rahmen einer empirischen Evaluation soll nun überprüft werden, ob die Implementierung der Lastprognose den gestellten Zielen gerecht wird.

4.1 Evaluation der langfristigen Mustererkennung

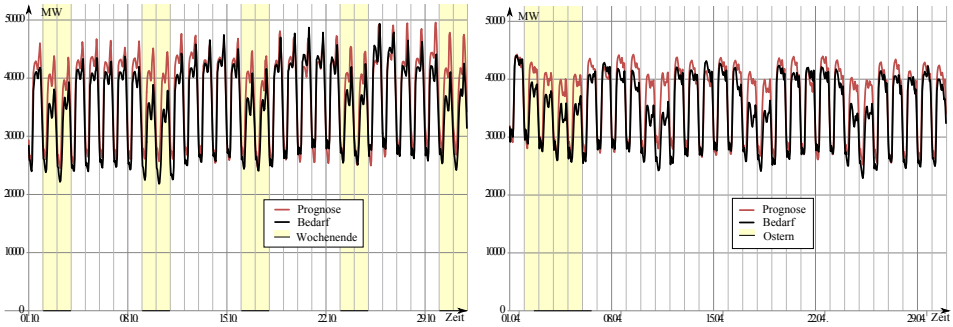
Da das WattDB-Monitoring bisher noch kein produktives System jahrelang aufgezeichnet hat, sind keine WattDB-Daten zur Evaluation verfügbar. Deshalb musste die langfristige Mustererkennung mit anderen Daten evaluiert werden. Anspruch an die Daten war, dass sie sich über mindestens drei Jahre erstrecken und langfristige Muster enthalten.
Als brauchbare Daten stellte sich schließlich der Gesamt-Energiebedarf von Großbritannien[1] heraus. 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 unseren Mustertypen entsprechen. So ist der Energiebedarf stark von täglichen Mustern geprägt. Als jährliches Muster zeigt sich u. a. ein höherer Energiebedarf in den Wintermonaten sowie ein geringerer Energiebedarf an Feiertagen. Diese Daten eignen sich daher auch zur Überprüfung beweglicher jährlicher Muster wie Ostern. Die Prognose soll im Jahr 2007 ohne Datenmaterial beginnen und mit immer mehr Daten fortlaufend berechnet werden, um das Lernverhalten zu beobachten.

Pessimismus und geringer Prognosefehler Der Anteil an Unterschätzungen ist im ersten Jahr 2007 mit ca. 45% noch sehr hoch. Hier zeigt sich, dass eine kurzfristige Prognose anfangs 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. Trotz Pessimismus bleibt auch der Prognosefehler unter den geforderten 10%.
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.

Vertretbarer Overhead Die Berechnung der Prognose auf Basis langfristiger Muster spielt eine geringere Rolle, da sie nicht häufig erforderlich ist. Ein weiterer zu beachtender Overhead stellt die Speicherung von Monitoring-Daten über einen längeren Zeitpunkt dar. Wie in [MD12] abgeschätzt ist, beläuft sich die Datenmenge aller vom Monitoring-System gespeicherten Daten auf weniger als 2 GB pro Jahr.

Berücksichtigung längerfristiger Muster Der interessanteste Teil der Evaluation hat nun herauszufinden, ob die Prognose langfristige Muster zuverlässig erkennt. Dazu beschränken wir uns hier auf zwei Beispiele, die den Verlauf der Prognose auf Basis langfristiger Muster mit dem der tatsächlichen Werte vergleicht. In Abb. 3a 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.

[1] http://www.nationalgrid.com/uk/Electricity/Data/Demand+Data/



(a) Prognose für Oktober 2010 **(b)** Prognose für April 2010

Abb. 3: Prognosen auf Basis langfristiger Muster

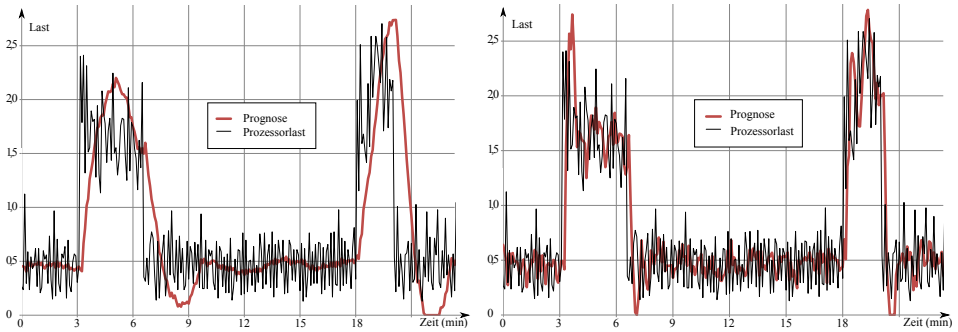
Mehrere Muster sind in der Grafik erkennbar; das tägliches Muster des Energiebedarfs ist besonders stark ausgeprägt. Dieses Muster ist sowohl im Verlauf des tatsächlichen Bedarfs als auch im Verlauf der Prognose erkennbar. Dies lässt darauf schließen, dass die Prognose dieses Muster erkennt und entsprechend 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.
Es bleibt noch zu untersuchen, ob bewegliche jährliche Muster wie etwa Ostern von der Prognose erkannt und berücksichtigt werden. In Abb. 3b 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. Die Prognose fällt wie gewünscht niedriger aus als an normalen Wochentagen, was dafür spricht, dass die Prognose auch dieses Muster korrekt erkennt.

4.2 Evaluation der kurzfristigen Trendanalyse

Die Evaluation der kurzfristigen Trendanalyse konnte mit Daten einer Anwendung durchgeführt werden. Dafür wurde die Prozessorlast eines DB-Servers der SPH AG über acht Tage aufgezeichnet, wobei alle 10 Sekunden ein Datenpunkt genommen wurde [SHK11]. Für alle Evaluationen wurde eine Prognose für die Last in 20 Sekunden berechnet (mit lookahead-Parameter $l = 2$) . Dies entspricht in etwa der Zeit zum Starten eines WattDB-Knotens.

Bestimmung des lookback-Parameters k. Für die kurzfristige Trendanalyse ist zunächst ein geeigneter Parameter k zu finden. Wenn alle 10 Sekunden ein Datenpunkt genommen wird, bedeutet $k = 6$, 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. 4a zu sehen.

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



(a) Kurzfristige Trendanalyse mit k=36 (6 Min.) **(b)** Kurzfristige Trendanalyse mit k=8 (80 Sek.)

Abb. 4: Kurzfristige Trendanalysen

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.

In Abb. 4b 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. Sehr kleine k unter 5 führen sowohl zu hohen Prognosefehlern als auch stärkerer Unterschätzung, während höhere k bessere durchschnittliche Ergebnisse liefern. Das erste lokale Minimum bei $k = 8$ wurde für die kurzfristige Trendanalyse gewählt. Die Datenpunkte der letzten 80 Sekunden einzubeziehen erscheint (wie man in Abb. 4b sieht) auch angemessen, wenn eine Prognose für die nächsten 20 Sekunden berechnet werden soll.

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

Pessimismus und geringer Prognosefehler 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. Mit unter 2% erreicht die kurzfristige Trendprognose einen ausgezeichneten mittleren Prognosefehler. Dieser liegt noch weit unter den geforderten 10%, sodass dieses Ziel für die Gesamtprognose realistisch ist.

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.
Im realen Einsatz hat das Prognoseverfahren deutlich mehr Daten zur Verfügung. Da aber die Query nur $k = 8$ Tupel betrachtet und diese mit einem Index über den Primärschlüssel auswählt, bleibt die Ausführungszeit auch bei größeren Datenvolumina annähernd gleich. Da nur alle 10 Sekunden neue Monitoring-Daten eintreffen, muss eine neue Trendanalyse auch nur dann erfolgen. Die erforderliche Berechnungszeit von < 2 Millisekunden (auf vergleichbarer Hardware) wird durch mögliche Energieeinsparungen mehr als amortisiert.

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.

4.3 Gesamtevaluation

Anhand der Daten für die Evaluation der kurzfristigen Trendanalyse soll untersucht werden, wie gut die Gesamtprognose in der ersten Woche ist. Da die langfristige Mustererkennung, wie in Kapitel 4.1 gezeigt, mit der Zeit besser wird, bedeutet dies, dass langfristig mit besseren Ergebnissen gerechnet werden kann. Die Gesamtevaluation bezieht sich auf die vorliegenden Daten eines DB-Servers [SHK11]. Mit der langfristigen Mustererkennung wurde für jede Minute eine Prognose berechnet. Da diese Prognose am ersten Tag keine Ergebnisse liefern kann, wird die Gesamtevaluation ab dem zweiten Tag über die verbleibenden sieben Tage durchgeführt. Die Trendanalyse wurde mit dem pessimistischen Verfahren auf Basis des Maximums, $k = 8$ und $l = 2$ für alle 10 Sekunden berechnet. Beide Prognosen wurden dann zu einer Gesamtprognose verrechnet (Kapitel 3.3).

Tab. 2 vergleicht die Ergebnisse der Gesamtevaluation mit denen der kurzfristigen Evaluation. 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 schon annähernd erreicht, mit zunehmendem Pessimismus der langfristigen Mustererkennung sollte dieser Wert schon in kurzer Zeit unter 10% fallen.
Unterschätzungen der Gesamtprognose sind offensichtlich bedeutungslos. Jedoch unterschätzen noch 0,37% der Prognosen die Last um mehr als 10%, was nach Zielsetzung nie der Fall sein sollte. Auch hier ist davon auszugehen, dass die langfristige Mustererkennung

Tabelle 2: 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%

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.

5 Ausblick und Fazit

Trotz der guten Ergebnisse sind weitere Optimierungen denkbar: Es kann versucht werden, bessere Prognosen mit einem geringeren Prognosefehler oder geringeren Unterschätzungen zu erreichen. Andererseits lässt sich den Overhead der Prognose weiter minimieren.
Für WattDB wurde zunächst das Monitoring-Framework und nun das Verfahren zur Lastprognose entwickelt. Doch so lange die Prognosewerte nicht zur Steuerung des DBMS genutzt werden, sind sie wertlos. Es ist daher als nächster Schritt nötig, dass WattDB die prognostizierten Werte interpretiert und daraufhin Knoten zu- und abschaltet.

Aktivierung von Knoten Unser Verfahren prognostiziert zu erwartende Lastwerte, um durch rechtzeitiges Zu- und Abschalten von Knoten eine ausreichende Rechenkapazität vorzuhalten und so die Energieproportionalität des Systems anzunähern. Gl. (6) skizziert dazu ein sehr einfaches Verfahren zur Berechnung der benötigten Knoten.

$$Knoten_{benötigt} = \left\lceil \frac{Prognose}{Kapazität} * Knoten_{gesamt} \right\rceil \tag{6}$$

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 eine Heuristik (z.B. Maximum) genügt oder eine deutlich komplexere Entscheidungsfindung nötig ist, ist noch zu untersuchen.

ECA-Regelwerk für WattDB Um eine optimale Steuerung von WattDB, insbesondere der Anpassung der Clustergröße, zu erreichen, ist ein *Event-Condition-Action*-Regelwerk (ECA) zu entwerfen. Ereignisse könnten z.B. der Ablauf eines Timers, 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. In Tab. 3 sind einfache Beispiele solcher Regeln definiert. Ein ECA-Regelwerk könnte aber auch deutlich komplexere Regeln enthalten, welche auf die unterschiedlichen Kennzahlen für CPU, RAM, Festplatte und Netzwerk eingehen könnten.

Weitere Nutzung des Prognoseverfahrens Langfristige Muster sind in vielen Lastverläufen verbreitet. Obwohl für DBMS-Lasten entwickelt wurde, könnte deshalb unser Prognoseverfahren auch in anderen verteilten Systemen genutzt werden. Geeignet wäre sie

Tabelle 3: ECA-Regeln für die Steuerung von WattDB (Beispiel)

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: $Knoten_{benötigt} - Knoten_{aktiv}$)

für jedes verteilte System, das die Clustergröße dynamisch anpassen kann und langfristige Muster in den Lastverläufen zeigt, welche den verwendeten Mustertypen entsprechen. Denkbar wäre z.B. der Einsatz in einem MapReduce-Cluster [DG08]. Eine andere 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, werden Cluster zukünftig immer mehr zum Einsatz kommen. Damit wächst auch der Bedarf, die Last solcher Systeme zu prognostizieren.

Fazit In der Evaluation konnte gezeigt werden, dass sich unsere Lastprognose wie erhofft verhält und den gesetzten Zielsetzungen gerecht wird. Durch die Aufteilung in eine kurzfristige Trendanalyse und eine langfristige Mustererkennung ist es gelungen, kurzfristige Lastspitzen und Lasteinbrüche als auch wiederkehrende Muster zu berücksichtigen. Beide Teilprognosen wurden dabei konsequent pessimistisch konzipiert und tendieren deshalb eher zum Überschätzen als zum Unterschätzen, wobei der durchschnittliche Prognosefehler deutlich unter der Kapazität eines Knotens liegt.

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.

Literatur

[DG08] Jeffrey Dean und Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Januar 2008.

[HHOS11] Theo Härder, Volker Hudlet, Yi Ou und Daniel Schall. Energy Efficiency Is Not Enough, Energy Proportionality Is Needed! In *Database Systems for Adanced Applications*, LNCS 6637, Seiten 226–239, 2011.

[MD12] Pedro Martins Dusso. A Monitoring System for WattDB: An Energy-Proportional Databsae Cluster, Januar 2012.

[Sch07] P.M. Schulze. *Beschreibende Statistik*. Oldenbourg, 2007.

[SH11] Daniel Schall und Volker Hudlet. WattDB: an energy-proportional cluster of wimpy nodes. In *Proc. SIGMOD*, Seiten 1229–1232, 2011.

[SHK11] Daniel Schall, Volker Höfner und Manuel Kern. Towards an Enhanced Benchmark Advocating Energy-Efficient Systems. In *Proc. TPCTC*, August 2011.

[THS10] Dimitris Tsirogiannis, Stavros Harizopoulos und Mehul A. Shah. Analyzing the energy efficiency of a database server. In *Proc. SIGMOD*, Seiten 231–242, 2010.