Green Coding – die neue Dimension der Softwareentwicklung

Hans-Peter Keilhofer, Jakob Deiner

Der Klimawandel ist die größte Herausforderung, der die Menschheit zu Beginn des 21. Jahrhunderts ausgesetzt ist. Niemand zweifelt mehr daran, dass der Energieverbrauch global reduziert werden muss, um die CO_2-Emissionen zu senken und die Klimaveränderung zumindest abzubremsen. Welche Möglichkeiten hat die IT-Industrie, um Energie einzusparen und zu mehr Nachhaltigkeit beizutragen? Stichworte in diesem Zusammenhang sind grüne IT und Green Coding.

Unter **grüner IT** verstehen wir im Folgenden den Hardwarebetrieb mit grünem Strom oder allgemein geringem Energieverbrauch. **Green Coding** bezieht sich dagegen auf den Energieverbrauch der Software und hier insbesondere die Bereiche Softwarearchitektur, die konkreten Softwarekomponenten und die Betriebsplattform.

Wie können wir als Unternehmen selbst oder auch gemeinsam mit unseren Kunden zu einer grüneren Programmierung beitragen? Wir haben uns verschiedene Ansätze angesehen, die wir im Folgenden vorstellen.

Unternehmensarchitektur

Im Kontext von grüner IT und Green Coding besteht die Kernaufgabe der Unternehmensarchitektur darin, das Unternehmensziel „geringer ökologischer Fußabdruck“ durch geeignete Maßnahmen zu unterstützen.

Eine Maßnahme mit unmittelbarer Auswirkung ist das Forcieren von grüner IT. Dazu zählt das Etablieren grüner Rechenzentren sowie der Betrieb von Software in einer grünen Cloud, die jeweils gemäß den Prinzipien von grüner IT betrieben werden. Voraussetzung hierfür ist natürlich, dass die Software überhaupt in der Cloud betrieben werden kann. „Rechenzentren in der Cloud sind um bis zu 93 Prozent energie- und bis zu 98 Prozent CO_2-effizienter als firmeneigene Rechenzentren.“[1]

Falls dedizierte Rechenzentren unverzichtbar sind, sollte zumindest die entstehende Abwärme sinnvoll genutzt werden, zum Beispiel, um Firmengebäude zu beheizen. Der Strom sollte möglichst aus erneuerbaren Energien stammen, dies gilt gleichermaßen für die gesamte Unternehmens-IT. Durch intelligentes Scheduling des Rechenzentrums können komplexe Berechnungen bevorzugt zu Zeiten geplant werden, in denen viel Ökostrom zur Verfü-

gung steht. Wenn beispielsweise die firmeneigene Photovoltaikanlage an sonnigen Tagen zur Mittagszeit größere Mengen Solarstrom produziert, kann dieser optimal für teure Batchläufe genutzt werden.

Eine zweite wichtige Maßnahme ist, dass die Unternehmensarchitektur Green Coding in den Fokus der IT-Architektur und Entwicklung rückt. Green Coding muss als unternehmensweite, nichtfunktionale Anforderung mit hoher Priorität definiert werden. Daraus sollten konkrete nichtfunktionale Anforderungen sowie „Best Practices“ für IT-Architekten und Entwickler abgeleitet werden. Eine Auswahl an Richtlinien wird im Verlauf des Beitrags vorgestellt. Um ein gemeinsames Verständnis sicherzustellen, sollten darüber hinaus Schulungsprogramme für Architekten und Entwickler aufgesetzt werden. Non Functional Requirements (NFRs) und empfohlene Vorgehensweisen lassen sich am besten anhand verständlicher Praxisbeispiele verdeutlichen.

Wichtig ist, dass sich die Unternehmensarchitektur auf die firmeneigenen Softwareprodukte mit dem potenziell höchsten Energieverbrauch konzentriert. Dasselbe gilt für sehr große Projekte. Für kleinere bis mittelgroße Projekte obliegt die Green Coding Verantwortung der jeweiligen IT-Architektur.

So ist die Einhaltung der Green Coding NFRs bei einer Webanwendung mit mehreren tausend parallelen Nutzern viel kritischer als bei einer Desktop-Anwendung für zehn bis fünfzehn Nutzer.

Grünere IT-Architektur

Unterhalb der Unternehmensarchitekturebene kann die IT-Architektur auf System- oder Komponentenebene einen positiven Beitrag zum Umweltziel leisten.

Um weniger CO_2-Emissionen zu erzeugen, müssen zunächst die Komponenten mit dem höchsten Energiebedarf ermittelt werden. Anschließend können diese idealerweise optimiert werden. Die Optimierung kann zum Beispiel darin bestehen, dass Teilkomponenten bei Leerlauf heruntergefahren werden. Beim Betrieb eines Clusters mit Docker-Containern kann ein Knoten situationsabhängig hinzugefügt oder entfernt werden. Die Ermittlung des Energiebedarfs erfolgt dabei in der Regel ausschließlich über die Messung des Stromverbrauchs der Softwarekomponente. Im Abschnitt Performance Engineering gehen wir näher auf die verschiedenen Messverfahren ein.

Energieverbrauch beginnt bei den Anforderungen. Während der Anforderungsanalyse für ein Softwareprojekt muss daher kritisch hinterfragt werden, ob Anforderungen wie Echtzeitverarbeitung oder hoch dynamische Inhalte zwingend benötigt werden. Sind sie verzichtbar, können Lastspitzen vermieden und dadurch Energie gespart werden.

Natürlich muss ein vernünftiges Gleichgewicht zwischen der Forderung nach einer Minimierung des Energieverbrauchs und den funktionalen Anforderungen immer in Absprache mit dem Kunden gefunden werden.

Prinzipien und Entwicklungsmuster zur Verbrauchsminimierung

Im Folgenden stellen wir Green Coding Verfahren vor, die von Facharchitektur, IT-Architektur und Entwicklung gleichermaßen berücksichtigt werden sollten.

Bessere Algorithmen

Die wohl offensichtlichste Optimierung von Software ist die Verwendung des für eine bestimmte Aufgabe am besten geeigneten und leistungsfähigsten Algorithmus. Mit diesem Problem beschäftigen sich bereits ganze Generationen von Programmiererinnen und Programmierern. Die Wahl des Algorithmus hat wohl das größte Potenzial zur Verbesserung des Energieverbrauchs eines Programms.

Programmiererinnen und Programmierer sollten daher für jede Aufgabe prüfen, ob sich ein bekannter und bereits optimierter Algorithmus anwenden lässt, beziehungsweise ob sich die Aufgabe auf einen bekannten Algorithmus zurückführen lässt. Eigene Algorithmen zu entwickeln und zu optimieren ist sowohl in wirtschaftlicher als auch ökologischer Hinsicht der teuerste Weg und sollte daher vermieden werden. Diese Vorgehensweise empfiehlt sich im Übrigen unabhängig von den Green Coding Anforderungen.

Verminderung der Anzahl von Serveranfragen

Die Anzahl der Serveranfragen trägt wesentlich zum Energieverbrauch bei. Es muss daher für jede Anfrage abgewogen werden, ob sie wirklich notwendig ist.

So können zum Beispiel Anfragen zusammengefasst, Daten auf dem Client zwischengespeichert oder Caches für häufige gleichartige Requests eingerichtet werden. Durch das Nutzen von Caches werden unnötige Roundtrips über das Netzwerk und teure Festplatten- oder Datenbankzugriffe vermieden. Caches sind daher nicht nur ein Weg die Performance zu erhöhen, sondern auch den Energieverbrauch der Software zu senken.

Verminderung des Datenvolumens

Das Datenvolumen kann in der Regel durch Verwendung geeigneter schlanker Dateiformate und Komprimierung verkleinert werden. Ein Beispiel hierfür ist die Verwendung von JSON statt XML für Web-Services, da dieses Format wesentlich weniger Overhead produziert.

Wenn ein Zugriff über das Netzwerk erfolgen muss, so sollten die Daten komprimiert übertragen werden. Das spart Bandbreite und vermindert die Kosten der Übertragung. Jeder Router, der an einer Netzwerkübertragung beteiligt ist, benötigt Strom. Je mehr Daten also übertragen werden, desto länger wird der Strom auch verbraucht. Das Komprimieren der Daten ist daher eine einfache Möglichkeit, um Energie zu sparen. Für Medien verringern verlustbehaftete Formate wie JPEG, AAC und AV1 die benötigte Bandbreite um ein Vielfaches.

Datenbankindizes

Indizes auf Tabellen gehören zum Standardrepertoire eines IT-Architekten. Trotzdem findet man oft Software, die ohne Index auf die Daten zugreift. Mit wachsender Datenmenge verschärft sich das Problem – die Zugriffe werden langsam. Dadurch steigt die CPU- und IO-Last auf der Datenbank, was zu einem hohen Energieverbrauch führt.

Zero-Waste-Code

Während in den Anfängen der Softwareentwicklung für jedes Problem eine eigene Lösung erstellt werden musste, sind heute unzählige Open-Source-Bibliotheken für fast jeden Zweck frei verfügbar.

Grundsätzlich ist das eine sehr begrüßenswerte Entwicklung (vgl. „Proudly found elsewhere" statt „Not invented here"). Das Einbeziehen externer Abhängigkeiten erhöht jedoch auch das Risiko, eine Menge überflüssigen Code auszuliefern.

Insbesondere im Kontext von Webanwendungen, die bei jeder neuen Benutzersitzung den Download des Codes einschließlich der Abhängigkeiten erfordern, wird durch weitgehend ungenutzte Bibliotheken potenziell viel Bandbreite und CPU-Parsing-Zeit verschwendet.

Eine gute Möglichkeit, „toten" Code sicher zu entfernen, ist der Einsatz von Tree-Shaking-Engines. Im JavaScript-Umfeld sind „module bundlers" wie webpack oder Rollup weit verbreitet. Sie entfernen beim Zusammenfassen mehrerer JavaScript-Dateien in eine Datei automatisch nicht verwendeten Code. Für Java und Kotlin existiert mit „ProGuard" ebenfalls ein bekanntes Tool.

Performance Engineering

„Das ‚Performance-Engineering' ist ein ganzheitlicher Ansatz, der Leistungsanforderungen von Beginn an als integralen Bestandteil der Entwicklung eines neuen Produktes sieht."[2] Performance Engineering etabliert dazu Techniken, um sicherzustellen, dass die nicht-funktionalen Anforderungen (wie Durchsatz, Latenzzeit oder Speichernutzung) erfüllt werden.[3]

Derartige Techniken gilt es auch für die nicht-funktionale Anforderung „geringer Energieverbrauch" zu etablieren. Zu diesem Zweck muss der Energieverbrauch während der Entwicklung, aber auch später im Betrieb, gemessen und gegebenenfalls Maßnahmen ergriffen werden. Voraussetzung dafür ist wiederum, dass die Anforderung mit einer messbaren Bedingung formuliert wird. Beispielsweise könnte eine nicht-funktionale Anforderung lauten: „Komponente X in System Y soll auf einem Intel Prozessor mit 2 Cores nicht mehr als 0.1 kWh Strom verbrauchen, wenn die Berechnung Z ausgeführt wird."

Energieverbrauch wird im Folgenden vereinfacht mit „Stromverbrauch" gleichgesetzt. Grundsätzlich fallen auch bei der Herstellung und bei der Entsorgung von Hardware CO2-Kosten an. Diese Kosten betrachten wir hier nicht näher, da sie im Unternehmenskontext beim Einkauf beziehungsweise der Materialbeschaffung berücksichtigt werden sollten.

Wie kommt man nun zu der Kenngröße „Energieverbrauch" für eine zu lösende Programmieraufgabe?

Zunächst ist festzulegen, wie der Stromverbrauch gemessen wird. Grundsätzlich muss man unterscheiden, ob absolut oder relativ gemessen wird. Relative Messungen messen die Zunahme des Stroms in Prozent und sind in der Regel einfacher anzustellen und allgemeiner gültig. Absolute Messungen sind abhängig von der Zielplattform.

Entweder misst man den Stromverbrauch des gesamten Computers mittels eines Smart Plug[4] an der Steckdose. Diese Messmethode ist jedoch sehr ungenau und man muss die Differenz des Stromverbrauchs ohne und mit laufendem Programm berechnen. Alternativ kann indirekt über die Anzahl der benötigten CPU-Zyklen für eine Operation oder für ein ausgeführtes Programm der Verbrauch hochgerechnet werden. Die dritte Möglichkeit ist, den Stromverbrauch der CPU über hardwarenahe Software auszulesen. Dies ist in der Regel die genaueste Messmethode, da bei modernen Hardwarebausteinen die Informationen zum gerade verbrauchten Strom über eine Schnittstelle ausgelesen werden können.

Beispielsweise kann das Tool „Open Hardware Monitor“[5] den Stromverbrauch auslesen.

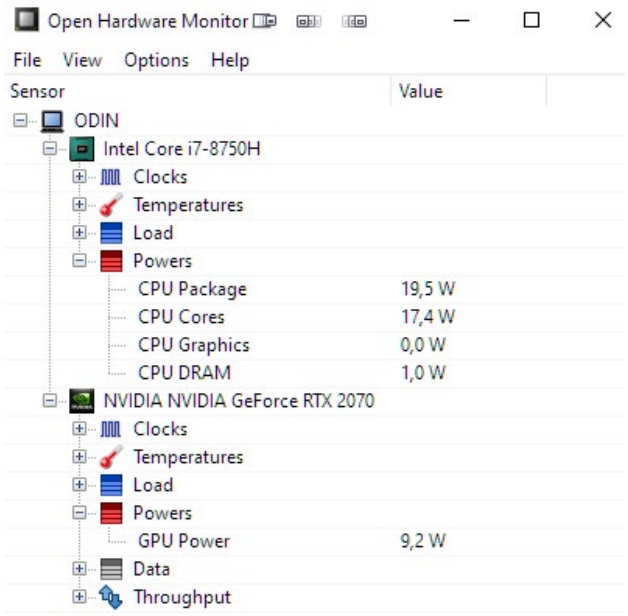


Abbildung 1: Beispielansicht Open Hardware Monitor

Neben der Messmethode – also dem **wie** – gibt es auch mehrere Ansätze, wann im Softwareentwicklungszyklus der Stromverbrauch gemessen werden soll. So ist es denkbar, ein Plugin für Unittests zu schreiben, das neben den funktionalen Prüfungen auch den Stromverbrauch misst und mit einer Sollgröße abgleicht. Wie oben erwähnt, könnte man die Messung über die verbrauchten CPU-Zyklen durchführen und dann auf die Plattform hochrechnen.

In Integrationstests kann der Verbrauch einer Komponente gemessen werden.

Ein Monitoring der Gesamtapplikation kann auf der Zielplattform im laufenden Betrieb erfolgen.

Um die NFR „geringer Stromverbrauch“ einzuhalten, wird idealerweise in allen Phasen der Entwicklung gemessen. Je früher eine Komponente auffällt, desto besser kann man auf negative Entwicklungen reagieren. Wie bei allen nicht-funktionalen Anforderungen muss auch der Energieverbrauch an die Gegebenheiten des Kunden, wie zum Beispiel Betriebssystem und Zielplattform, angepasst werden.

Es gibt bereits verschiedene Tools für unixbasierte Systeme, wie zum Beispiel PowerTOP, das neben der CPU-Usage auch den Stromverbrauch in Watt ausgibt, siehe Abbildung 2. Frameworks, zum Beispiel JoularJX[6], Jalen[7] und weitere[8] bieten darüber hinaus die Möglichkeit, den Stromverbrauch aus dem Programm heraus zu lesen.

Synergieeffekte

Wenn Programmiererinnen und Programmierer sich mit den oben beschriebenen Tools und Messmethoden auseinandersetzen, dann wird nicht nur der Energieverbrauch der Software sinken, sondern in der Regel auch die Benutzbarkeit und Performance des Programms in vielen Punkten positiv beeinflusst.

Ressourcenverbrauch

Über den gesamten Softwareentwicklungszyklus, das heißt vom ersten Inkrement bis zum Go-Live, sollte stets der Ressourcenverbrauch im Auge behalten sowie im Sinne von hoher Energieeffizienz möglichst niedrig gehalten werden.

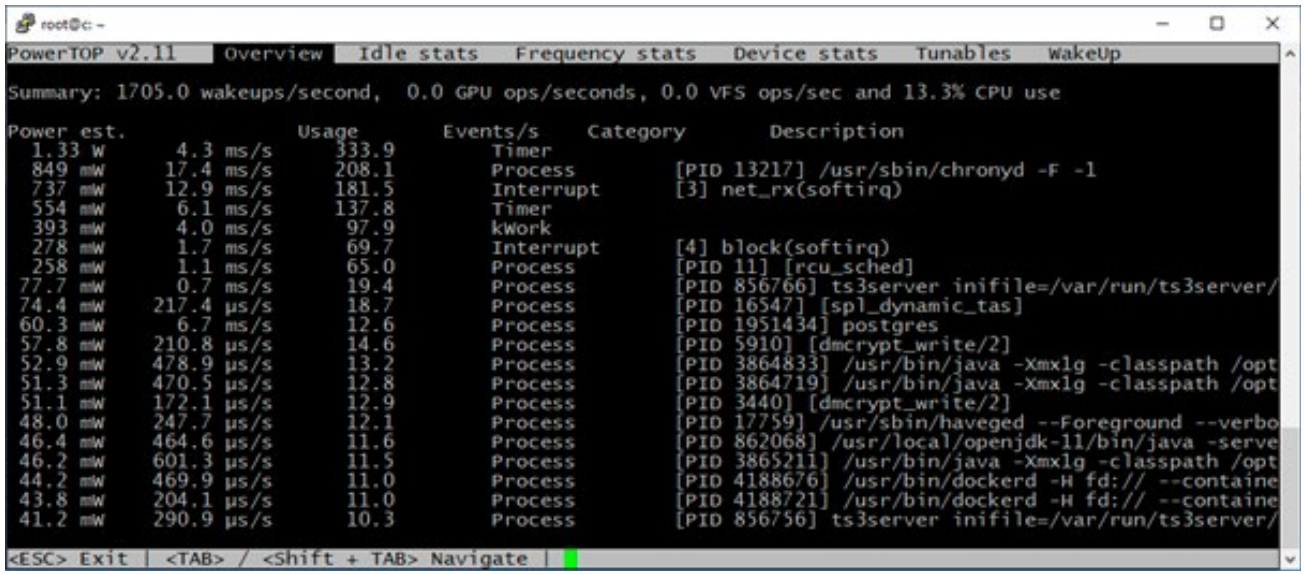


Abbildung 2: PowerTOP Tool

Der Ressourcenverbrauch einer Applikation beinhaltet sowohl die CPU-Zeit als auch die CPU-Auslastung, den Arbeitsspeicherbedarf, Datenträgerspeicherplatz und Datenträgerauslastung sowie die Netzwerklast. Für die meisten Business-Anwendungen wird der GPU-Verbrauch eher zweitrangig sein. Aber zum Beispiel für Desktop-Anwendungen mit aufwendig gestalteter UI oder Web-Anwendungen mit vielen Medien- oder 3D-Inhalten sollte er mitberücksichtigt werden.

Einen ersten Anhaltspunkt zur Messung des Ressourcenverbrauchs auf Prozessebene liefert der in Windows enthaltene Task-Manager, siehe Abbildung 3.

Seit Windows 10 Version 1803 zeigt der Task-Manager zusätzlich den Stromverbrauch eines Prozesses an, wenngleich nur als grobe Orientierung. Weitere Details liefert der Ressourcenmonitor, der über einen Link im Tab „Leistung“ des Task-Managers aufgerufen werden kann.

In MacOS ist die Aktivitätsanzeige integriert, für Unix-basierte Betriebssysteme erfüllen die Kommandozeilentools top und dessen Erweiterung htop ähnliche Zwecke.

Zur Analyse des Ressourcenverbrauchs innerhalb der Applikation, sprich einzelner Komponenten, Objekte oder Methoden, bieten sich Profiler an. Viele bekannte IDEs wie IntelliJ, eclipse, Visual Studio oder VS Code bringen bereits Profiler für die geläufigsten Programmiersprachen Java, C/C++, C# und weitere mit. Für TypeScript und JavaScript enthalten die Entwicklertools der verbreitetsten Browser Chrome, Edge und Firefox entsprechende Werkzeuge. Falls für die Programmiersprache der Wahl kein Profiler verfügbar ist, kann zur Not auf trace-Logging mit Zeitstempel und den wichtigsten Parametern wie CPU-Last, RAM-Verbrauch etc. ausgewichen werden.

Lasttest-Frameworks wie Java Microbench Harness und JMeter helfen, den Ressourcenverbrauch der Software unter Stress einzuschätzen. Durch Lasttests lassen sich Speicherlecks oder Flaschenhals-Komponenten und -Funktionen mit zu hohem CPU-Verbrauch im Regelfall schnell ermitteln.

Relationale Datenbankmanagementsysteme (RDBMS) werden mit einer Anfragesprache wie SQL angesteuert. Dabei wird vom Entwickler im SQL-Statement spezifiziert, was als Ergebnismenge geliefert werden soll – jedoch nicht, wie der Zugriff auf die Daten zur Laufzeit erfolgt. Für diesen Zweck enthält das DBMS die Komponente „Anfrageoptimierer“, die einen möglichst effizienten Ausführungsplan für das eingegebene SQL zu erzeugen versucht. Einschränkend muss man hinzufügen, dass dies leider nicht immer funktioniert, respektive das SQL selbst oftmals noch Optimierungspotenzial hat.

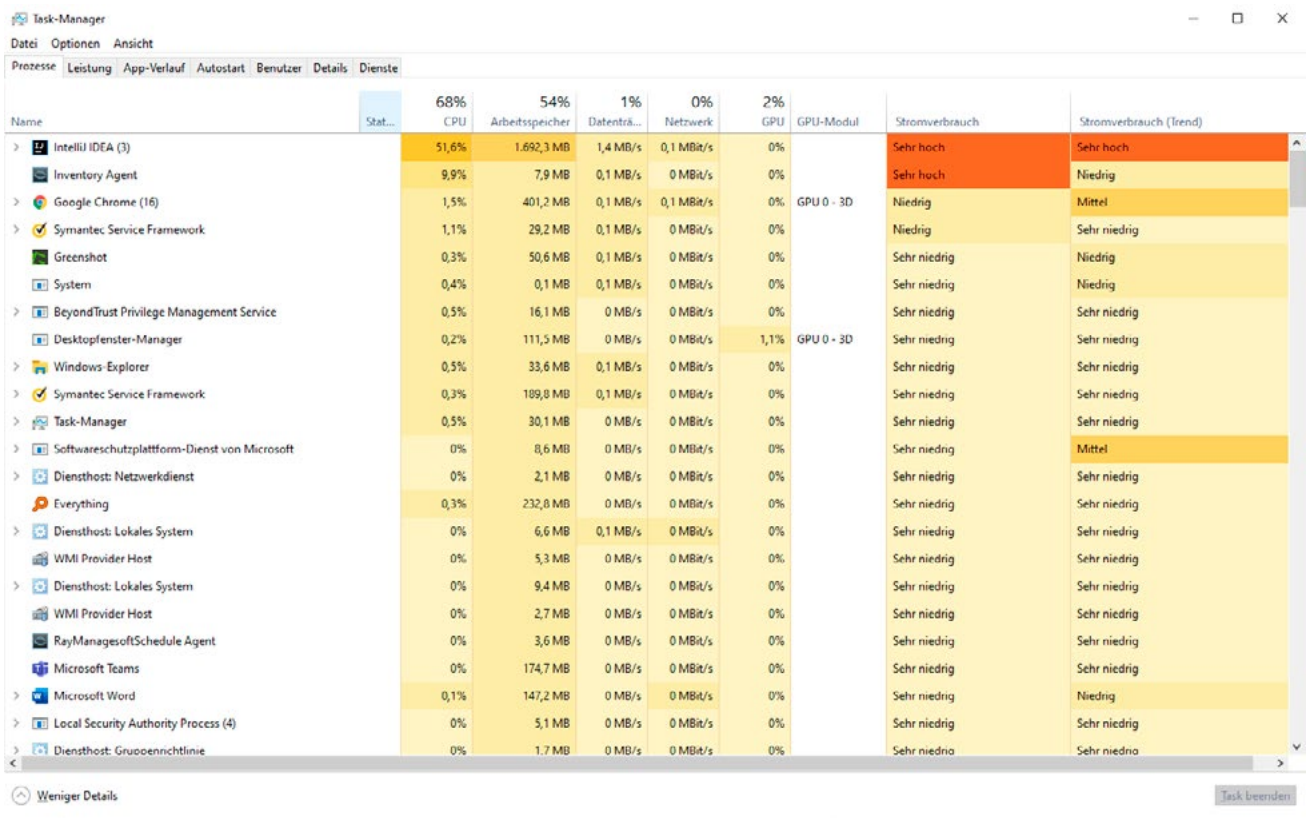
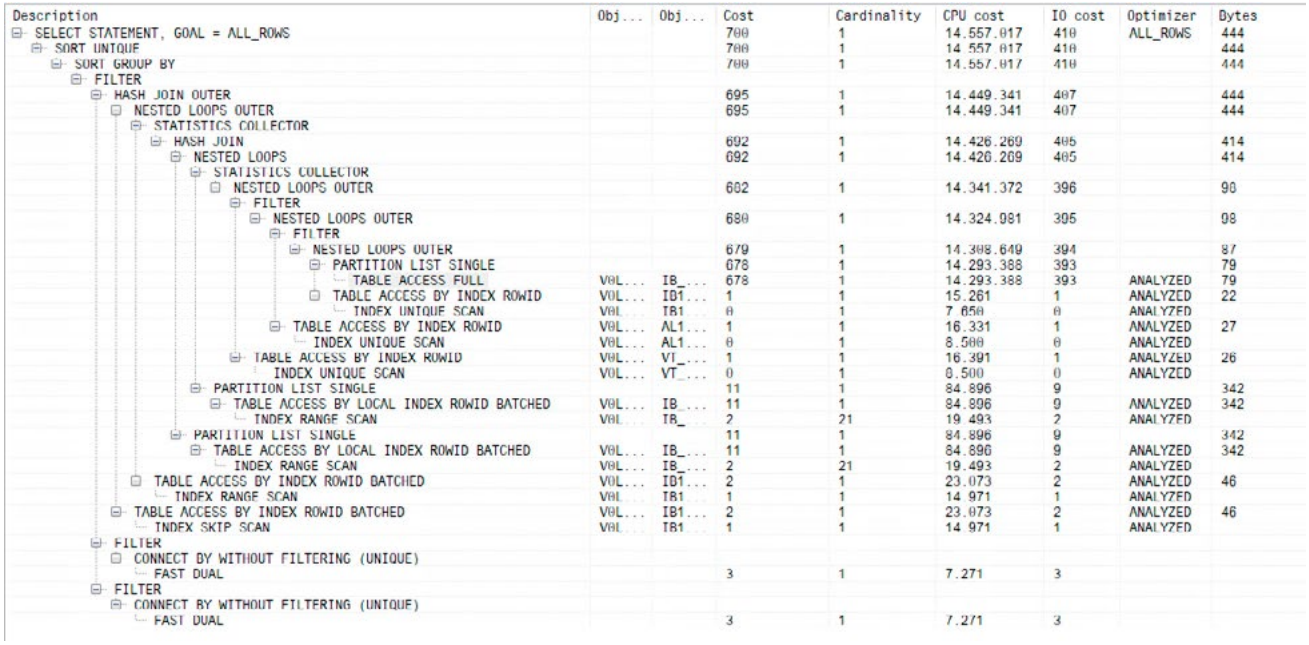


Abbildung 3: Windows Task-Manager



Description	Obj...	Obj...	Cost	Cardinality	CPU cost	IO cost	Optimizer	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			700	1	14.557.017	410	ALL_ROWS	444
SORT UNIQUE			700	1	14.557.017	410		444
SORT GROUP BY			700	1	14.557.017	410		444
FILTER								
HASH JOIN OUTER			695	1	14.449.341	407		444
NESTED LOOPS OUTER			695	1	14.449.341	407		444
STATISTICS COLLECTOR								
HASH JOIN			692	1	14.426.269	405		414
NESTED LOOPS			692	1	14.426.269	405		414
STATISTICS COLLECTOR								
NESTED LOOPS OUTER			602	1	14.341.372	396		90
FILTER								
NESTED LOOPS OUTER			680	1	14.324.981	395		98
FILTER								
NESTED LOOPS OUTER			679	1	14.308.649	394		87
PARTITION LIST SINGLE			678	1	14.293.388	393		79
TABLE ACCESS FULL	V0L...	IB_...	678	1	14.293.388	393	ANALYZED	79
TABLE ACCESS BY INDEX ROWID	V0L...	ID1...	1	1	15.261	1	ANALYZED	22
INDEX UNIQUE SCAN	V0L...	IB1...	0	1	7.650	0	ANALYZED	
TABLE ACCESS BY INDEX ROWID	V0L...	AL1...	1	1	16.331	1	ANALYZED	27
INDEX UNIQUE SCAN	V0L...	AL1...	0	1	8.500	0	ANALYZED	
TABLE ACCESS BY INDEX ROWID	V0L...	VI_...	1	1	16.391	1	ANALYZED	26
INDEX UNIQUE SCAN	V0L...	VT_...	0	1	8.500	0	ANALYZED	
PARTITION LIST SINGLE			11	1	84.896	9		342
TABLE ACCESS BY LOCAL INDEX ROWID BATCHED	V0L...	IB_...	11	1	84.896	9	ANALYZED	342
INDEX RANGE SCAN	V0L...	IB_...	2	21	19.493	2	ANALYZED	
PARTITION LIST SINGLE			11	1	84.896	9		342
TABLE ACCESS BY LOCAL INDEX ROWID BATCHED	V0L...	IB_...	11	1	84.896	9	ANALYZED	342
INDEX RANGE SCAN	V0L...	IB_...	2	21	19.493	2	ANALYZED	
TABLE ACCESS BY INDEX ROWID BATCHED	V0L...	ID1...	2	1	23.073	2	ANALYZED	46
INDEX RANGE SCAN	V0L...	IB1...	1	1	14.971	1	ANALYZED	
TABLE ACCESS BY INDEX ROWID BATCHED	V0L...	IB1...	2	1	23.073	2	ANALYZED	46
INDEX SKIP SCAN	V0L...	IB1...	1	1	14.971	1	ANALYZED	
FILTER								
CONNECT BY WITHOUT FILTERING (UNIQUE)								
FAST DUAL			3	1	7.271	3		
FILTER								
CONNECT BY WITHOUT FILTERING (UNIQUE)								
FAST DUAL			3	1	7.271	3		

Abbildung 4: PL/SQL Developer „Explain Plan“

Die meisten RDBMS bieten ein Explain Plan Tool, mit dem sich die Datenbankentwickler den Ausführungsplan anzeigen lassen und ihre Statements hinsichtlich geringer Kosten tunen können.

DBMS-IDEs wie SQL Developer und PL/SQL Developer bieten eine bequeme Möglichkeit, den Ausführungsplan aufzurufen. Ohne die Detailinterpretation verstehen zu müssen, bekommen die Entwickler bei jeder Anpassung sofort Feedback, ob sich die Gesamtkosten, CPU-Kosten sowie IO-Kosten und Speicherverbrauch dadurch positiv oder negativ entwickeln.

DevOps und Betrieb

Moderne CI/CD-Pipelines, realisiert durch Build-Server wie Jenkins oder Bamboo, sind einerseits ein Segen bezüglich Qualität. In vielen Konfigurationen laufen zum Beispiel automatisiert alle Tests, bevor ein neues Feature oder ein Bugfix in den Hauptentwicklungszweig des Quellcodeverwaltungssystems integriert werden darf. Andererseits erhöht dieses Pattern unzweifelhaft den Energieverbrauch während der Entwicklung beträchtlich – man stelle sich vor, bei jeder noch so kleinen Änderung laufen 1000 oder mehr Tests. Ein Prozess, der mehrere Minuten bis Stunden dauern kann. Aus Green Coding Perspektive sollte man sich daher durchaus die Frage stellen, wie oft Unittests und Integrationstest wirklich laufen müssen? Tatsächlich nach jedem einzelnen Check-in? Oder ist es gegebenenfalls völlig ausreichend, sie einmal pro Tag, bevorzugt zu einem Zeitpunkt mit viel verfügbarer grüner Energie, laufen zu lassen?

Als Mitte der 2000er Jahre virtuelle Maschinen durch Open-Source-Lösungen wie VirtualBox einen Aufschwung erlebten, wurden diese von der Entwicklergemeinde dankend adaptiert. Ein Nachteil von VMs ist die „Schwergewichtigkeit“ und der damit einhergehende hohe Ressourcen- und Energieverbrauch. Eine VM muss ein gesamtes Betriebssystem samt Kernel, Neben- und Hintergrundprozessen auf dem Host emulieren. Im Jahr 2013 kam mit Docker für viele Anwendungsfälle eine elegantere und leichtgewichtigere Lösung. Container nutzen im Gegensatz zu VMs den Kernel und viele OS-Funktionen des Host-Systems mit. Dadurch erzeugen Container-Instanzen viel weniger Overhead als VMs und haben somit per se einen geringeren Energieverbrauch.

„Mit Hilfe der Container-Orchestrierung werden Deployment, Ressourcenzuweisung, Skalierung und Vernetzung von Containern automatisiert.“[9] Container-Orchestrierungsplattformen wie das quelloffene Kubernetes oder das darauf basierende kommerzielle Openshift bieten diverse Mechanismen für optimierte Ressourcennutzung durch optimierte Konfiguration der Deployment-Rezepte.

Optimale Nutzung der zur Verfügung stehenden Hardware kann unter anderem dadurch erreicht werden, dass jedem einzelnen Pod fix Ressourcen zugewiesen werden (CPU/RAM/Disk space). Pod bezeichnet dabei eine Containerinstanz.

Werden die festgelegten Grenzen über- oder unterschritten, skalieren Orchestrierungsplattformen die Anzahl

der Pods automatisch in beide Richtungen (horizontale Skalierung).[10] Durch die Definition sowohl minimal zugesicherter als auch maximal erlaubter Ressourcen ist die automatische Verteilung und gegebenenfalls Verschiebung von Pods auf ausreichend leistungsfähige Cluster-Nodes umsetzbar (vertikale Skalierung).[11] Für die Ersteller der Deployment-Rezepte gilt es dabei zu beachten, dass die Hardware-Limits nicht überschritten werden, um unerwünschten Nebeneffekten vorzubeugen.[12]

„Der Schlüssel zur effizienten Ressourcenauslastung ist die Minimierung von Leerlauf. Ein Server, der ungenutzte Kapazitäten hat, ist totes Kapital“[13] beziehungsweise vermeidbarer Energieverbrauch.

JUnit Extension

Um den Energieverbrauch in Tests messen zu können, haben wir haben eine JUnit-Extension erstellt. Die Erweiterung nutzt die kostenlose Open-Source-Software „Open Hardware Monitor“ zum Auslesen der Stromverbrauchswerte. Es kann konfiguriert werden, ob in einer Initialisierungsphase die Grundlast gemessen und später vom Verbrauch abgezogen wird. Oder alternativ, ob die Grundlast fix vorgegeben wird. Auf jedem PC laufen in der Regel verschiedenste Hintergrundprozesse, die den Stromverbrauch auch während des Tests beeinflussen. Um beispielsweise zwei Algorithmen in ihrer Energieeffizienz zu vergleichen, sollten daher möglichst alle anderen Programme geschlossen, die Tests mehrfach wiederholt und dann die Mittelwerte der Messungen verglichen werden. Das Tool erhebt nicht den Anspruch, absolute Stromverbrauchswerte zu liefern, bietet aber die Möglichkeit, verschiedene Implementierungen bezüglich des Stromverbrauchs zu vergleichen.

Fazit

Alle, die am Entwicklungsprozess und am Betrieb einer Software beteiligt sind, haben das Potenzial, Strom und damit Energie zu sparen. In diesem Beitrag haben wir gezeigt, wie die nicht-funktionale Anforderung „niedriger Energieverbrauch“ in Fachbereichen, IT-Architektur, Programmierung, Test und Betrieb berücksichtigt werden kann.

Die vorgestellten Aspekte können anhand der dargestellten Checkliste wieder ins Gedächtnis gerufen und in jeder Phase eines Softwareprojekts angewendet werden. In der Regel ergeben sich positive Nebeneffekte auf andere NFRs wie Performance oder Benutzbarkeit der Software.

Allein durch Messungen, die zum Beispiel mit unserer JUnit-Extension durchgeführt werden können, entsteht ein Gefühl für die Größe des Energieverbrauchs in Softwarekomponenten. So können sinnvolle Anforderungen an neue oder zu ändernde Komponenten gestellt werden.

Ansprechpartner

Hans-Peter Keilhofer
Executive IT Consultant

Hans-Peter.Keilhofer@msg.group

Jakob Deiner
Lead IT Consultant

Jakob.Deiner@msg.group

1. Weber, Melanie, in: IDC (2021): Nachhaltigkeit in deutschen Industrieunternehmen 2021, S. 22, URL: https://info.microsoft.com/rs/157-GQE-382/images/de-ebook-SRGCM5002.pdf?mkt_tok=MTU3LUdRRS0zODIAAAGAo6eTYVcKOl5slrMo75SWo-I1xcOVA-UPecEClzi_d5Ar2CTcgVUgCn47gkaWl75lYu7GCAw9-XdywVMDcT9alUMQBrSRheL_G1xkUpogaJeSVqRDmmbGzJQJ (Stand: 15.02.2022).
2. Universität Bonn (2019): Performance Engineering, URL: https://net.cs.uni-bonn.de/de/wg/cs/projekte/performance-engineering/ (Stand: 15.02.2022).
3. Vgl. Wikipedia (2021): Performance engineering, URL: https://en.wikipedia.org/wiki/Performance_engineering (Stand: 15.02.2022).
4. Producto GmbH (2021): Steckdosen & Zubehör: Tests & Meinungen, URL: https://www.testberichte.de/f/1/2972/470861.470864/1.html (Stand: 15.02.2022).
5. Möller, Michael (2021): Open Hardware Monitor, URL: https://openhardwaremonitor.org/ (Stand: 15.02.2022).
6. Noureddine, Adel (2021): JoularJX, URL: https://www.noureddine.org/research/joular/joularjx (Stand: 15.02.2022).
7. Noureddine, Adel (2013): Measure the energy consumption of Java applications with Jalen, URL: https://www.noureddine.org/articles/measure-the-energy-consumption-of-java-applications-with-jalen (Stand: 15.02.2022).
8. Cruz, Luís (2021): Tools to Measure Software Energy Consumption from your Computer, URL: https://luiscruz.github.io/2021/07/20/measuring-energy.html (Stand: 15.02.2022).
9. Red Hat (2019): Was ist Container-Orchestrierung?, URL: https://www.redhat.com/de/topics/containers/what-is-container-orchestration (stand: 15.02.2022).
10. Vgl. Kubernetes (2020): Horizontal Pod Autoscaler, URL: https://kubernetes.io/de/docs/tasks/run-application/horizontal-pod-autoscale/ (Stand: 15.02.2022).
11. Vgl. Kubernetes (2022): Managing Resources for Containers, URL: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ (Stand: 15.02.2022).
12. Vgl. Credativ GmbH (2020): Verwaltung von Hardware-Ressourcen in Kubernetes, URL: https://www.credativ.de/blog/howtos/verwaltung-von-hardware-ressourcen-in-kubernetes/ (Stand: 15.02.2022).
13. Pustina, Lukas/ Schneller, Daniel (2015), in heise online: Operations heute und morgen, Teil 3: Virtualisierung und Containerisierung, URL: https://www.heise.de/ratgeber/Operations-heute-und-morgen-Teil-3-Virtualisierung-und-Containerisierung-2762412.html?seite=all (Stand: 15.02.2022).

Checkliste
Die wichtigsten Punkte für alle Stakeholder

- ☑ Nicht funktionale Anforderungen zu grüner IT und Green Coding sind definiert
- ☑ IT-Architekten und Entwickler sind geschult
- ☑ System skaliert dynamisch und fährt sich bei Leerlauf herunter
- ☑ Lastspitzen werden vermieden
  - Echtzeitverarbeitung wird vermieden, wo möglich
  - Dynamische Inhalte halten sich in vertretbarem Rahmen
- ☑ Komponenten mit dem höchsten Energieverbrauch sind identifiziert
  - Schätzung des Stromverbrauchs ist erfolgt
  - Proof-of-Concept Messungen wurden durchgeführt
- ☑ Green Coding Prinzipien und Entwicklungsmuster in folgenden Punkten berücksichtigt:
  - Effizienter Code mit optimalen Algorithmen
  - Vermeidung unnötige Round-Trips
  - Caching
  - Optimiertes Datenvolumen
  - Komprimierte Netzwerkkommunikation
  - Datenbankindizes
  - Optimierte Ressourcennutzung durch optimierte Konfiguration
  - Zero-Waste-Code
  - Prinzip: „Jede Codezeile hat das Potential den Energieverbrauch zu senken“
- ☑ Performance Engineering
  - Es ist klar, was, wann und wie gemessen wird
    - Was: Gesamtsystem/Prozess/Komponente/Methode
    - Wann: Unittest, Integrationstest, Betrieb (ideal: in jeder Phase)
    - Wie: Task-Manager, Open Hardware Monitor oder Smart Plugs, Frameworks, z. B. JoularJX, Jalen und weitere
- ☑ Ressourcenverbrauch
  - CPU/RAM/Speicher/Netzwerk
  - Optimieren
    - Profiler
    - SQL Explain Plan
- ☑ DevOps und Betrieb
  - DevOps Pipeline ist hinsichtlich Energieverbrauch optimiert
  - Containerbasierte Plattform mit intelligenter, dyn. Skalierung wird genutzt
  - Ressourcen werden optimal genutzt (CPU, RAM, Speicher, Netzwerk, GPU, ...)
- ☑ Konkrete Green Coding NFRs sind berücksichtigt