DCDB & EAR

Steven Pan
Seminar „Green Computing“
Universität Potsdam
Wintersemester 2021/22

1 DCDB

Data Center Data Base (DCDB) ist ein Open-Source HPC-Monitoring-System, welches im großen Stil Metriken verschiedener Quellen sammelt und in einer Datenbank speichert [18]. Aktuell läuft es mitunter auf dem Flaggschiff-System des Leibniz-Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften SuperMUC-NG [21]. Der Source-Code ist zum Zeitpunkt dieser Arbeit im Repository frei zugänglich[1].

1.1 Motivation

DCDB ist eine Eigenentwicklung des LRZ. Das Rechenzentrum hatte zum Zeitpunkt der Konzipierung eine Vielzahl von uneinheitlichen Monitoring-Systemen und Datenquellen, darunter [21]:

- Kühlungsinfrastruktur - Data Center Systeme
- Stromversorgung - lokale und Globale Ressource Management Systeme
- Gebäudesysteme - Facility-Monitoring
- HPC-Monitoring - sowohl Hardware- als auch Software-Monitoring

Neben den fehlenden Schnittstellen der vorhandenen Monitoring-Systeme, es waren oftmals Excel-Exports, war den Autoren von DCDB das Monitoring ihrer Systeme nicht granular genug.[21]

Somit sollte DCDB als neues holistisches Monitoring-System entstehen, welches zunächst nur Hardware- und später auch Softwaremetriken sammeln sollte. Ein Verwendungszweck der gigantischen Mengen von erhobenen Daten sollte sich später ergeben.[21] Als Anforderungen an DCDB nennen die Autoren vor allem folgende Punkte.

[1] `https://gitlab.lrz.de/dcdb/dcdb`

- **Verteilbarkeit, Skalierbarkeit**: Das System SuperMUC-NG hat 311.040 Cores verteilt auf 6.480 Nodes [16]. Ein Monitoring-System muss die gewaltigen Mengen von erhobenen Sensorwerten in Systemen dieser Größenordnung bewältigen können [18].
- **Heterogene Datenquellen**: Die bereits oben genannten Datenquellen sollen in das System integriert werden können und vergleichbare Werte einspeisen [18].
- **Effizienz**: Um auf den Compute-Nodes In-Band-Metriken zu erheben und die laufenden Applikationen dabei möglichst minimal zu beeinflussen, d.h. niedriger Overhead, müssen die verantwortlichen Komponenten von DCDB eine geringe Ressourcenbelastung aufweisen. Dies betrifft sowohl die Netzwerkressourcen beim Zusammenführen der Daten als auch die Compute-Ressourcen beim Erheben bzw. Senden der Daten auf den Nodes selber.[18]
- **Modulare Erweiterbarkeit**: DCDB soll als Open-Source Tool an das Zielsystem angepasst werden können. Dabei betonen die Autoren vor allem die Möglichkeiten zusätzliche oder auch neuartige Datenquellen während des Betriebs in das System zu integrieren oder alle Komponenten auszutauschen, solange sie die für DCDB nötigen Interfaces haben.[18].

1.2 Komponenten

DCDB basiert auf einer hierarchischen Architektur mit drei Hauptkomponenten, welche über ein Kommunikationsprotokoll verbunden sind und in Abb. 1 dargestellt sind: sog. Pushern zur Datenerhebung, Collect Agents zur Datenzusammenführung und Storage Backends zur Datenspeicherung.

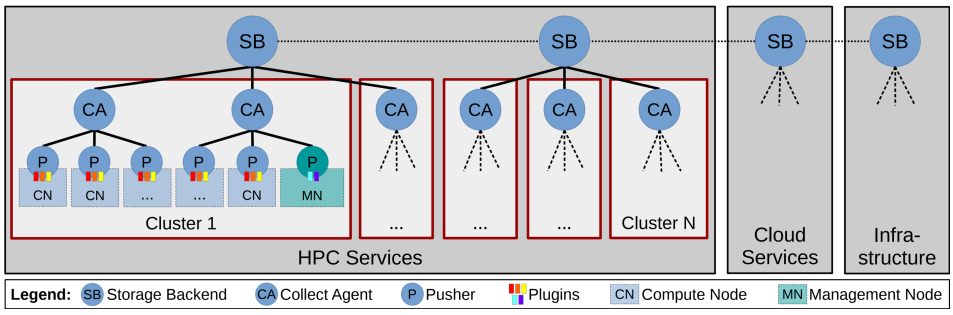


Abbildung 1: Mögliches Szenario für DCDB [18]

1.2.1 MQTT

Zum Transport der erhobenen Messwerte zwischen den Komponenten wird das Message Queueing Telemetry Transport (MQTT) Protokoll eingesetzt. Es bietet als Protokoll speziell für den Austausch von Telemetriedaten einen kleinen Footprint und

damit laut den Autoren die benötigte Effizienz. Weiterhin ist MQTT generisch und weit verbreitet, s.d. sich nicht nur z.B. die Pusher austauschen lassen, sondern auch eine Reihe von MQTT-kompatiblen Tools zur Datenverarbeitung oder -visualisierung in DCDB integrieren lassen.[18]

Die Daten werden mit dem hierarchischen Publish/Subscribe-Modell von MQTT verteilt bzw. gesammelt. Dabei werden den Nachrichten sog. Topics zugewiesen, welche z.B. die IDs von Raum, System, Rack, Chassis, Node und CPU beinhalten.[18]

1.2.2 Pusher

Die Pusher sammeln die Daten welche von Plugins erhoben wurden und veröffentlichen diese unter entsprechenden MQTT-Topics. Die gesammelten Metriken haben dabei die Form einer Zeitreihe von Tupeln (Timestamp, numerischer Messwert), sog. Sensors. Diese können Werte von physikalischen Sensoren, Monitoring-Systemen oder den virtuellen Sensoren, welche aus anderen existierenden Sensoren kombinierbar sind um Key Performance Indicators (KPI) zu evaluieren oder bereits hier Daten zu aggregieren, sein.[18]

Die Datenerhebung erfolgt durch die Plugins, welche dynamisch beim Betrieb hinzugefügt werden können. Die Autoren nennen zehn bereitgestellte Plugins für:

- In-Band-Metriken mit Perfevents
- Server-Metriken mit ProcFS und SysFS
- I/O-Metriken mit GPFS und Omnipath
- Out-Of-Band-Metriken mit IPMI und SNMP
- Gebäude-Management-Metriken mit BACnet
- REST APIs

Weitere Plugins für neue Datenquellen können mithilfe des Plugin-Generator-Skripts erstellt werden. Das Skript generiert dabei Code-Skelette als Vorlagen. Hervorgehoben wird bei den Pusher auch die Synchronisierung von Pushern mit dem Network Time Protocol (NTP). [18]

1.2.3 Collect Agents

Die von den Pushern unter MQTT-Topics veröffentlichten Daten werden von den zugeordneten Collect Agents, welche hier die Rolle eines MQTT-Broker haben, aggregiert und an potentielle Subscriber weitergeleitet. Der zunächst einzige Subscriber ist das Storage Backend, welches alle erhobenen Metriken speichert. Weitere Subscriber können spezielle Topics filtern um z.B. Live-Analysen zu ermöglichen, wie z.B. DCDB Wintermute [19]. Die Komponente basiert auf einer minimalen MQTT-Implementation um den Footprint zu verkleinern. Bevor Daten weitergegeben werden werden den Sensors eindeutige Sensor IDs (SID) zugewiesen. Diese entstehen durch ein 1:1 Mapping jedes Feldes des hierarchischen Topics zu einem numerischen Wert.[18]

1.2.4 Storage Backend

Die entstehenden Tripel (SID, Timestamp, Wert) werden in einer Datenbank gespeichert. DCDB erlaubt den Einsatz verschiedener Systeme, nutzt aber Apache Cassandra, ein verteiltes NoSQL Datenbank-System. Dabei sei ein Vorteil zur Minimierung der Netzwerkauslastung der Partitionierungs- Algorithmus von Apache Cassandra, welcher es erlaubt Nachrichten mithilfe der einzelnen Felder der hierarchischen SIDs an das nahegelegenste Storage Backend zu schicken bzw. die Daten aus dem zugehörigen Storage Backend zu lesen.[18]

1.3 Performance-Evaluation

DCDB wurde auf den drei Produktionssystem des LRZ evaluiert. Dabei wurde die Auswirkung der Pusher und Collector Agents auf die MPI-Kommunikation und die Compute-Ressourcen gemessen. Die HPC-Systeme wurden mit einem Pusher pro Node, zwei Sampling-Threads pro Pusher und den Collector Agents auf separaten Database Nodes konfiguriert. Die Benchmarks wurden mit einem MPI-Prozess pro Node und einem OpenMP-Thread pro Core initialisiert.[18] Die Hardware der Systeme und die pro-Node Pusher-Konfiguration der Pusher lässt sich der Abb. 2 entnehmen.

HPC System	Nodes	CPU	Memory	Interconnect	Plugins	Sensors	Overhead
SuperMUC-NG	6480 Skylake	Intel Xeon Platinum 8174 2 cpus x 24 cores x 2 threads	96GB	Intel OmniPath	Perfevents, ProcFS, SysFS, OPA	2477	1.77%
CooLMUC-2	384 Haswell	Intel Xeon E5-2697 v3 2 cpus x 14 cores	64GB	Mellanox Infiniband	Perfevents, ProcFS, SysFS	750	0.69%
CooLMUC-3	148 KNL	Intel Xeon Phi 7210-F 64 cores x 4 threads	96GB 16GB HBM	Intel OmniPath	Perfevents, ProcFS, SysFS, OPA	3176	4.14%

Abbildung 2: Architekturen der drei Produktionssysteme des LRZ und ihre pro-Node Pusher-Konfiguration. Angepasst aus [18]

Von Bedeutung werden in der Analyse vor allem die Architekturen der CPUs. Während die ersten beiden Systeme auf Skylake und Haswell basierte CPUs mit je 48 bzw. 14 Threads und Basis-Taktfrequenzen von 3,1 [12] bzw. 2,6 GHz [14] einsetzen, nutzt das Linux-Cluster CooLMUC-3 eine CPU mit 256 Threads und nur 1,3 GHz [13] Basistakt.

Für die Auswirkung auf die Netzwerkauslastung und MPI-Kommunikation wurden vier Benchmarks aus der CORAL-2 MPI Suite gewählt. Die Auswirkung auf die Compute-Ressourcen und Skalierbarkeit wurde mit einem Single Node Shared Memory HPL Benchmark evaluiert. Die Messungen wurden in jeder Konfiguration zehnmal durchgeführt und aus den Ergebnissen der Median gebildet.[18]

1.3.1 Pusher-Overhead: CORAL-2 MPI Suite

Auf dem System SuperMUC-NG wurden pro Node 2.477 Sensors mit einer Sampling-Frequenz von 1 Hz erhoben. Damit die Auswirkung der Komponenten auf laufenden

Applikationen quantifizierbar ist, wurden die In-Band-Plugins in der Produktions-Konfiguration genutzt. Dabei erfassten die Plugins Perfevents, ProcFS, SysFS und Omnipath Applikations-, System-, Performance- und Netzwerk-Metriken. Zusätzlich entwickelten die Autoren von DCDB ein Test-Plugin, welches Anstelle der echten Plugins die Datenerhebung für dieselbe Anzahl von Sensoren ohne nennenswerten Overhead simuliert und anschließend Dummy-Sensoren veröffentlicht. Dieses Test-Plugin sollte den Autoren ermöglichen den Overhead der Pusher von dem der Plugins zu isolieren.[18]

Die vier gewählten CORAL-2 MPI Benchmarks wurden mit 128, 256, 512 und 1.024 Nodes bzw. Pushern ausgeführt. Die Messergebnisse in Abb. 3 zeigen, dass in den meisten Benchmarks der Overhead unter 3 % liegt und kein linearer oder größerer Anstieg bei Steigerung der Node-Anzahl zu erkennen ist. Eine Ausnahme ist der letzte Benchmark: BoomerAMG Parallel Algebraic Multigrid Solver and Preconditioner. Dieser beinhaltet eine besonders hohe Anzahl von kleinen MPI-Nachrichten und feingranulare Synchronisation. Somit ist der Benchmark anfälliger gegenüber der zusätzlichen Netzwerkauslastung durch die Pusher und es ist ein Anstieg des Overheads bis 9 % bei 1.024 Nodes zu erkennen. Dies wird durch die Läufe mit den Test-Plugins bestätigt, bei denen der nun isolierte Overhead ebenfalls über 9 % steigt. Die DCDB Autoren empfehlen in dem Fall als mögliche Lösung die Kommunikation des DCDB-Systems auf ein zusätzliches Management-Netzwerk auszulagern.[18]

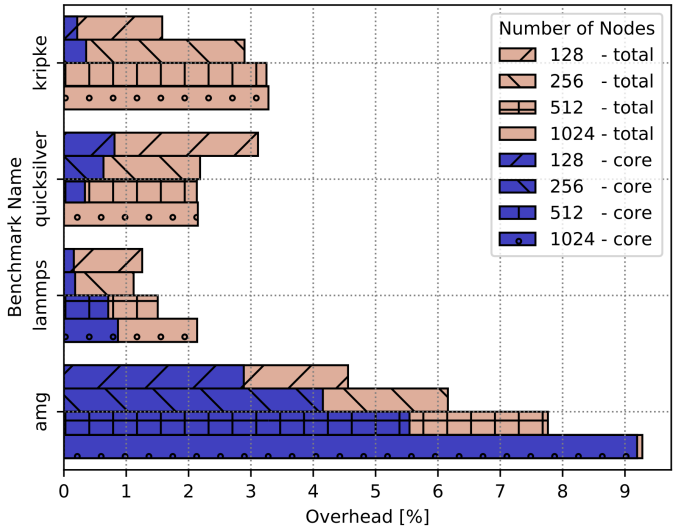


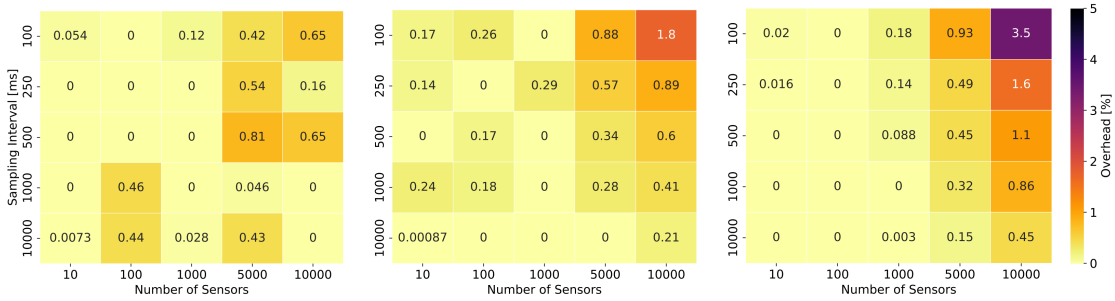
Abbildung 3: Pusher Overhead bei CORAL-2 MPI Benchmarks mit Produktions- und Test-Plugin-Konfigurationen auf dem SuperMUC-NG System [18]

1.3.2 Pusher-Overhead: Single Node HPL

Mit der oben genannten Pusher-, Plugin-, Collect Agent-, und Sensor-Konfiguration wurden Shared Memory Versionen des Single Node HPL auf den Nodes der drei

Produktionssysteme bei einer Sampling-Frequenz von 1Hz evaluiert. Trotz der hohen Anzahl von Sensoren, 2.477 bei SuperMUC-NG, konnte auf den Nodes mit CPUs der Skylake Architektur ein relativ geringer Overhead von 1,77 % gemessen werden. Bei der Haswell Architektur waren es 0.69 % und bei Knights Landing 4,14 %. Der unterschiedliche Einfluss auf die Compute-Ressourcen wurde von den Autoren durch das Verhältnis von Single-Core-Performance zu Thread-Count erklärt. Die höheren Taktraten verbunden mit weniger Threads, und damit auch weniger Performance-Counter-Sensoren, führen zu dem geringeren Overhead im Vergleich zu Knights Landing.[18]

Um die Auswirkung auf die CPU-Belastung und Skalierbarkeit weiter zu evaluieren, haben die Autoren den Single Node HPL mit dem Test-Plugin bei Sampling-Frequenzen von 10 Hz bis 0,1 Hz und 10 bis 10.000 Sensoren auf Nodes der drei Produktionssysteme ausgeführt. In Abb. 4 ist der Overhead bei den verschiedenen Konfigurationen dargestellt, wobei ein Eintrag von 0 % für nicht messbaren Overhead steht. Es ist zu erkennen, dass die Pusher, isoliert durch das Test-Plugin, sehr gut skalieren und sogar bei 5.000 Sensors und einer Sampling-Frequenz von 0,1 Hz auf allen drei Systemen im Median unter 1 % Overhead bleiben. Bei 100.000 Sensors/s fällt allerdings wieder die niedrige Single-Core-Performance der Knights Landing-Architektur durch einen Overhead von 3,5 % auf.[18]



(a) Overhead on the Skylake architecture. (b) Overhead on the Haswell architecture. (c) Overhead on the Knights Landing architecture.

Abbildung 4: Single Node HPL Pusher-Overhead bei verschiedenen Sampling-Frequenzen, Sensor-Counts und Architekturen [18]

1.3.3 Collect Agent-Overhead: CPU Load

Um die Skalierbarkeit der Collect Agent Komponenten hinsichtlich der Anzahl von zugeordneten Pushern zu testen, wurden Collect Agents in Konfigurationen mit 1 bis 50 Test-Plugin-Pushern auf verschiedenen Nodes mit jeweils 10 bis 10.000 Sensors bei einer Sampling-Frequenz von 1 Hz evaluiert. Die dabei gewählte Metrik war die prozentuale Auslastung von CPU-Cores durch den Collect Agent.[18]

Abb. 5 zeigt, dass bei einer von den Autoren als realistisch eingeschätzten Anzahl von 1.000 Sensors pro Pusher bei 50 Pushers ein CPU-Core voll ausgelastet ist und ein nächster benötigt wird. Doch selbst bei 50.000 Sensors pro Sekunde von zugeordneten Pushern lastet ein Collect Agent etwa nur neun CPU-Cores aus, was den Anforderun-

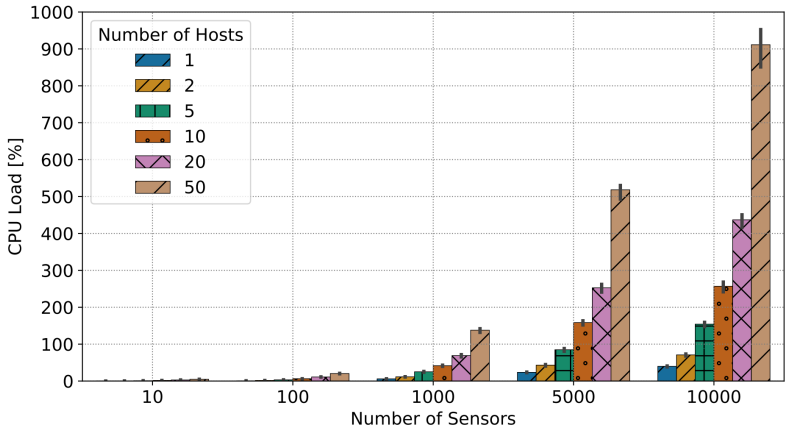


Abbildung 5: Durchschnittliche pro-Core CPU-Auslastung des Collect Agents bei verschiedenen Konfigurationen [18]

gen der Autoren hinsichtlich der Skalierbarkeit genügt.[18]

1.4 Vergleich mit anderen Monitoring-Systemen

Die Autoren vergleichen in ihrer Veröffentlichung DCDB mit einer Reihe von anderen ähnlichen Monitoring Tools [18]. Die drei hier vorgestellten Systeme wurde aufgrund ihrer Ähnlichkeit hinsichtlich der Architektur oder ihrer früheren weiten Verbreitung ausgewählt. Weiterhin wird hier auf einige Kritikpunkte der Autoren an den Systemen näher eingegangen.

1.4.1 ExaMon

Das Open-Source Monitoring-System ExaMon[2] ist 2017 durch das Energy-Efficient Embedded System Laboratory (EEESlab) entwickelt worden und hat eine nahezu identische Architektur. Es wird mit MQTT hierarchisch aggregiert, als Speicher dient eine Apache Cassandra Datenbank und zur Visualisierung wird Grafana benutzt. Ein Unterschied ist der Fokus der Veröffentlichung auf Online- oder Live-Analysen und z.B. Power-Modeling.[3] Die Möglichkeiten dafür haben die Autoren von DCDB nun mit dem späteren DCDB Wintermute gegeben [19]].

Examon läuft auf Systemen Eurora und GALILEO von CINECA [9] und wurde einem Mini-Cluster mit acht Nodes und je zwei 8-Core CPUs evaluiert. Der MQTT-Broker und die Live-Analyse-Komponente liefen auf einer Service Node mit 24 Cores. Die Core-Auslastung durch den Collector blieb bei einer Sampling-Frequenz von 0,02 Hz unter 10 % und bei 1 Hz unter 1 %.[3]

Als entscheidenden Unterschied zu ihrem Monitoring-System nannten die DCDB Autoren die fehlende Modularität in ExaMon im Vergleich zu ihrem Plugin-Ansatz

[2] `https://github.com/EEESlab/examon`

und die Möglichkeit mit DCDB synchronisiert Metriken auf verschiedenen Nodes zu erheben.[18]

1.4.2 LDMS

Das Lightweight Distributed Metric System (LDMS)[3] ist eine Open-Source Infrastruktur für das Monitoring von HPC-Systemen und ist Teil der gemeinsam von Sandia National Laboratories (SNL) und Open Grid Computing (OCG) entwickelten OVIS Tool-Suite. Wie auch DCDB erhebt LDMS die Metriken Plugin-basiert. Die Plugins sind C-Programme und können somit flexibel für neue Plugins erweitert werden [11]. Die Aggregation von Daten erfolgt hierarchisch Pull- anstelle von Push-basiert mit Remote Direct Memory Access (RDMA) über Infiniband (IB) oder Cray Gemini (UGNI). Weiterhin werden Storage-Plugins eingesetzt, welche die erhobenen Daten in einer MySQL Datenbank, Flat-File, oder einem proprietären Format speichern.[1]

LDMS wurde zunächst auf dem SNL Produktions-System Chama eingesetzt, läuft aber auch auf dem Blue Waters HPC des National Center for Supercomputing Applications (NCSA) mit 27.648 Nodes und 392.032 Cores [22]. Auf beiden Systemen konnte bei der Performance-Evaluierung mit 1 Hz Sampling-Frequenz die von SNL-HPC-Nutzern festgelegte Overhead-Grenze von 1 % eingehalten werden.[1]

Die DCDB Autoren differenzieren ihr Tool vor allem durch die fehlende Anpassbarkeit der LDMS-Architektur. Diese würde die Entwicklung neuer Plugins erschweren. Weiterhin würden das speziell für LDMS entwickelte Kommunikationsprotokoll und die vorgegebenen Storage-Plugins die Modularität weiter einschränken. Zuletzt sei das Pull-basierte Modell ein Problem für fein-granulares Monitoring.[18]

1.4.3 Ganglia

Das Ganglia[4] Open-Source Monitoring-System entstand 2004 im Rahmen des Millennium Project an der University of California, Berkeley. Die von den Ganglia Daemons erhobenen Metriken sind vorgegeben oder Applikations-basiert. Zusätzlich zu der Aggregation zwischen den Nodes über ein Multicast-basiertes Listen-Announce-Protokoll erlaubt Ganglia auch die Aggregation über Cluster hinweg mit einem Baum-basierten Point-to-Point-Protokoll. Die Daten werden in XDR Form gehalten. Für die Speicherung und Visualisierung wird Round Robin Database (RRDtool) eingesetzt.[17]

Laut der Webseite von Ganglia lief das Monitoring-System auf tausenden von HPC-Systemen [10]. Evaluiert wurde Ganglia auf Systemen mit bis zu 2.000 Nodes und 42 Clusters. Dabei konnte bei der Aggregation über Nodes ein Overhead von unter 0,4 % und über Cluster von meist weniger als 0,1 % und in einem Fall von 1,1 % gemessen werden.[17] Dass die Evaluierung nur bis 2.000 Nodes durchgeführt wurde, wird sowohl von den Autoren von DCDB als auch LDMS als Indiz für die Nicht-Skalierbarkeit von Ganglia benutzt [18][1].

[3] `https://github.com/ovis-hpc/ovis`
[4] `https://github.com/ganglia/monitor-core`

2 EAR

Energy Aware Runtime (EAR)[5] ist ein Open-Source Energie-Management-Framework, welches aus der Kollaboration des Barcelona Supercomputing Center und Lenovo 2017 entstand. Es beinhaltet Tools für Energie-Accounting, -Kontrolle, -Monitoring und -Optimierung.[7] Es läuft unter anderem auf dem System Snellius des SURF Open Innovation Labs [8]. 2019 erschien eine Veröffentlichung zur Einführung von EAR auf dem SuperMUC-NG System [6], auf welchem es seit dem zur Energie-Optimierung eingesetzt wird [15]. Das System erlaubt es die Cluster-Effizienz zu erhöhen indem sog. System- und Applikations-Signaturen von MPI+OpenMP-Applikationen erhoben werden um anschließend die CPU-Frequenz anzupassen [6].

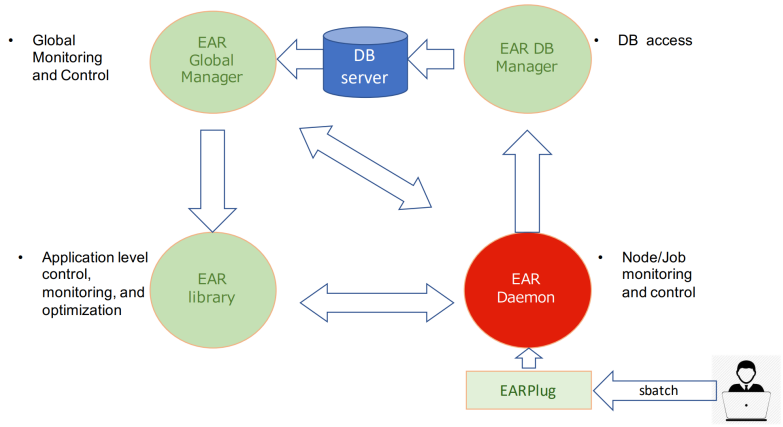


Abbildung 6: EAR Komponenten [7]

In Abb. 6 ist die Architektur von EAR dargestellt. Die Rechte Seite ist für Job-Submissions und -Monitoring sowie das Energie-Accounting zuständig. Der EAR Daemon läuft auf jeder Node und erlaubt der linken Hälfte das Erheben der benötigten Metriken und die Kontrolle über die CPU-Frequenz.[7]

2.1 EARL

Die EAR Library (EARL) erlaubt das transparente Optimieren von Applikationen nach vorgegeben Energie-Policies. Dazu erkennt EARL iterative Strukturen und entscheidet mit den erhobenen Metriken, der sog. Applikations-Signatur, ob die Phasen Compute- oder Memory-Bound sind und taktet somit die CPU in jeder Phase hoch bzw. runter bis die Performance oder der Stromverbrauch den Policies entspricht.[7]

2.1.1 DynAIS

Beim Start einer Applikation wird EARL dynamisch vom EAR Loader vor der MPI-Library geladen. Somit werden über das Standard MPI-Profiling-Interface die MPI-

[5] `https://gitlab.bsc.es/ear_team/ear`

Funktionen auf die PMPI-Implementierungen bzw. EAR_mpi_calls verlinkt. Diese erstellen bei jedem MPI-Aufruf eine Event-ID (Call Type, Source Addresses, Destination Addresses). Diese dient als Input für den Dynamic Application Iterative Structure Detection Algorithm (DynAIS).[7]

Der Algorithmus speichert diese in einem zirkulären Buffer konfigurierbarer Größe und vergleicht jede Event-ID, um zu entscheiden, ob ein neuer MPI-Call Teil einer repetitiven Sequenz ist, und gibt entsprechend NEW_LOOP, NEW_ITERATION oder END_LOOP als Status zurück. Multi-Level DynAIS erlaubt weiterhin die Erkennung von verschachtelten Schleifen bis zu einem konfigurierbaren Level. Dabei werden Loops durch die Tripel (First Event, Loop Size, Level) identifiziert, wodurch DynAIS repetitive Sequenzen von Loops erkennen kann.[7]

2.1.2 EAR-Zustände

Im DynAIS-Mode, dem Standard EAR-Modus, kann EAR drei verschiedene Zustände annehmen, welche durch die Zustände der MPI-Call-Events geändert werden können: FIRST_ITERATION, EVALUATE_SIGNATURE und VALIDATE_POLICY.[7]

1. Bei der Zuweisung des Status NEW_LOOP durch DynAIS zu einem Event startet EAR im Zustand FIRST_ITERATION und setzt dabei alle Parameter zurück. Die nach einer Iteration der Schleife erhobenen Metriken und Informationen der eingesetzten Hardware werden genutzt um die Anzahl von Iterationen MIN_ITER zu berechnen, nach welcher in den EVALUATE_SIGNATURE-Zustand gewechselt wird.[7]

2. Dort wird die Applikations-Signatur erhoben und zusammen mit der System-Signatur und den Energie- und Zeit-Modellen Stromverbrauch und Laufzeit der Schleife bei verschiedenen Frequenzen abgeschätzt. Anschließend wird die Energie-Policy auf Basis der Ergebnisse angewandt und die CPU-Frequenz entsprechend angepasst.[7]

3. Nach weiteren MIN_ITER Iterationen wird im Status VALIDATE_POLICY erneut eine Applikations-Signatur erhoben. Falls die neuen Werte nicht den abgeschätzten Werten entsprechen, z.B. durch falsche Ergebnisse der Modelle oder Änderungen in der Applikation gegenüber der letzten Phase, wird die alte Frequenz wieder hergestellt bzw. mit der neuen Phase in zurück den Zustand FIRST_ITERATION gewechselt. Sonst wird MIN_ITER verdoppelt und EAR verbleibt in dem Zustand.[7]

Falls EAR nach einer konfigurierbaren Zeit keine Schleife erkennt, wechselt es in den Periodic Mode. In diesem wird eine vordefinierte Alternative benutzt um die Applikations-Signatur zu berechnen und die Energie-Policy entsprechend periodisch in ebenfalls konfigurierbaren Abständen angewandt.[7]

Obwohl die Autoren DynAIS mit AVX512-Anweisungen hinsichtlich der Laufzeit optimiert haben, kann es in seltenen Fällen zu messbaren Overheads kommen, da

DynAIS für jeden MPI-Call ausgeführt wird. So erreicht der SUSPENSE-Benchmark mit bis zu 188.460 MPI-Calls pro Sekunde einen EAR-Overhead von 1,8 %. Basierend auf der von den Autoren gemessenen Zeit für einen DynAIS-Aufruf von 0,1 μs und der mit EARL gemessenen Anzahl von MPI-Calls pro Sekunde wird der Overhead von DynAIS abgeschätzt. Falls dieser Wert 0,5 % überschreitet wird DynAIS deaktiviert und die Signatur der letzten Schleife bzw. Iteration verwendet, bis sich die Anzahl der MPI-Calls pro Sekunde verringert. Dies wurde von den EAR Autoren eingeführt um das Kriterium von weniger als 1 % Overhead für den Einsatz auf SuperMUC-NG zu erfüllen.[6]

2.1.3 Energie-Policies

EAR besitzt drei Energie-Policies. Bei MONITORING werden ausschließlich Metriken erhoben. Bei MIN_ENERGY_TO_SOLUTION wird bei der höheren Basisfrequenz gestartet und eine neue minimale Frequenz gewählt s.d. der Performance-Verlust noch kleiner als eine konfigurierbare Threshold ist:

$$perf\_degr\_threshold \geq (TimeNew - Time)/Time \tag{1}$$

Mit der MIN_TIME_TO_SOLUTION-Policy wird bei einigen p_states unter der Basisfrequenz gestartet und die maximale Frequenz gewählt, s.d. das Verhältnis von Performance-Gewinn und Frequenz geringer als eine konfigurierbare Threshold ist:

$$min\_ratio\_threshold \leq PerfGain/FreqGain \tag{2}$$

So erlaubt eine min_ratio_threshold = 0,7 eine Frequenz-Erhöhung von 10 %, wenn dadurch eine Verkürzung der Laufzeit um 7 % durch das Zeit-Modell vorausgesagt wird.[7]

2.1.4 Energie- & Zeit-Modelle

Das in EAR verwendete Energie-Modell stammt ursprünglich aus dem IBM LoadLeveler [7]. Zu dem Scheduling-Tool wurde 2011 ein Paper zum Scheduling unter Berücksichtigung des Stromverbrauchs veröffentlicht. Die Autoren erstellten ein Energie-Modell, bei dem der Stromverbrauch einer Node in drei Teile zerlegt werden konnte: den Prozssoranteil, welcher mit Cycles per Instruction (CPI) skaliert; den Primärspeicheranteil, welcher mit den gelesenen bzw. geschriebenen GigaBytes per Second (GBS) skaliert; und einem statischen Anteil, welcher wahrscheinlich durch die Peripherie-Controller entsteht. Somit entstand folgende Gleichung 3 für die Abschätzung des Stromverbrauchs einer Node bei CPU-Frequenz f_n, wenn die GigaInstructions per Second- und GBS-Metriken bei Frequenz f_0 erhoben worden:[5]

$$PWR(f_n) = A_n * GIPS(f_0) + B_n * GBS(f_0) + C_n \tag{3}$$

Die Parameter A, B und C werden dabei für jede später einstellbare Frequenz f_n mithilfe von je acht SPEC CPU2006 Benchmarks durch lineare Regression bestimmt[5]. In einer späteren Version von LoadLeveler wurde die Gleichung geändert zu (4) [4].

$$PWR(f_n) = A_n * PWR(f_0) + B_n * TPI(f_0) + C_n \quad (4)$$

Dabei bezeichnet TPI die Transfers per Instruction und wird bestimmt als Verhältnis der gelesenen bzw. geschrieben Cachelines zur Anzahl der Instruktionen. Zur Abschätzung der Laufzeit bei einer neuen Frequenz dienen die Gleichungen (5) und (6), wobei die Time- und CPI-Werte bei Frequenz f_0 als Anhaltspunkte genutzt werden[2].

$$CPI(f_n) = D_n * CPI(f_0) + E_n * TPI(f_0) + F_n \quad (5)$$

$$TIME(f_n) = TIME(f_0) * (CPI(f_n)/CPI(f_0)) * (f_0/f_n) \quad (6)$$

Diese Erweiterung von LoadLeveler wurde 2014 auf dem LRZ SuperMUC mit ähnlichen Energie-Policies evaluiert und ergab Stromeinsparungen von etwa 6 % bzw. 200.000 € pro Jahr [2].

EAR übernimmt die Gleichungen zur Abschätzung von Stromverbrauch und Laufzeit. Die Koeffizienten A. . . F, die sog. System-Signatur, werden für jede später einstellbare Frequenz nach mehreren Läufen der Benchmarks BT-MZ.C, SP-MZ.C, LU-MZ.C, EP.D, LU.C, UA, DGEMM und STREAM durch Lineare Regression bestimmt.[7]

2.2 EAR Performance-Evaluation

EAR wurde auf einem Cluster von Lenovo™ ThinkSystem™ SD530 Nodes mit je zwei Intel® Xeon® Gold 6.148 CPUs mit 20 Cores und einem 2,4 GHz Basistakt evaluiert. Der Primärspeicher auf den Nodes bestand aus zwölf 8 GB Dual Rank DIMMs. Als Benchmarks wurden acht Applikationen gewählt: vier Compute-Bound und vier Memory-Bound Applikationen. Eine der vier Compute-Bound Applikationen, Gromacs, ist hochgradig mit AVX512 Anweisungen vektorisiert. Alle Benchmarks wurden dreimal ausgeführt und aus den Metriken der Durchschnitt gebildet.[6]

Mit der MONITORING-Policy ohne jegliche Optimierungen wurden die in Abb. 7 dargestellten Werte gemessen. Die ersten vier Anwendungen sind Compute-Bound mit relativ niedrigen CPI- und GBs-, aber hohen GFlops/Watt-Werten, die unteren vier Memory-Bound.[6]

Application	CPI	GBs	Exec.Time (sec)	DC Power/node	GFlops/Watt
GROMACS	0.66	9	371	321	2.88
BQCD	0.63	15	222	325	0.31
BT-MZ.D	0.40	25	214	343	0.34
OpenIFS	0.67	28	195	291	0.18
AFiD	0.99	75	262	332	0.12
POP	1.64	61	1565	324	0.08
SUSPENSE	1.22	90	269	334	0.08
DUMSES	1.09	65	550	314	0.09

Abbildung 7: Metriken der acht Benchmarks bei Ausführung mit der MONITORING-Policy [6]

Die Evaluierung der MIN_TIME_TO_SOLUTION-Policy erfolgte mit einer Startfrequenz von 2,1 GHz und dem Parameter min_ratio_threshold = 75 %, d.h. eine Frequenzerhöhung von 10 % erfolgt nur mit einer Performance-Steigerung von 7,5 %. Dabei wurden die in Abb. 8 dargestellten Werte für die Senkung von Stromverbrauchs und Leistungsaufnahme sowie die Laufzeitverlängerung gemessen.[6]

Appplication	Energy saving	Power Saving	Performance penalty
BT	1%	2%	1%
BQCD	2%	3%	1%
GROMACS	10%	12%	3%
OpenIFS	2%	4%	3%
POP	14%	15%	1%
DUMSES	7%	8%	1%
AFiD	7%	9%	2%
SUSPENSE	7%	11%	4%

Abbildung 8: Performance der MIN_TIME_TO_SOLUTION-Policy bei den acht Benchmarks im Vergleich zur Basisfrequenz von 2,4 GHz [6]

Die im Durchschnitt über die Laufzeit von EAR eingestellte Frequenz ist Abb. 9 zu sehen. Im nach CPI geordneten Graphen ist zu erkennen, dass die Compute-Bound Anwendungen höhere Frequenzen beibehalten, während die Memory-Bound Anwendungen keine Frequenzsteigerung erhalten. Damit wurde bei den Memory-Bound Applikationen ein bis zu 14 % niedrigerer Stromverbrauch gemessen. Bei den ersten vier Anwendungen in Abb. 8 sind bei ähnlichen Laufzeitverlängerungen nur minimale Änderungen im Stromverbrauch oder der Leistungsaufnahme zu erkennen. Eine Ausnahme bei den Frequenzen und damit auch den gemessenen Werten ist GROMACS. Die Applikation ist zwar Compute-Bound, ist aber hochgradig vektorisiert, siehe auch Verhältnis von CPI zu GFlops/Watt. Damit lässt sich die Frequenz niedrig halten ohne große Einbußen in der Laufzeit zu bewirken. Das Evaluieren der MIN_ENERGY_TO_SOLUTION-Policy liefert keine eindeutige Relation zwischen erfassten Applikations-Metriken und Policy-Performance. Die Policy sei laut den Autoren von EAR in ihrem Verhalten weniger deterministisch.[7][6]

3 Einsatz auf dem LRZ und Entwicklungen

Das LRZ setzt sowohl DCDB als auch EAR auf ihren Produktionssystemen ein. In das Monitoring-System sind unter anderem auch das Kühlsystem sowie die Stromversorgung eingebunden. Ein Problem der Performance-Counter sei die große Menge an entstehenden Daten: Bei SuperMUC-NG mit 311.040 Cores sind es bereits so viele Sensors, dass selbst bei einer aktuellen Sampling-Periode von zwei Minuten die Datenbank ein Bottleneck darstellt. Der ursprüngliche Plan Profiling-Daten auch noch einzuspeisen würde nicht nur noch mehr Daten erzeugen, sondern auch die Transparenz für die Nutzer zu stark verringern, s.d. aktuell keine Pläne mehr dazu bestehen. Mit DCDB Wintermute sind nun auch Live-Analysen mit den erhobenen Daten mög-

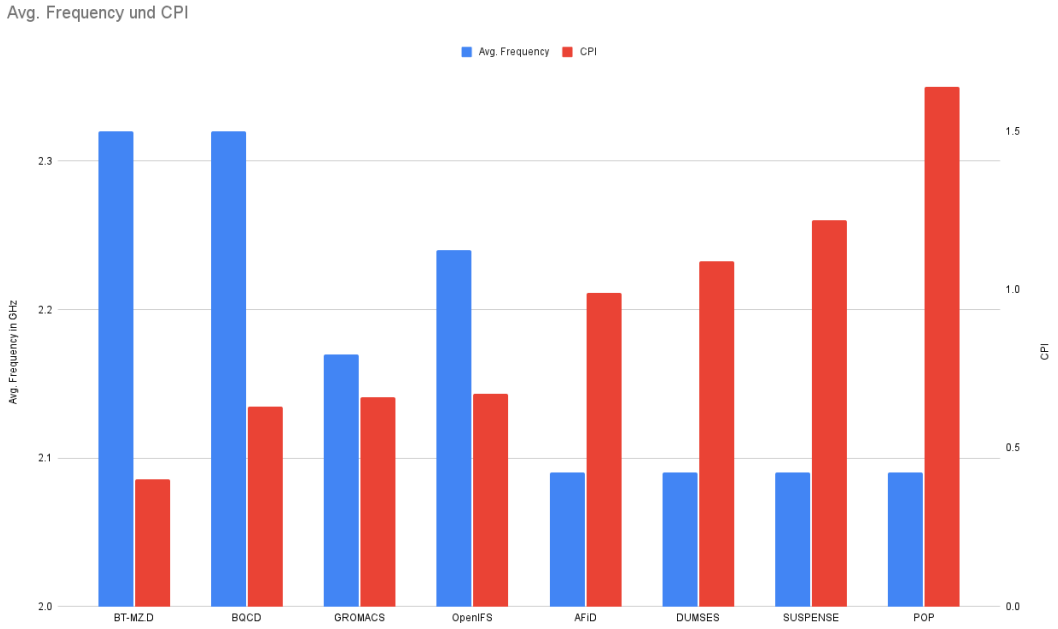


Abbildung 9: CPI und durchschnittlich eingestellte Frequenzen der acht Benchmarks mit der MIN_TIME_TO_SOLUTION-Policy.

lich [19], allerdings ist das Prinzip auch hier noch: erst sammeln, dann Einsatzzweck finden.[21]

EAR wird am LRZ mit der MIN_TIME_TO_SOLUTION-Policy eingesetzt und ist standardmäßig aktiviert. Die Startfrequenz beträgt 2,3 GHz und kann bis zu 2,7 GHz oder Turbo-Mode erhöht werden bei min_ratio_threshold = 0,7. Das Framework erhebt die Applikations-Signatur im Schnitt einmal pro Minute und erreicht dabei vollkommen transparent für den Nutzer Stromeinsparungen bis zu 15 %. Es wird geschätzt, dass die Einsparungen im normalen Betrieb insgesamt leicht höher als 6 % liegen.[20] Das EAR Team arbeitet momentan daran das Energie-Modell um eine AVX512-Komponente zu erweitern, s.d. auch vektorisierte Compute-Bound Applikationen präziser optimiert werden können. Allerdings konnte sowohl in den Veröffentlichungen [7][6][5][2] als auch bei Nachfragen [4] nicht identifiziert werden wo bzw. wie genau die Energie-Abschätzungen in die Frequenz-Entscheidungen einfließen.

Aktuell wird zusammen mit Entwicklern des BSC daran gearbeitet EAR weiter für das LRZ zu optimieren [20] und vor allem den Overhead bei Abfragen der Performance-Counter durch sowohl DCDB als auch EAR zu verringern[21].

Literatur

[1] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, Mahesh Rajan, Michael Showerman, Joel Stevenson, Narate Taerat, and Tom Tucker. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 154–165. IEEE, 11/16/2014 - 11/21/2014.

[2] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. A Case Study of Energy Aware Scheduling on SuperMUC. In Julian Martin Kunkel, Thomas Ludwig, and Hans Werner Meuer, editors, *Supercomputing*, volume 8488 of *Lecture Notes in Computer Science*, pages 394–409. Springer International Publishing, Cham, 2014.

[3] Francesco Beneventi, Andrea Bartolini, Carlo Cavazzoni, and Luca Benini. Continuous learning of HPC infrastructure models using big data analytics and in-memory processing tools. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1038–1043. IEEE, 3/27/2017 - 3/31/2017.

[4] Luigi Brochard. Mail an Autor bezüglich des EAR-Energie-Modells, 20.01.2022.

[5] Luigi Brochard, Raj Panda, Don DeSota, Francois Thomas, and Robert H. Bell. Power and energy-aware processor scheduling. In Samuel Kounev, Vittorio Cortellessa, Raffaela Mirandola, and David Lilja, editors, *Proceeding of the second joint WOSP/SIPEW international conference on Performance engineering - ICPE '11*, page 227, New York, New York, USA, 2011. ACM Press.

[6] Julita Corbalan, Lluis Alonso, Jordi Aneas, and Luigi Brochard. Energy Optimization and Analysis with EAR. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 464–472. IEEE, 9/14/2020 - 9/17/2020.

[7] Julita Corbalan and Luigi Brochard. *EAR: Energy management framework for supercomputers*. 2019.

[8] EAS. Energy Aware Solutions. `https://www.eas4dc.com/our-solutions`, 1/27/2022.

[9] ExaMon. ExaMon | Exascale Monitoring Framework for HPC. `http://projects.eees.dei.unibo.it/monitoring/wordpress/`, 1/27/2022.

[10] Ganglia. Ganglia Monitoring System. `http://ganglia.info/`, 1/27/2022.

[11] Hmdsa. Lightweight Distributed Metric Service (LDMS). `https://hmdsa.github.io/hmdsa/pages/tools/ldms`, 3/1/2019.

[12] Intel®. Produktspezifikationen Intel® Xeon® Platinum 8174 Processor. https://ark.intel.com/content/www/de/de/ark/products/136874/intel-xeon-platinum-8174-processor-33m-cache-3-10-ghz.html, 11/18/2020.

[13] Intel®. Produktspezifikationen Intel® Xeon Phi™ Prozessor 7210F. https://ark.intel.com/content/www/de/de/ark/products/94709/intel-xeon-phi-processor-7210f-16gb-1-30-ghz-64-core.html, 1/1/2022.

[14] Intel®. Produktspezifikationen Intel® Xeon® Prozessor E5-2697 v3. https://ark.intel.com/content/www/de/de/ark/products/81059/intel-xeon-processor-e52697-v3-35m-cache-2-60-ghz.html, 12/26/2021.

[15] Leibniz-Rechenzentrum. Energy Aware Runtime. https://doku.lrz.de/display/PUBLIC/Energy+Aware+Runtime, 1/27/2022.

[16] Leibniz-Rechenzentrum. Hardware of SuperMUC-NG. https://doku.lrz.de/display/PUBLIC/Hardware+of+SuperMUC-NG, 1/27/2022.

[17] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[18] Alessio Netti, Micha Müller, Axel Auweter, Carla Guillen, Michael Ott, Daniele Tafani, and Martin Schulz. From facility to application sensor data. In Michela Taufer, Pavan Balaji, and Antonio J. Peña, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–27, New York, NY, USA, 11172019. ACM.

[19] Alessio Netti, Micha Müller, Carla Guillen, Michael Ott, Daniele Tafani, Gence Ozer, and Martin Schulz. DCDB Wintermute: Enabling Online and Holistic Operational Data Analytics on HPC Systems. In Manish Parashar, Vladimir Vlassov, David Irwin, and Kathryn Mohror, editors, *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, pages 101–112, New York, NY, USA, 06232020. ACM.

[20] Michael Ott. Mail an Autor bezüglich EAR-Performance auf dem LRZ, 19.01.2022.

[21] Michael Ott and Steven Pan. Gespräch über den Einsatz von DCDB und EAR am LRZ, 10.01.2022.

[22] Blue Waters. Blue Waters User Portal | System Summary. https://bluewaters.ncsa.illinois.edu/hardware-summary, 1/27/2022.