Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften

Department of Informatics
Scientific Computing

Hausarbeit im Seminar: Systemmonitoring unter Linux

CPU Counter

Arno Sagawe

arno.sagawe@gmx.de

Studiengang Wirtschaftsinformatik

Matr.-Nr. 5894143

Fachsemester 8

Betreuer: Prof. Dr. Thomas Ludwig

Julian Kunkel

Inhaltsverzeichnis

1 Systemmonitoring

Im Folgenden wird ein kurzer Überblick über das was Systemmonitoring bedeutet bzw. umfasst gegeben; welche Protokolle beim Einsammeln von Systeminformationen verwendet werden können, wie der Zugriff auf die gesammelten Daten erfolgen kann, welche Methoden (Modus) bei der Nutzung eines Systemmonitors genutzt werden können.

Bei einem Systemmonitor (engl. *system monitor*, SM) handelt es sich um einen Prozess der Daten über (s)ein System bzw. eine Anwendung sammelt und speichert. Damit ein Systemmonitor funktioniert sind zwei Konfigurationen vorzunehmen:

1. Die Anwendung (SM) muss für das eigentliche Monitoren konfiguriert werden.
2. Das System muss so konfiguriert werden, sodass es überwacht werden kann.

Die Anwendung (SM) benötigt Informationen (bspw. darüber wo relevante Logfiles liegen). Wenn die Anwendung ausgeführt wird muss bekannt sein was wie zu monitoren ist. Daher sind einige Fragen beim Entwurf eines Systemmonitor zu beantworten, bspw: Wie soll der SM konfiguriert werden? Welches Protokoll soll zum Einsammeln der Daten genutzt bzw. entwickelt werden? Wie wirkt sich das Monitoren, d.h. der Einsatz des SM auf die Leistung des Systems aus? Wie werden die relevanten Systemdaten gesammelt?

1.1 Protokoll

Es existiert eine Menge an Werkzeugen (engl. *tools*) für das Sammeln von Systemdaten, welche das Simple Network Management Protocol (SNMP) nutzen. Die meisten Betriebssysteme, Hardware- bzw. Netzwerkkomponenten unterstützen dieses Standardprotokoll, auf diese oder jene Art. Um die SNMP-Daten zu interpretieren werden spezielle Tools verwendet (meist proprietäre Herstellersoftware oder eine Managementinforamtionsbasis [MIB], welche die gesammelten SNMP-Daten um ihre Semantik anreichert). Der Vorteile des SNMP-Protokolls sind die geringen Anforderung an die Bandbreite eines Netzwerkes und die weite Verbreitung von SNMP in der Industrie. Des Weiteren können Protokolle wie CORBA, JMX, TCP/IP für die Überwachung von (verteilten) System eingesetzt werden.

1.2 Datenzugriff

Die Frage bzgl. des Zugriffs auf die zu sammelnden Daten beinhaltet auch die Frage nach der Schnittstelle (engl. *interface*) über die der SM die gesammelten Daten anderen Anwendungen zur Verfügung stellt; Beispiel: Ein SM ist als CORBA-Server implementiert und bietet den CORBA-Clients die Möglichkeit Informationen über andere CORBA-Clients zu erhalten bzw. sich anderen mitzuteilen.

Der Systemmonitor kann die gesammelten Daten direkt in eine Datenbank schreiben, die von anderen Softwareeinheiten, die nicht zu dem SM gehören, eingesehen und ausgewertet werden können. Dieser Ansatz ist insofern kritisch, als dass das Datenbankschema für alle anderen Anwendungen verbindlich ist. Besser ist, wenn der SM als Wrapper konzipiert wird (inkl. Persistenzmechanismen und Schnittstelle für andere Anwendungen, um die Daten abrufen zu können).

Zum Sammeln der Daten können drei Methoden bzw. Modus werdet werden, nämlich: monitor poll, agent push, and a hybrid scheme.

1.2.1 Poll Monitor

Bei dieser Methode werden (statistisch) Daten bzw. Informationen über den aktuellen Status einer (externen) Komponente durch eine Client-Software (synchron) gesammelt (engl. *polling*). Beispielsweise können Komponenten von einem SM „gepollt“ werden via SNMP, oder indem Skripte (via Telnet/SSH innerhalb der Komponenten) ausgeführt werden, oder es werden betriebssystemspezifische Kommandos (in der Laufzeitumgebung der Komponenten) abgesetzt, oder die von den Komponenten bereitgestellten *state output files* (Log-, Protokolldatei) werden ausgelesen.

Der Vorteil beim *poll monitor* sind die geringen Auswirkungen auf die Komponente, die „gepollt“ wird. Die CPU der Komponente wird nur während des Pollings beansprucht, und wird zwischen den Pollingaufrufen nicht zusätzlich belastet. Nachteilig am poll monitor sind die möglichen unbeabsichtigt langen Pollingperioden (also die Zeit für das Polling), die entstehen, wenn das Polling länger dauert als erwartet.

1.2.2 Agent push

Beim *agent push* senden die Komponenten die (lokal) gesammelten Daten (von sich aus) an den SM. Das versenden der Daten kann periodisch erfolgen oder via Request, also per Aufforderung des SM an die Komponente. Der Vorteil an diesem Modus ist, dass der SM nur während des Akzeptierens und des Speicherns der eingesammelten Daten belastet wird, und, dass der SM die gesam-

melten Daten in einem für ihn lesbaren Format erhält. Der Nachteil ist, dass der Pollingzyklus und die Optionen für das Polling dezentral im (verteilten) System gehalten werden. Änderungen an der Logik für das Monitoren (bspw. Zyklen und Optionen für das Monitoren, sowie die Software) müssen an alle Komponenten kommuniziert werden.

1.2.3 Hybrid mode

Eine Kombination des agent mode und des poll monitor ist der hybride Ansatz (engl. *hybrid mode*), bei dem die Systemkonfiguration bestimmt wo Daten gesammelt werden, entweder auf dem SM oder der Komponente (engl. *agent*). Wenn eine Komponente (Gerät, Anwendung) startet kann dieses selbständig bestimmen wie die Daten für das Systemmonitoring bereitgestellt werden.

2 Performance Counter

Die Performance Counter eines Prozessors wurden zur Leistungsüberwachung während der Programmausführung entwickelt (mittels Mitlesen bzw. Mitschreiben). Derzeit besitzt nahezu jede moderne CPU eine Vielzahl unterschiedlicher Counter, mit denen sich Events während des Programmablaufs protokollieren lassen. Der Softwaredesigner kann sein Programm während der Ausführung im Hinblick auf Programmablauf, Programmperformance und Programmfehlverhalten beobachten. Die Informationen der Countern können mit spezieller Software weiter verarbeitet und in transformierter Form analysiert bzw. ausgewertet werden (z.B. durch Diagramme, Tabellen usw.; siehe Abschnitte 3.2 und 3.3). Die Informationen können genutzt werden um Verbesserung der System-, der Programm- sowie der Compilerleistung zu erzielen[1].

Beispielsweise können Performance Countern genutzt werden, um die Anzahl der Instruktionen, welche die CPU ausführen wird, vorhergesagt werden, ebenso wie oft er Treffer im L1-Cache landete, wie viele Sprünge er richtig oder falsch vorhersagte, wie oft die zweite Pipeline zum Einsatz kam, die Anzahl von Fließkommaoperationen die ausgeführt wurden. Programme können hierdurch auf die Hardware abgestimmt entwickelt werden. Im Folgenden werden die Möglichkeiten der Leistungsüberwachung eines Intel Pentium 4 (P4) Prozessors exemplarisch vorgestellt.

2.1 Register im Pentium 4

Zur kurzzeitigen Speicherung von Angaben und Informationen, die zur weiteren Verarbeitung zur Verfügung stehen müssen, sind die Register als feste Bestandteile der Prozessoren eingeführt worden. Sie haben in der Regel eine Kapazität von wenigen Bytes und werden über einen Namen angesprochen.

2.2 Model Specific Registers (MSRs)

Die MSRs sind entwickelt worden, um eine Anzahl unterschiedlicher Hard- und Softwarefunktionen zu überwachen und zu steuern, inkl. der Performance Counter Ereignisse. Die Implementierung der MSRs variiert zwischen Prozessorgenerationen. Die MSRs müssen für ihre jeweiligen Einsatzgebiete konfiguriert und initialisiert werden, bevor sie für diese Aufgaben genutzt werden können (wie z.B. für das Perfomance Counting oder auch für Hardware- bzw. Systemchecks,

1 Dieser Abschnitt basiert auszugsweise auf der Seminararbeit „Performance Counter im Pentium 4“ von Michael Pitz aus dem Jahre 2002 und dem „IA32 Intel Architecture Software Developer's Manual, Volume 3“.

Vgl. Abschnitt 1). Das Auslesen und Beschreiben der MSRs funktioniert mit dem Befehl Read MSR (kurz: RMSR) bzw. Write MSR (kurz: WMSR).

2.3 Event Selection Contol Register (ESCR) MSRs

Mit geeigneter Software werden über die ESCR MSRs die speziellen Ereignisse (Events) ausgewählt, die von Countern gezählt werden sollen. Jedes ESCR ist mit einem Paar von Countern (i.d.R. zwei) verbunden und jedem Performance Counter stehen verschiedene ESCRs zur Verfügung.

2.4 Counter Configuration Control Registers (CCCR)s

Über die ESCR Flags und Felder lassen sich nur die Ereignisse auswählen, die zu zählen sind. Counter Configuration Control Register (CCCR) werden benötigt, um den Counter zu initialisieren. Die Überwachung des Zählvorgangs übernehmen nun die CCCR automatisch.

2.5 Debug Store (DS) Mechanismus

Der Debug Store ist ein durch Software festgelegter Bereich im Speicher, der dazu dient, Informationen vor zu halten, wie:

a) **Branch Records**: Ist das Branch Trace Store (BTS) Flag Bit in der IA32_DEBUGCTL MSR gesetzt, so werden die Sprungdaten, die im Zwischenspeicher des BTS auflaufen, in den reservierten Bereich des DS verschoben; sobald ein Sprung ausgeführt wird, ein Interrupt oder eine Ausnahme auftritt, werden diese Ereignisse im DS Bereich gespeichert.

b) **PEBS Records**: Wird ein Performance Counter so konfiguriert, dass er Precise Event-Based Sampling (PEBS) Daten überwacht und sammelt, so werden diese ebenfalls im DS Bereich gespeichert. PEBS-Datensätze entstehen, wenn ein Counterüberlauf passiert. In dem gespeicherten Datensatz sind alle Zustände der acht sog. Allzweck Register (general purpose registers), des EIP Registers und des EFLAGS Registers zum Zeitpunkt des Überlaufs enthalten.

Alle 18 Performance Counter des P4 sind jeweils 40 Bit groß. Für die P4 CPU wurde extra der Read Performance Monitoring Counter (RDPMC) Befehl implementiert. Über diesen kann man auswählen, ob die ganzen 40 Bit ausgelesen werden sollen oder nur die ersten 32 Bit. Es ist nur sinnvoll, die ersten 32 Bit auszulesen, wenn sichergestellt ist, dass im Counter keine Zahl größer 232 steht.

Um die verschiedenen Ereignisse, die Einfluss auf die Leistung haben, zu zählen, gibt es folgende Einrichtungen im P4 Prozessor:

- **IA32_MISC_ENABLE MSR**: zeigt die Verfügbarkeit des performance monitoring und der precise event-based sampling (PEBS) in einem IA-32 Prozessor an.
- **45 ESCR MSRs**: dient dazu die Ereignisse auszuwählen, die von den speziellen performance countern gezählt werden sollen.
- **18 performance counter model specific registers**: dienen dazu die Ereignisse zu zählen, die ausgewählt wurden.
- **Debug Store (DS)**: liegt in einem geschützten Bereich des Speichers der die PEBS Datensätze und Sprungdaten (branch records) speichert.
- **IA32_DS_AREA MSR**: stellt die Verbindung zum DS geschützten Bereich her.
- **DS Funktionsflag (Bit 21)**: dient dazu die CPUID Anweisung auszugegeben. Die CPUID zeigt die Verfügbarkeit des DS Mechanismus in einem IA-32 Prozessor an.
- **IA32_PEBS_ENABLE MSR**: stellt die PEBS Einrichtung und das wiederholte Identifizieren zur Verfügung.
- **Des Weiteren**: Ein Satz vordefinierter Ereignisse und Ereignismetriken, die zur Vereinfachung bei der Aktivierung und Bedienung spezieller Performance Counter Ereignisse vorgesehen sind.

Die 18 Performance Counter können in vier Gruppen eingeteilt werden:

I. **Die BPU Gruppe**:

1. MSR_BPU_COUNTER0 und MSR_BPU_COUNTER1
2. MSR_BPU_COUNTER2 und MSR_BPU_COUNTER3.

II. **Die MS Gruppe**:

1. MSR_MS_COUNTER0 und MSR_MS_COUNTER1.
2. MSR_MS_COUNTER2 und MSR_MS_COUNTER3.

III. **Die FLAME Gruppe**:

1. MSR_FLAME_COUNTER0 und MSR_FLAME_COUNTER1.
2. MSR_FLAME_COUNTER2 und MSR_FLAME_COUNTER3.

IV. **Die IQ Gruppe**:

1. MSR_IQ_COUNTER0 und MSR_IQ_COUNTER1.
2. MSR_IQ_COUNTER2 und MSR_IQ_COUNTER3.
3. MSR_IQ_COUNTER4 und MSR_IQ_COUNTER5.

2.6 Zwischenfazit Performance Counter

Das Nutzen der Performance Counter ist eine eigene Art des Systemmonitorings. Das oben vorgestellte überwachen von Komponenten (via SNMP bspw.) dient dem Überprüfen von Komponenten im produktiven Einsatz, um die Verfügbarkeit bzw. entstehende Engpässe und vorhandene Probleme anzuzeigen. Das Überwachen von Programmen per Performance Counter kommt eher bei deren Entwicklung und Wartung, als im produktiven Einsatz vor[2].

Seit der GNU/Linux Kernel Version 2.6.31 ist es möglich die Performance Counter zu nutzen. Dazu kann das Tool perf (siehe: Abschnitt 3.3 Perf) genutzt werden. Eine Liste der Performance Counter auf einem Ubuntu 10.04 GNU/Linux System mit Intel Core2 Duo CPU T5670 (1.80GHz) ist im Anhang zu sehen. Im Kapitel 3 werden Werkzeuge und deren Verwendung vorgestellt, um auf die Performance Counter zu zugreifen.

2 Ausnahmen bestätigen die Regel. Denkbares Szenario für den Einsatz von Performance Counter im Regelbetrieb: Mitarbeiter eines Rechenzentrum überwachen mittels Performance-Counter-Analysen die von den Kunden aufgespielte Software, welche auf der Hardware des Rechenzentrumbetreibers ausgeführt wird. Sollte die Software der Kunden nicht optimal an die Hardware der Systeme des Rechenzentrums angepasst sein, so können die Mitarbeiter Empfehlungen geben oder Vorgaben machen, um die Engpässe in der Software zu beseitigen.

3 Programme im Überblick

Für das OpenSource Betriebssystem GNU/Linux sind diverse meist kostenfreie und ebenfalls quelloffene Systemmonitoringprogramme verfügbar. Programme dieser Art gibt es als kleine Tools aber auch als Enterprise Solutions. Hier werden ein paar Beispiele gegeben:

top ermöglicht dem Anwender oder Systemverwalter eine kontinuierliche Ausgabe der wichtigsten Informationen zur Ressourcenausnutzung. Periodisch wird eine zweiteilige Tabelle ausgegeben. Deren erster, oberer Teil enthält allgemeine Informationen über das System als grobe Statistik der Prozessinformationen, wie die mittlere Systemlast, Anzahl der User, die CPU Auslastung.[3]

iostat liefert Input/Output Statistiken für Devices, Partitionen und Netzwerk-Dateisysteme (NFS). Daneben gibt es auch CPU Statistiken aus. Iostat ist Teil des sysstat Pakets[4]. **vmstat** (virtual memory statistics) ist ein wertvolles Monitoring-Tool, das sowohl Informationen zum Speicher als auch Informationen zum Block IO und zur CPU Aktivität ausgibt[5].

pmap zeigt den Speicherverbrauch an, den eine Anwendung generiert. Mit pmap können Bottlenecks aufgespürt werden.

Nagios ist ein äußerst vielseitiges Werkzeug, welches eine netzwerkweite Überwachen alle Systeme erlaubt. Möglich ist es den On-/Offline Status von Servern zu überwachen genauso wie den Status einzelner Dienste auf diesen Server, mit Hilfe von SNMP sogar Dinge wie die momentane Prozessorauslastung, Festplattenverbrauch, Performance Counter usw. Nagios wird oft für das unternehmensweite Monitoring verteilter Systemen verwendet.[6]

Im Folgenden werden drei weitere Systemmonitoringprogramme ausführlicher behandelt collectd, oprofile und prof. Diese drei Programme liegen vom Umfange zwischen den kleinen Tools (top, ntop, powertop) und der Enterprise Solution (Nagios).

3 [GZG01] Seite 962.

4 http://www.thomas-krenn.com/de/wiki/Linux_I/O_Performance_Messungen_mit_iostat

5 http://www.thomas-krenn.com/de/wiki/Linux_Performance_Messungen_mit_vmstat

6 http://www.nagios.org/

3.1 Collectd

3.1.1 Was ist Collectd?

Collectd ist ein Daemon[7] der periodisch Systemdaten erfasst und Statistiken über die Leistung (Performance) eines unixartigen Betriebssystems erstellt. Die Daten können lokal oder über das Netzwerk erhoben werden, welche in unterschiedlichsten Datenformen abgespeichert werden können (bspw. in einem für RRD lesbares Format oder als CSV-Datei). Die von collectd erstellten Statistiken können ausgewertet werden um Leistungsengpässe (Bottlenecks) in Systemen zuerkennen und um Maßnahmen einzuleiten mit denen diese Leistungsengpässe behoben werden. Collectd ist in C geschrieben, was, lt. dem Entwicklerteam, die Ausführung des Programmcodes performanter macht und die Portierung des

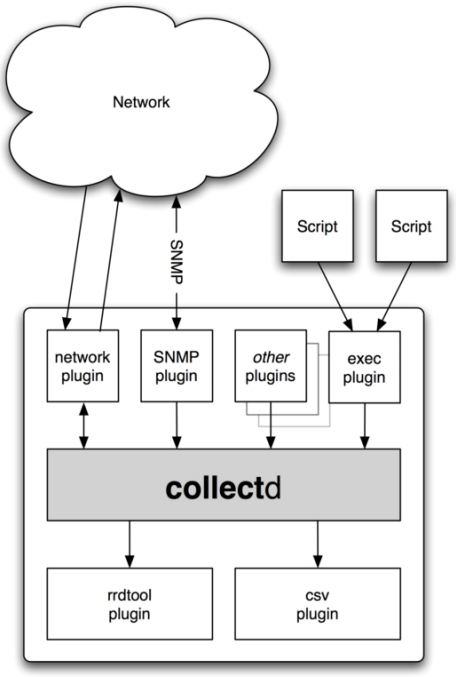


Abbildung: Collectd-Systemarchitektur[8]

Programmcodes ermöglicht. Das Programm Collectd kann mit Plugins hinsichtlich des Funktionsumfangs erweitert werden[9]. Die Plugins bieten die Möglichkeit Standardstatistiken (bspw. Informationen über Festplattenaktivitäten und die Ei-

7 Als Daemon wird unter Unix bezeichnet oder unixartigen Systemen ein Programm, das im Hintergrund abläuft und bestimmte Dienste zur Verfügung stellt bezeichnet. Interaktionen, zwischen Anwender und Anwendung (Daemon) finden dabei nur auf indirektem Wege statt, bspw. über Signale, Pipes und vor allem (Netzwerk-)Sockets.

8 http://www.collectd.org/images/architecture-schematic.png

9 Für collectd sind über 90 Plugins verfügbar; siehe: http://collectd.org/wiki/index.-php/Table_of_Plugins; Stand 10.06.2010;

genschaften der Festplatte) und Spezialstatistiken zu erstellen (bspw. Informationen über die aktuell instanziierten Managed Beans in einem MBeanServer der JMX nutzt). Die Statistiken werden u.a. als hochauflösenden Grafiken die Graphen darstellen angezeigt. Collectd kann in Netzwerküberwachungssysteme (bspw. Nagios) integriert werden. Des Weiteren ist Collectd in der Lage das Simple Network Management Protocol (SNMP) zu nutzen. Collectd wird häufig in Netzwerkkomponenten wie Switches, Router u.ä. Verwendet, aber auch in Embeded Systems, Thermostaten und Notstromaggregate (USVs).

3.1.2 Installation und Konfiguration

Im Standard Ubuntu 10.04 GNU/Linux Repository ist das Collectd Paket in der Version 4.8 enthalten[10]. Mit dem Paketverwaltungssystem kann das Paket installiert werden[11] (es werden alle zusätzlich benötigten Pakete mit installiert, wie bspw. rrd). Des Weiteren ist, ebenfalls mit der Paketverwaltung, ein Apache2 Webserver zu installieren, um die Bilder aus den rrd-Daten bequem im Browser einzusehen (dazu weiter unten mehr).

Nach der Installation von Collectd kann die Konfiguration vorgenommen werden; Ubuntu 10.04 GNU/Linux liegen die relevanten Dateien in /etc/collectd/[12]:

- ➔ **collectd.conf**: In dieser Datei werden die Grundkonfigurationen (wie bspw. Hostname) vorgenommen, sowie die zu verwendeten Plugins aktiviert und konfiguriert. Standardmäßig werden folgende Parameter aktiviert (siehe Anhang: Collectd - Standard). Für das Beispiel hier werden alle Plugins, bis auf logfile, cpu und cpufreq, deaktiviert (siehe Anhang: Seminar).

- ➔ **collection.conf**: Hier wird angegeben wo und wie gesammelten Daten gespeichert werden sollen.

- ➔ **filters.conf**: Die von Collectd gesammelten Daten können (um bspw. einer Datenproliferation entgegenzuwirken) auf der Basis von Filterregeln analysiert werden, um den eingehenden Datenstrom zu verwalten. Das Konzept der Filterregeln basiert auf dem OpenSource Paketfilter *ip_tables* für GNU/Linux.

10 Ein aktuelle Version kann von der Projekthomepage heruntergeladen werden; http://collectd.org/download.shtml.

11 Eine Installationsanleitung von *scratch* für Debian Etch GNU/Linux ist unter http://collectd.org/wiki/index.php/First_steps verfügbar.

12 Die Konfigurationsdateien werden beschrieben in *collectd.conf(5)* bzw. können in der Onlineversion eingesehen werden; http://collectd.org/documentation/manpages/collectd.conf.5.shtml.

- ➔ **thresholds.conf**: In dieser Konfigurationsdatei können Parameter (Schwellwerte) angegeben werden, die, wenn sie überschritten bzw. unterschritten oder erreicht werden, als Entscheidungsgrundlage für das versenden von Nachrichten genutzt werden. Damit ist Collectd (seit der Version 4.3) nicht mehr nur ein Programm, welches Daten einsammelt und in unterschiedlichen Statistiken darstellt, sondern auch ein Werkzeug um (verteilte) Systeme zu überwachen (Monitoring).

Nicht alle Dateien müssen konfiguriert werden. In der collectd.conf werden alle anderen Dateien ggf. inkludiert.

3.1.3 Anwendungsbeispiel

Nachdem die Konfiguration vorgenommen ist kann Collectd als Dienst (Daemon) gestartet werden. In diesem Beispiel werden nur die die Plugins `cpu` und `cpufreq` (siehe Anhang: Collectd - collectd.conf (Seminar)) geladen. Die Daten die Collectd sammelt werden standardmäßig in /var/lib/collectd/rrd/localhost gespeichert und liegen binär vor. Aus diesen Dateien sind nun PNG-Dateien (Bilder bzw. Graphen) zu erstellen. Dazu empfiehlt es sich ein Shell-Skript zu schreiben, da dieses später per crond als Job periodisch ausgeführt werden kann (siehe Anhang: Collectd - rrd2png.sh). Die erstellten Bilder zeigen einen Graphen je Plugin bzw. zu „collectenden“ Komponente und können mit den gängigen Bildanzeigeprogrammen dargestellt werden (siehe Anhang: Collectd - Graph). Eine weitere hier genutzte Möglichkeit ist, die Bilder auf einem Webserver (bspw. Apache2) bereitzustellen und via Browser zu betrachten.

Der Graph welcher von vom Plugin `cpu` generiert wird zeigt in einem Säulendiagramm die CPU-Belastung durch die Benutzer (grün), der Betriebssystem (rot), die Prioritäten (türkis) und die IO-Wait-States (blau). Die Daten werden unter Linux aus dem `/proc`-Filesystem ausgelesen und gesammelt. Es können noch detailierte Informationen im Graphen angegeben werden (Idle, Steal, SoftIRQ, IRQ etc.)[13].

Der vom Plugin cpufreq erstellte Graph zeigte die durchschnittliche Taktrate der CPU zu einem bestimmten Zeitpunkt (letzten 5 min. [rot], 10 min. [blau], 15 min. [grün]) an. Die Daten werden aus der Datei scaling_cur_freq unter /sys/devices/system/cpu/cpu*0*/cpufreq/ ausgelesen[14].

Für das Auffinden der Ursachen von Engpässen eignet sich Collectd nicht, jedoch für das Anzeigen von Engpässen.

13 Vgl. http://collectd.org/wiki/index.php/Plugin:CPU

14 Vgl. http://collectd.org/wiki/index.php/Plugin:CPUFreq

3.2 Oprofile

Ein Programmiere kann sein erstelltes Programm auf Leistungsengpässe (engl. *Bottlenecks*) hin untersuchen, in dem er den Quellcode seines Programms in einem Review analysiert oder er nutzt ein Programm, wie Oprofile , das ihm beim Aufspüren von Performance-Engpässe hilft.

3.2.1 Was ist Oprofile?

Oprofile sammelt zur Laufzeit Informationen moderner Prozessoren und wertet diese aus. Oprofile ist ein sog. Profiler[15] der komplexe, interagierende Programme und Bibliotheken „vermisst“ (von messen) und so eine Analyse des gesamten Systemverhaltens ermöglicht. Bei dem Zählen und Messen werden statistische[16] Informationen erhoben.

Oprofile bietet die Möglichkeit, bei bestimmten Ereignissen bis zu mehrere tausend Mal pro Sekunde spezifische Systemzustände zu protokollieren. Im einfachsten Fall stammen diese Ereignisse (engl. e*vents*) vom i*nstruction pointer* der CPU. Damit kann der Entwickler feststellen, welche Programmteile am häufigsten ausgeführt werden, eventuell einen Bottleneck aufspüren und dann für Optimierungen sorgen. Dieses periodische *sampling* von Events und Zuständen wird auf Kernel-Ebene entweder durch einen *system timer* realisiert oder - wesentlich effizienter - durch so genannte Performance Counter, die sich in fast allen modernen Prozessoren finden.

Gegenüber klassischen Lösungen, etwa dem speziellen Instrumentieren[17] der Programme durch einen Compiler (bspw. mit `gcc -pg`), hat eine solche Implementierung mehrere Vorteile: Es ist keine erneute Übersetzung der Programme und Bibliotheken aus den Quellcodes nötig und da auch kein zusätzlicher Instru-

15 Als Profiler werden Programmierwerkzeuge bezeichnet, die das Laufzeitverhalten von Software analysieren. Es gibt unterschiedliche Problembereiche in der Softwareentwicklung, die durch ineffiziente Programmierung ausgelöst werden. Ein Profiler hilft dem Entwickler durch Analyse und Vergleich von laufenden Programmen die Problembereiche aufzudecken. Daraus kann man Maßnahmen zur strukturellen und algorithmischen Verbesserung des Quellcodes ableiten.

16 Mittels der statistischen Auswertung wird die Programmanalyse nicht exakt mit jedem Programmbefehl einer Messung unterzogen. Es wird vielmehr in einem bestimmten zeitlichen Zyklus die Laufzeit gemessen. Dieses Verfahren nennt man auch Sampling. Der Profiler greift damit in bestimmten Taktzyklenabständen in dem Programmablauf ein und ermittelt damit stichprobenartig, welche Programmteile seit dem letzten Zyklus aufgerufen wurden. Daraus wird ein statistischer Mittelwert errechnet, der in das Ergebnis der Analyse einfließt. Bei der statistischen Auswertung wird das laufende Programm nicht verändert.

17 Beim Instrumentieren werden Informationen über Problemfälle in (verteilten) Anwendungen durch Platzieren von Ablaufverfolgungsanweisungen im Programmcode angezeigt. Des Weiteren können mit Ablaufverfolgungsanweisungen Anwendungen zur Laufzeit überwacht werden.

menten-Ballast in den Programmen steckt, beeinträchtigt auch kein Messcode den normalen Ablauf. Während des Profilings erhöht sich die Systemlast allerdings zwischen 1 und 10 Prozent. Dieser Wert hängt von der gewählten Sampling-Frequenz und der übrigen Systemauslastung ab.

Durch die Unterstützung von Performance Counter lassen sich detaillierte Daten zu extrem systeminternen Aspekten sammeln (L1-, L2, TLB-, Cache-Misses und -Hits, nicht ausgerichtete [misaligned] Referenzen, Pipeline Flush, Pipeline Stall, parallel ausgeführte Instruktionen [Pairing] u.v.a.m. je nach CPU). Oprofile unterstützt neben den 32-Bit-Prozessoren von Intel und AMD, auch Alpha, IA-64, PowerPC, Mips sowie Xscale-ARM-CPUs.

Oprofile besteht aus dem Kernel nahem Programmcode zum Erzeugen der Events aus einem Userspace-Daemon, der die zu sammelten Daten holt und zur späteren Analyse speichert bzw. bereitstellt. Werkzeuge zur späteren Analyse der Daten stellt das Oprofile Projekt ebenfalls bereit. Diese können die Ereignisse sowohl einzelnen Assembler-Instruktionen wie auch C- und C++-Code zuordnen und in diesen als Kommentar anfügen. Das Oprofile-Paket enthält obendrein ein grafisches Frontend, das auf dem Qt-Toolkit basiert[18].

3.2.2 Installation und Konfiguration

Im Standard Ubuntu 10.04 GNU/Linux Repository ist das Oprofile Paket in der Version 0.9.6 enthalten[19]. Mit dem Paketverwaltungssystem kann das Paket installiert werden[20] (es werden alle zusätzlich benötigten Pakete mit installiert, wie bspw. libopagent1). Damit Oprofile die Textpassagen anzeigen kann, welche in bestimmten Programmen zu Engpässen führen, sind die Debugging-Symbole der zu profilierenden Programme zu installieren (für die hier gezeigten Beispiele: imagemagick-dbg). Des Weiteren empfiehlt es sich, ebenfalls mit der Paketverwaltung, das Paket oprofile-gui, die grafische Benutzerschnittstelle für Oprofile, zu installieren.

Nach der Installation von collectd kann die Konfiguration vorgenommen werden. Zu konfigurieren sind unter anderem das Sampling-Intervall sowie die Events,

18 Der Abschnitt 3.2 basiert im Wesentlichen auf dem Artikel "Optimierung mit Profil" von René Rebe erschienen im Linux-Magazin 2006/07 abrufbar im Heftarchiv unter http://www.linux-magazin.de/Heft-Abo/Ausgaben/2006/07/Optimierung-mit-Profil und dem Wikipedia Artikel „Profiler“ abrufbar unter http://de.wikipedia.org/wiki/Profiler_(Programmierung); jeweils aufgerufen am: 11. Juni 2010)

19 Eine native Version kann von der Projekthomepage heruntergeladen werden; http://oprofile.sourceforge.net/download/. (Aufgerufen am: 11. Juni 2010)

20 Eine Installationsanleitung von *scratch* ist unter http://oprofile.sourceforge.net/doc/install.html verfügbar. (Aufgerufen am: 11. Juni 2010)

bei denen Daten gesammelt werden sollen. Optional kann der Entwickler auch das Kernel-Image angeben, um eine Aufschlüsselung der Kernelsymbole zu erhalten.

Beispiel:

```
opcontrol \
  --setup –vmlinux=/boot/vmlinux \
  --event=CPU_CLK_UNHALTED:500000:0:1:1 \
  --separate=library
```

Der Parameter `--vmlinux` gibt das Kernel-Image an (optional). Wird kein Kernal-Image verwendet erwartet Oprofile die Option `--no-vmlinux`. Das im obigen Beispiel spezifizierte Event `CPU_CLK_UNHALTED` steht für den CPU-Clock-Zyklus, der aktiv ist, wenn die CPU nicht angehalten ist. Das Kommande legt fest, dass Oprofile jedes 500000ste Event protokolliert (bei 2 GHz werden so 4000 Strichproben pro Sekunde erhoben). Die nachfolgenden Werte haben je nach Event eine andere Bedeutung. So lässt sich zum Beispiel festlegen, ob Events im Kernel oder im Userspace protokolliert werden sollen, hier mit `1:1` beides.

Eine vollständige Liste der im System verfügbaren Events liefert der Aufruf `opcontrol --list-events` (siehe Anhang). Die eigentliche Datenerfassung wird mit `opcontrol --start` gestartet. Nun kann der Entwickler jene Programme starten, deren Performance er analysieren will. Nach hinreichender Zeit ist das Datenerfassen zu stoppen mit `opcontrol --shutdown`.

3.2.3 Anwendungsbeispiel

Ein Beispiel für die Analysemöglichkeiten von Oprofile geben, wird ein isolierteres Problem untersucht: das Schärfen eines Bildes mit Imagemagick. Das Tool Convert soll eine 66 MByte große Tiff-Datei mit 16 Bit pro Farbkanal mit dem Unsharp-Mask-Filter bearbeiten:

```
convert –unsharp 4 prozessor.tiff prozessor.jpeg
```

Selbst auf einem Intel Core2 Duo CPU T5670 mit 1.80GHz dauert der Prozess 6,7 Sekunden. Ohne Angabe des Parameters `--exclude-dependent` listet Oreport detailliert die Event-Anteile der jeweiligen Bibliotheken auf. Um sich auf die zu optimierende Komponente zu konzentrieren, lässt sich die Ausgabe durch die Angabe eines Binary eingrenzen (siehe Anhang: Oprofile - opreport -t 0.5 /usr/bin/convert).

Ungefähr 65 % der Zeit verbringt das Programm nach diesem Ergebnis in der Bibliothek libMagick.so. Die konkreten Funktionen und Methoden, sprich die Symbole, liefert Opreport mit der Option `-l`, (siehe Anhang: Oprofile – opreport

-t 0.5 -l /usr/bin/convert). Hat der Entwickler sich einmal für die zu analysierenden Symbole entschieden, kommentiert `opannotate` den Quellcode mit den jeweiligen Sample-Anteilen pro Source-Zeile (siehe Anhang: Oprofile - opannotate -t 0.5 --source).

Falls keine Sourcen vorhanden sind, bleibt noch die Option `--assembly`. Damit kommentiert das Programm den disassemblierte Maschinencode. Wenn bei komplexen Ausdrücken die Statistik pro Zeilennummer zu ungenau ist und die exakten Instruktionen ausfindig gemacht werden müssen, hilft es manchmal, die Parameter `--source` und `--assembly` zu kombinieren. Damit versieht das Tool detaillierte Assembler-Listings mit Orientierungshilfen (siehe Anhang: Oprofile - opannotate -t 0.5 --source).

Die Verwendung anderer Profiling Counter Events, etwa Cache Misses, verläuft wie am obigen Beispiel beschrieben. Nach Angabe des entsprechenden Eventnamens mit `oprofile --setup --event`, kann der Entwickler `opreport` und `opannotate` entsprechend benutzen, um die gesammelten Daten seinem Quellcode zuzuordnen.

Oprofile liefert mittels umfassende Datensammlung Aufschluss darüber, womit der bzw. die Prozessoren ausgelastet sind, hinsichtlich der konsumierten Zeit. Die gesammelten Informationen ermöglichen es dem Entwicklern, Leistungsengpässe zu ermitteln und sich bei der Analyse auf jene eng eingegrenzten Bereiche zu konzentrieren, die besonders viel Rechenzeit beanspruchen oder ein sog. Bottleneck bilden. Wenn unter Verwendung von Oprofile problematische Stellen im Programmcode ausgemacht sind, sollte der Entwickler zunächst die Optimierungsanleitung des CPU-Herstellers zu rate ziehen, bevor er sich an Änderungen der eingesetzten Algorithmen wagt.

3.3 Perf

3.3.1 Was ist Perf?

Performance Counter sind spezielle Hardwareregister, welche in den meisten modern CPUs zur Verfügung stehen. Diese Register zählen die Anzahl unterschiedlicher Hardware-Events (wie bspw. ausgeführte Instruktionen, cache-misses suffered oder branches mispredicted) ohne den Prozessor bzw. den Kernel und laufende Anwendungen für den Erhalt der Informationen zusätzlich zu belasten. Des Weiteren sind diese Register in der Lage Interrupts abzusetzen, wenn beispielsweise ein Schwellwert (Wert eines Registers) überschritten wird, und kann daher genutzt werden den Programmcode, welcher auf der CPU läuft zu profilen[21].

Das Linux Performance Counter Subsystem abstrahiert von der Hardware und bietet die Möglichkeit pro Task, pro CPU und pro Workload Counter und ganze Countergruppen zu nutzen, und bietet obendrein Möglichkeiten des Samplings[22]. Neben der Hardwareabstraktion bietet perf auch eine Abstraktion auf „Software Event“ Ebene, wie minor/major page faults, task migrations, task context-switches and tracepoints.

Das neue Tool perf macht einen umfangreichen Gebrauch von den Performance Countern; perf kann genutzt werden für die Optimierung, zum Validieren und dem „Vermessen“ von Anwendungen, Workloads bzw. des gesamten Systems.

3.3.2 Installation und Konfiguration

Im Standard Ubuntu 10.04 GNU/Linux Repository ist das perf Paket nicht enthalten. Vor allem aber fehlt der GNU/Linux-Kernel in der Version 2.6.33, der für das Nutzen von Performance bzw. CPU Counter erforderlich ist. Der Kernel in besagter Version kann aus dem Kernel-PPA via Webbrowser heruntergeladen werden[23]. Die Pakete linux-headers-2.6.33-020633rc6_2.6.33-020633rc6_all.deb, linux-headers-2.6.33-020633rc6-generic_2.6.33-020633rc6_amd64.deb, linux-image-2.6.33-020633rc6-generic_2.6.33-020633rc6_amd64.deb sind herunterzuladen und später (in der Reihenfolge) zu installieren. In den Sourcen des GNU/Linux Kernels 2.6.33 von Ubuntu ist der Programmcode von perf leider nicht enthalten; perf muss daher aus den original Sourcen des GNU/Linux Kernels 2.6.33 heraus kompiliert werden. Damit das Kompilieren funktioniert sind

21 Vgl.: 2.2 Oprofile

22 Vgl.: 2.2 Oprofile

23 Das Ubuntu-Projekt stellt neue aktuelle Kernelversionen auf http://kernel.ubuntu.com/~kernel-ppa/mainline/v2.6.33/ bereit (Abgerufen am 15.06.2010)

weitere Pakete (via apt-get) zu installieren, nämlich: elfutils-devel, libelf-dev , binutils-dev und libdwarf-dev. In dem entpackten Originalkernel liegen die Sourcen von perf in tools/perf. In diesem Verzeichnis ist nun `make` auszuführen. Perf wird übersetzt. Ggf. kann mit `make install` perf systemweit installiert werden. Das Programm perf muss nicht konfiguriert werden.

Damit Perf die Textpassagen anzeigen kann, welche in bestimmten Programmen zu Engpässen führen, sind die Debugging-Symbole der zu profilierenden Programme zu installieren (für die hier gezeigten Beispiele: imagemagick-dbg)

3.3.3 Anwendungsbeispiel

Im Folgenden werden einige Beispiele für den Umgang mit perf geben. Weitere Bespiele finden sich auf der Projekthomepage[24].

Mit `perf list` können sich alle verfügbaren Performance Counter angezeigt werden; um die Tracepoint Event anzuzeigen müssen debugfs gemountet werden (`mount -t debugfs none /dbg`).

```
yggdrasil:~> perf list
 [...]
  cpu-cycles OR cycles  [Hardware event]
  instructions          [Hardware event]
  cache-references      [Hardware event]
  cache-misses          [Hardware event]
  branch-misses         [Hardware event]
  bus-cycles            [Hardware event]
  cpu-clock             [Software event]
  task-clock            [Software event]
  page-faults OR faults [Software event]
  minor-faults          [Software event]
  major-faults          [Software event]
 [...]
```

Einer (oder alle) dieser oben gelisteten Performance Counter kann genutzt und ausgelesen werden. Hier wird zum Beispiel `cpu-clock` ausgelesen bzgl. `convert -unsharp 4 prozessor.tiff prozessor.jpeg` und angezeigt:

```
yggdrasil:~> sudo perf stat -e cpu-clock convert -unsharp 4
prozessor.tiff prozessor.jpeg

Performance counter stats for 'convert -unsharp 4
prozessor.tiff prozessor.jpeg':

   10471.738892  cpu-clock-msecs

    6.711020484  seconds time elapsed
```

24 https://perf.wiki.kernel.org/index.php/Perf_examples

Des Weiteren können die Tracepoints verwendet werden um den workload zu profilieren:

```
yggdrasil:~> perf record -f -e cpu-clock convert -unsharp 4
prozessor.tiff prozessor.jpeg
[ perf record: Woken up 21 times to write data ]
[ perf record: Captured and wrote 2.563 MB perf.data
(~111964 samples) ]
```

Die gesammelten Daten können nun als Report ausgeben werden. Im Folgenden Beispiel ist zu sehen, welche Programmroutinen wie viel Zeit konsumieren. Ein Report wird mich `perf report` erstellt:

```
# Samples: 111847
#
# Overhead  Command  Shared Object                        Symbol
# ........  .......  ...................................  ..............................
#
    22.27%  convert  /lib/libc-2.11.1.so                  [.] memcpy
    21.96%  convert  /usr/lib/libMagickCore.so.2.0.1      [.] BlurImageChannel.omp_fn.9
    21.83%  convert  /usr/lib/libMagickCore.so.2.0.1      [.] BlurImageChannel.omp_fn.10
     6.17%  convert  /usr/lib/libjpeg.so.62.0.0           [.] 0x000000000087c3
     5.98%  convert  /usr/lib/libMagickCore.so.2.0.1      [.] GetVirtualPixelsFromNexus
     5.05%  convert  /usr/lib/libMagickCore.so.2.0.1      [.] ImportQuantumPixels
     3.90%  convert  /usr/lib/libMagickCore.so.2.0.1      [.] SetPixelCacheNexusPixels
     1.35%  convert  /usr/lib/libgomp.so.1.0.0            [.] 0x00000000009fb9
     1.29%  convert  /usr/lib/libjpeg.so.62.0.0           [.] jpeg_fdct_islow
     1.20%  convert  /usr/lib/libMagickCore.so.2.0.1      [.] CopyMagickMemory
     0.79%  convert  [kernel]                             [k] clear_page_c
     0.59%  convert  [kernel]                             [k] copy_user_generic_string
     0.55%  convert  /usr/lib/libMagickCore.so.2.0.1      [.] ReadPixelCachePixels
     0.33%         convert  [kernel]                      [k] do_page_fault
     0.22%         convert  [kernel]                      [k] usb_hcd_irq
     0.18%         convert  [kernel]                      [k] _spin_unlock_irqrestore
     0.14%         convert  [kernel]                      [k] __mem_cgroup_commit_charge
     0.13%         convert  [kernel]                      [k] get_page_from_freelist
     0.11%         convert  [kernel]                      [k] ____pagevec_lru_add
     0.10%         convert  /lib/libc-2.11.1.so           [.] 0x0000000008303b
     0.07%         convert  [kernel]                      [k] free_hot_cold_page
```

Das Tool perf ermöglicht es des Weiteren diejenigen Instruktionen anzuzeigen, welche auf Engpässe hin deuten und ggf. zu optimieren sind, mittels `perf annotat`. Laut Projekthomepage wird perf zukünftig weitere Performance Counter Features unterstützen.

4 Fazit

Collectd eignet sich für die graphische Darstellung von Engpässen (bzw. Auslastungen) und für das überwachen von Komponenten. Das Auslesen bzw. Nutzen von Performance bzw. CPU Counter wird nicht unterstützt. Für das Aufspüren von Engpässen eignet sich Collectd nicht. Es ist keiner Profiler wie Oprofile und perf es sind. Daher ist Collectd nur sehr bedingt für die Entwicklung bzw. Wartung von Softwareeinheiten nutzbar.

Oprofile ist ein vollwertiger Profiler, der bei der Entwicklung und Wartung von Softwarekomponenten genutzt werden kann. Neben dem CLI von Oprofile kann auch eine GUI verwendet werden, was die Benutzbarkeit erhöht. Oprofile unterstützt Performance Counter direkt und erzeugt nur einen minimalen Overhead, um das System zu überwachen. Nachteilig ist, dass Oprofile noch keinen 1.x Status erreicht hat.

Perf ist der offizielle Profiler für den GNU/Linux Kernel (>= 2.6.31). Ebenso wie Oprofile unterstützt perf Performance Counter direkt und kann bei der Entwicklung und Wartung von (verteilten) Anwendungen genutzt werden. Der Hauptvorteil von perf ist die Nähe zum Linux-Kernel und der Kernel-Entwicklergemeinde. Aus diesem Grunde kann angenommen werden, das perf sich in den nächsten Jahren als der Linux-Profiler etablieren wird. Nachteilig ist, dass perf noch in der ersten Entwicklungsphase ist. Es fehlt beispielsweise eine GUI mit der perf genutzt werden kann. Jedoch entstehen schon APIs für Programmiersprachen um perf zu nutzen[25].

25 http://lwn.net/Articles/373842/

A Anhang

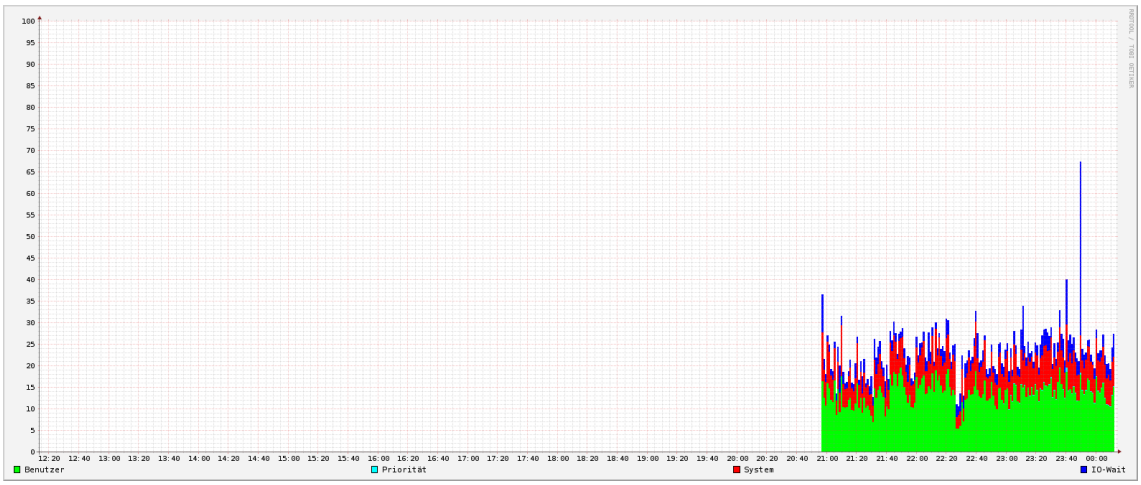
Collectd - collectd.conf (Default)

```
conf
FQDNLookup true
LoadPlugin logfile
<Plugin syslog>
       LogLevel info
</Plugin>
LoadPlugin battery
LoadPlugin cpu
LoadPlugin df
LoadPlugin disk
LoadPlugin entropy
LoadPlugin interface
LoadPlugin irq
LoadPlugin load
LoadPlugin memory
LoadPlugin processes
LoadPlugin rrdtool
LoadPlugin swap
LoadPlugin users
<Plugin rrdtool>
       DataDir "/var/lib/collectd/rrd"
</Plugin>
Include "/etc/collectd/filters.conf"
Include "/etc/collectd/thresholds.conf"
```

Collectd - collectd.conf (Seminar)

```
FQDNLookup true
LoadPlugin logfile
<Plugin syslog>
       LogLevel info
</Plugin>
LoadPlugin cpu
LoadPlugin cpufreq
<Plugin rrdtool>
       DataDir "/var/lib/collectd/rrd"
</Plugin>
```

Collectd – Graph



Oprofile - opreport -t 0.5 -l /usr/bin/convert

```
CPU: Core 2, speed 1801 MHz (estimated)
Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with a unit mask of
0x00 (Unhalted core cycles) count 500000
samples  %        image name               symbol name
50102    65.1606  libMagickCore.so.2.0.1   /usr/lib/libMagickCore.so.2.0.1
18820    24.4765  libc-2.11.1.so           /lib/libc-2.11.1.so
6272      8.1571  libjpeg.so.62.0.0        /usr/lib/libjpeg.so.62.0.0
1093      1.4215  libgomp.so.1.0.0         /usr/lib/libgomp.so.1.0.0
541       0.7036  jpeg.so                  /usr/lib/ImageMagick-6.5.7/modules-
Q16/coders/jpeg.so
```

Oprofile - opannotate -t 0.5 --source

```
01               :static inline Quantum RoundToQuantum(const MagickRealType value)
02               :{
03               :  if (value < 0.0)
04    912  0.0585 :   8d8b3:       ucomisd %xmm1,%xmm2
05    966  0.0620 :   8d8b7:       ja     8d8d0 <BlurImageChannel+0x660>
06               :    return(0);
07               :  if (value >= (MagickRealType) QuantumRange)
08   1009  0.0647 :   8d8b9:       mov    $0xffffffff,%eax
09               :   8d8be:       ucomisd %xmm5,%xmm1
10    949  0.0609 :   8d8c2:       jae    8d8d0 <BlurImageChannel+0x660>

11   1135  0.0728 :   8d8c4:     addsd  669028(%rip),%xmm1
```

collectd.conf – rrd2png.sh Skript

```
#!/bin/bash
DATA="/var/lib/collectd/rrd/localhost"
HEIGHT="600"
WIDTH="1500"
# CPU 0 / Tag
rrdtool graph /var/www/status/cpu-0-day.png --start -43200
-u 100 -r -l 0 -w $WIDTH -h $HEIGHT \
DEF:user=$DATA/cpu-0/cpu-user.rrd:value:AVERAGE
AREA:user#00ff00:"Benutzer" \
DEF:nice=$DATA/cpu-0/cpu-nice.rrd:value:AVERAGE
AREA:nice#00ffff:"Priorität":STACK \
DEF:system=$DATA/cpu-0/cpu-system.rrd:value:AVERAGE
AREA:system#ff0000:"System":STACK \
DEF:iowait=$DATA/cpu-0/cpu-wait.rrd:value:AVERAGE
AREA:iowait#0000ff:"IO-Wait":STACK > /dev/null
# CPU 1 / Tag
rrdtool graph /var/www/status/cpu-1-day.png --start -43200
-u 100 -r -l 0 -w $WIDTH -h $HEIGHT \
DEF:user=$DATA/cpu-1/cpu-user.rrd:value:AVERAGE
AREA:user#00ff00:"Benutzer" \
DEF:nice=$DATA/cpu-1/cpu-nice.rrd:value:AVERAGE
AREA:nice#00ffff:"Priorität":STACK \
DEF:system=$DATA/cpu-1/cpu-system.rrd:value:AVERAGE
AREA:system#ff0000:"System":STACK \
DEF:iowait=$DATA/cpu-1/cpu-wait.rrd:value:AVERAGE
AREA:iowait#0000ff:"IO-Wait":STACK > /dev/null
# Systemlast / Tag
rrdtool graph /var/www/status/load-day.png --start -43200 -w
$WIDTH -h $HEIGHT \
DEF:short=$DATA/load/load.rrd:shortterm:AVERAGE
AREA:short#ff0000:"5 Minuten" \
DEF:mid=$DATA/load/load.rrd:midterm:AVERAGE
AREA:mid#0000ff:"10 Minuten" \
DEF:long=$DATA/load/load.rrd:longterm:AVERAGE
AREA:long#00ff00:"15 Minuten" > /dev/null
```

Oprofile - opreport -t 5 /usr/bin/convert

```
CPU: Core 2, speed 1801 MHz (estimated)
Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with a unit mask of
0x00 (Unhalted core cycles) count 500000

CPU_CLK_UNHALT...|
  samples|      %|
------------------
    76890 100.000 convert
         CPU_CLK_UNHALT...|
           samples|      %|
         ------------------
             50102 65.1606 libMagickCore.so.2.0.1
             18820 24.4765 libc-2.11.1.so
              6272  8.1571 libjpeg.so.62.0.0
              1093  1.4215 libgomp.so.1.0.0
               541  0.7036 jpeg.so
                43  0.0559 libpthread-2.11.1.so
                13  0.0169 ld-2.11.1.so
                 4  0.0052 libtiff.so.4.3.2
                 2  0.0026 tiff.so
```

Oprofile - opannotate -t 5 --source --assembly

```
01 MagickExport Image *BlurImageChannel(const Image *image,
02                :  const ChannelType channel,const double radius,const double sigma,
03                :  ExceptionInfo *exception)
04                :{ /* BlurImageChannel total: 249586 33.3074 */
05
06                // ...
07
08  12844  1.7140 :      for (i=0; i < (long) width; i++)
09                :      {
10                :        alpha=1.0;
11   4237  0.5654 :        if (((channel & OpacityChannel) != 0) &&
12                :            (image->matte != MagickFalse))
13                :          {
14                :            alpha=((MagickRealType) QuantumRange-pixels[i].opacity)/
15                :              QuantumRange;
16    482  0.0643 :            pixel.opacity+=(*k)*pixels[i].opacity;
17                :          }
18   4345  0.5798 :        if ((channel & RedChannel) != 0)
19  21447  2.8621 :          pixel.red+=(*k)*alpha*pixels[i].red;
20   4473  0.5969 :        if ((channel & GreenChannel) != 0)
21  14589  1.9469 :          pixel.green+=(*k)*alpha*pixels[i].green;
22   4469  0.5964 :        if ((channel & BlueChannel) != 0)
23  17881  2.3862 :          pixel.blue+=(*k)*alpha*pixels[i].blue;
24   9616  1.2833 :        if (((channel & IndexChannel) != 0) &&
25                :            (image->colorspace == CMYKColorspace))
26                :          pixel.index+=(*k)*alpha*indexes[x+(pixels-p)+i];
27  10803  1.4417 :        gamma+=(*k)*alpha;
28                :        k++;
29                :      }
```

Performance Counter - opcontrol --list-events

CPU_CLK_UNHALTED	L1D_M_EVICT	BR_RET_EXEC
INST_RETIRED_ANY_P	L1D_PEND_MISS	BR_RET_MISSP_EXEC
L2_RQSTS	L1D_SPLIT	BR_RET_BAC_MISSP_EXEC
LLC_MISSES	SSE_PREF_MISS	BR_CALL_EXEC
LLC_REFS	LOAD_HIT_PRE	BR_CALL_MISSP_EXEC
LOAD_BLOCK	L1D_PREFETCH	BR_IND_CALL_EXEC
STORE_BLOCK	BUS_REQ_OUTSTANDING	BR_TKN_BUBBLE_1
MISALIGN_MEM_REF	BUS_BNR_DRV	BR_TKN_BUBBLE_2
SEGMENT_REG_LOADS	BUS_DRDY_CLOCKS	RS_UOPS_DISPATCHED
SSE_PRE_EXEC	BUS_LOCK_CLOCKS	RS_UOPS_DISPATCHED_NO
DTLB_MISSES	BUS_DATA_RCV	NE
MEMORY_DISAMBIGUATION	BUS_TRAN_BRD	MACRO_INSTS
PAGE_WALKS	BUS_TRAN_RFO	ESP
FLOPS	BUS_TRAN_WB	SIMD_UOPS_EXEC
FP_ASSIST	BUS_TRAN_IFETCH	SIMD_SAT_UOP_EXEC
MUL	BUS_TRAN_INVAL	SIMD_UOP_TYPE_EXEC
DIV	BUS_TRAN_PWR	INST_RETIRED
CYCLES_DIV_BUSY	BUS_TRANS_P	X87_OPS_RETIRED
IDLE_DURING_DIV	BUS_TRANS_IO	UOPS_RETIRED
DELAYED_BYPASS	BUS_TRANS_DEF	MACHINE_NUKES_SMC
L2_ADS	BUS_TRAN_BURST	BR_INST_RETIRED
L2_DBUS_BUSY_RD	BUS_TRAN_MEM	BR_MISS_PRED_RETIRED
L2_LINES_IN	BUS_TRAN_ANY	CYCLES_INT_MASKED
L2_M_LINES_IN	EXT_SNOOP	SIMD_INST_RETIRED
L2_LINES_OUT	CMP_SNOOP	HW_INT_RCV
L2_M_LINES_OUT	BUS_HIT_DRV	ITLB_MISS_RETIRED
L2_IFETCH	BUS_HITM_DRV	SIMD_COMP_INST_RETIRE
L2_LD	BUSQ_EMPTY	D
L2_ST	SNOOP_STALL_DRV	MEM_LOAD_RETIRED
L2_LOCK	BUS_IO_WAIT	FP_MMX_TRANS
L2_REJECT_BUSQ	L1I_READS	MMX_ASSIST
L2_NO_REQ	L1I_MISSES	SIMD_INSTR_RET
EIST_TRANS_ALL	ITLB	SIMD_SAT_INSTR_RET
THERMAL_TRIP	INST_QUEUE_FULL	RAT_STALLS
L1D_CACHE_LD	IFU_MEM_STALL	SEG_RENAME_STALLS
L1D_CACHE_ST	ILD_STALL	SEG_RENAMES
L1D_CACHE_LOCK	BR_INST_EXEC	RESOURCE_STALLS
L1D_CACHE_LOCK_DURATI	BR_MISSP_EXEC	BR_INST_DECODED
ON	BR_BAC_MISSP_EXEC	BR_BOGUS
L1D_ALL_REF	BR_CND_EXEC	BACLEARS
L1D_ALL_CACHE_REF	BR_CND_MISSP_EXEC	PREF_RQSTS_UP
L1D_REPL	BR_IND_EXEC	PREF_RQSTS_DN
L1D_M_REPL	BR_IND_MISSP_EXEC	

Performance Counter - perf --list

cpu-cycles OR cycles	block_rq_issue	power_start
instructions	block_bio_bounce	power_frequency
cache-references	block_bio_complete	power_end
cache-misses	block_bio_backmerge	module_load
branch-instructions	block_bio_frontmerge	module_free
OR branches	block_bio_queue	module_get
branch-misses	block_getrq	module_put
bus-cycles	block_sleeprq	module_request
cpu-clock	block_plug	workqueue_insertion
task-clock	block_unplug_timer	workqueue_execution
page-faults OR faults	block_unplug_io	workqueue_creation
minor-faults	block_split	workqueue_destruction
major-faults	block_remap	signal_generate
context-switches OR	block_rq_remap	signal_deliver
cs	jbd2_checkpoint	signal_overflow_fail
cpu-migrations OR	jbd2_start_commit	signal_lose_info
migrations	jbd2_commit_locking	timer_init
alignment-faults	jbd2_commit_flushing	timer_start
emulation-faults	jbd2_commit_logging	timer_expire_entry
L1-dcache-loads	jbd2_end_commit	timer_expire_exit
L1-dcache-load-misses	jbd2_submit_inode_dat	timer_cancel
L1-dcache-stores	a	hrtimer_init
L1-dcache-store-	jbd2_run_stats	hrtimer_start
misses	jbd2_checkpoint_stats	hrtimer_expire_entry
L1-dcache-prefetches	ext4_free_inode	hrtimer_expire_exit
L1-dcache-prefetch-	ext4_request_inode	hrtimer_cancel
misses	ext4_allocate_inode	itimer_state
L1-icache-loads	ext4_write_begin	itimer_expire
L1-icache-load-misses	ext4_da_write_begin	irq_handler_entry
L1-icache-prefetches	ext4_ordered_write_en	irq_handler_exit
L1-icache-prefetch-	d	softirq_entry
misses	ext4_writeback_write_	sched_process_fork
LLC-loads	end	sched_stat_wait
LLC-load-misses	ext4_journalled_write	sched_stat_sleep
LLC-stores	_end	sched_stat_iowait
LLC-store-misses	ext4_da_write_end	sched_stat_runtime
LLC-prefetches	ext4_writepage	mce_record
LLC-prefetch-misses	ext4_da_writepages	sys_enter
dTLB-loads	ext4_da_write_pages	sys_exit
dTLB-load-misses	kmalloc	
dTLB-stores	kmem_cache_alloc	
dTLB-store-misses	kmalloc_node	
dTLB-prefetches	kmem_cache_alloc_node	
dTLB-prefetch-misses	kfree	
iTLB-loads	kmem_cache_free	
iTLB-load-misses	mm_page_free_direct	
branch-loads	mm_pagevec_free	
branch-load-misses		

Literaturverzeichnis

[GZG01] Grelk, K.; Zilm, T.; Günther, K.: *Linux-Befehlsreferenz zur Systemadministration*, mitp-Verlag, Bonn, 2001.