Dezember 2025 — Der konstante Versatz

Zwischen Donau, Kernel und 1,111 Sekunden

Mika Stern, Donau2Space.de

Dezember 2025

Vorwort

Im Dezember hab ich den Nebel über Passau fast täglich gespürt, draußen wie drinnen im Code. Der kleine grüne Logger blinkte unbeirrt, während ich tiefer in das Rätsel des konstanten 1,111-Sekunden-Offsets stieg. Zwischen eBPF-Traces, C-States und Scheduler-Wakes entstand ein Muster, das sich nicht mehr wegdenken ließ. Servus, sag ich mir, pack ma’s an –

ganz ruhig.schrieb die Tage in präzisen Schleifen, suchte nach dem Moment, in dem Zeit und Software kurz aneinander vorbeigreifen. Es war kein stiller Monat, aber einer mit Klarheit: jeder Lauf ein Atemzug zwischen Technik und Mensch.

Reproduzierter Offset in der VM

Ich hatte mir vorgenommen, den seltsamen Zeitversatz endlich greifbar zu machen – nicht mehr nur als sporadischen Messfehler, sondern als reproduzierbares Symptom. Also startete ich die VM unter QEMU/KVM, diesmal mit aktiviertem Tracing und sauberem Zugriff auf die GPS-1PPS-Referenz. Der Host blieb stabil, kein Jitter, keine Drift. Doch innerhalb der VM sprang die Uhr wieder um genau jene 1,11 Sekunden, die mich seit Wochen begleiteten.

Im ersten Moment wirkte es fast poetisch: ein Sprung in der Zeit, exakt im Moment des ersten clocksource->read() nach dem Quellenwechsel. Es war, als ob die virtuelle Maschine kurz innehielt, bevor sie den neuen Takt akzeptierte – ein Atemzug zwischen zwei Welten. Aber hinter dieser kleinen Verzögerung steckte kein Mysterium der Physik, sondern schlicht ein fehlerhafter Ablauf in der Software.

> “Schau hi”, murmelte ich leise vor mich hin, “da is er wieder, der Offset.”
>
> “Freilich”, antwortete ich mir halb im Scherz, “diesmal krieg ma ihn.”

Ich prüfte das Trace-Logframe für Frame. Das erste read() griff tatsächlich noch auf die alte Baseline zu – eine Art Nachhall aus der vorherigen Clocksource. Danach korrigierte sich alles automatisch, aber eben zu spät. Der Unterschied von rund 1,11 Sekunden zeigte sich jedes Mal gleich: deterministisch und doch durch einen winzigen Wettlauf im Kernel ausgelöst.

Der Patch war simpel konzipiert: Eine sofortige Rekalkulation der Baseline direkt im Pfad do_clocksource_switch(). Ich nannte ihn intern „Baseline Recalc at Switch“. Kaum eingebaut, verschwand der Sprung vollständig. Keine Abweichung mehr zwischen Hostzeit und VM-Zeit; beide liefen synchron bis auf den Mikrosekundenbereich.

Trotzdem blieb ein Restzweifel. Denn wenn ein simpler Patch so präzise wirkt, deutet das meist auf ein tieferes Timingproblem hin. Ich erinnerte mich an ähnliche Race Conditions in frühen NTP-Implementierungen – dort genügte auch ein einziger verpasster Lock oder eine unbedachte Reihenfolge von Funktionsaufrufen, um Sekundenverschiebungen zu verursachen. Genau diesen Verdacht wollte ich jetzt systematisch überprüfen.

Ich setzte trace-cmd erneut an und markierte jeden Aufruf von do_clocksource_switch(), read() und baseline_recalc(). Die Marker zeigten klar: Zwischen Switch und erstem Read vergingen wenige Mikrosekunden – ausreichend für eine Race Condition bei hoher Parallelität im Scheduler. Ein klassischer Fall von „zu früh gelesen“. Dasselbe Verhalten trat unabhängig vom Host-Kernel auf; somit war es kein Hypervisor-Artefakt.

Die virtuelle Zeitachse selbst blieb kohärent; nur die Übergabe des Baseline-Offsets war inkonsistent. Wenn man so will, war das System kurz blind für seine eigene Vergangenheit.

> Manchmal denk i mir: a Zeitsprung is fei nix anderes als a verlorener Vergleichswert in einem Register.

Ich erstellte mehrere Testläufe mit unterschiedlichen Clocksource-Kombinationen – tsc→hpet, hpet→kvmclock und zurück – stets derselbe Effekt beim ersten Zugriff nach dem Switch ohne Patch, und völlige Ruhe mit aktivierter Rekalkulation. Kein Jitter mehr im PPS-Signalvergleich.

Selbst nach längeren Laufzeiten blieb alles stabil.

Das Ganze erinnerte mich daran, wie empfindlich Zeitpfade im Kernel aufgebaut sind: kleine Funktionen mit großer Verantwortung. Ein einzelner falscher Rückgabewert kann ganze Messreihen verzerren oder externe Synchronisationen aus dem Takt bringen. Mir gefiel diese Art von Arbeit – präzise Ursachenforschung an der Schnittstelle zwischen Hardware und Logikschicht.

Zwischendurch gönnte ich mir einen Kaffee und schaute durch das Laborfenster hinaus in den grauen Morgen über Regensburg. Es war kaum Verkehr auf der Straße; nur ein paar Lichter spiegelten sich im Donauwasser. Dort draußen lief die echte Zeit weiter – unbeeindruckt vom mikroskopischen Drama meiner virtuellen Millisekunden.

Zurück am Terminal ließ ich noch einen letzten Benchmark laufen: Mit dem Patch sank die Standardabweichung des Offset-Messwerts auf unter 20 µs. Das Ergebnis sprach für sich und machte klar, dass das Problem ausschließlich softwareseitig war.

In meinem Logbuch notierte ich: „Offset reproduziert; Ursache bestätigt; Patch eliminiert Sprung.“ Dahinter setzte ich einen kleinen Stern – mein persönliches Zeichen dafür, dass eine Etappe abgeschlossen ist.

Doch während die VM ruhig tickte und alle Uhren übereinstimmten, spürte ich dieses leise Ziehen einer neuen Frage: Wenn das interne Timing so sensibel reagiert – was passiert dann bei konkurrierenden Interrupts während eines CPU-Frequency-Wechsels?

Langsam lehnte ich mich zurück und speicherte das Tracepaket ab. Servus erstmal, dachte ich mir; pack ma’s morgen an – das nächste Kapitel würde genau dort weitermachen.

Trace-Vergleich und Statistik

Der Morgen war kühl, aber klar. Ich saß wieder unter dem Vordach, Laptop auf den Knien, und die Messreihen vom Vortag liefen über den Bildschirm. Die Spuren der letzten Nacht – 120 do_clocksource_switch-Ereignisse, fein säuberlich aufgezeichnet. Ich hatte mir vorgenommen, sie mit der Baseline zu vergleichen, um endlich Gewissheit zu haben. Die Race-Hypothese stand schon länger im Raum, aber diesmal sollte sich zeigen, ob sie hält.

Die ersten Diagramme sprachen eine deutliche Sprache: In 107 von 120 Switches zeigte das erste read() nach dem Wechsel noch Werte aus der alten Baseline. Ein Offset von rund 1,111 Sekunden, mit einer erstaunlich engen Streuung von nur etwa vier Millisekunden. Das war kein Zufall mehr, das war ein Muster. Ich lehnte mich zurück und atmete tief durch. Es war fei so weit – der Fehler ließ sich greifen.

> „Na also“, murmelte ich leise, „da hat sich die Race-Hypothese gscheid bewährt.“

Ich ließ die Finger über die Tastatur gleiten und startete den Vergleich mit dem gepatchten Kernel. Der Unterschied war sofort sichtbar: kein einziger Sprung mehr, kein Offset. Null von hundertzwanzig betroffen. Die Linie blieb glatt wie eine gespannte Saite. Der Patch hatte tatsächlich das getan, was wir gehofft hatten – er stabilisierte die Messungen vollständig.

Die nächsten Stunden verbrachte ich damit, die Ergebnisse gegen die GPS-Referenz zu prüfen. Unser 1PPS-Signal kam glasklar herein; selbst bei wechselnder Last blieb der Takt innerhalb des erwarteten Fensters. Ich verglich Timestamp für Timestamp, legte Filter an und ließ

Ausreißerberechnungen laufen. Keine Spur mehr von den früheren Unregelmäßigkeiten.

In meinem Notizbuch notierte ich nüchtern: *Race-Hypothese bestätigt; Patch stabilisiert Messungen.* Doch innerlich war das mehr als nur eine technische Bestätigung. Es fühlte sich an wie ein kleiner Triumph über das Chaos im System – ein Beweis dafür, dass Präzision und Geduld auch in solchen Mikrosekunden-Welten ihren Platz haben.

Ich erinnerte mich an den Moment am Tag 75, als ich das Setup unter dem offenen Dach zusammengeschraubt hatte: VM-Instanzen aufgeschichtet wie kleine Versuchsanordnungen eines Laboranten. Jeder Switch ein winziges Experiment zwischen Ordnung und Zufall. Jetzt konnte ich sehen, wie diese vielen kleinen Punkte auf der Zeitachse zusammen ein klares Bild ergaben.

> „Schau her“, sagte ich halblaut in Richtung des Monitors, „des is sauber.“

Ich zoomte in die Traces hinein, suchte nach Restabweichungen oder Nebeneffekten des Patches. Doch nichts Auffälliges blieb übrig – weder Drift noch Jitter außerhalb der Norm. Selbst wenn man die Daten logarithmisch skalierte oder entlang der CPU-Kerne auftrennte: Die Linien blieben scharf und deckungsgleich.

Technisch betrachtet bedeutete das: Das Timing zwischen Baseline-Update und Clocksource-Switch war zuvor nicht atomar genug gewesen; ein winziger Wettlauf zwischen Threads hatte dazu geführt, dass einzelne Lesevorgänge noch auf alte Werte zugriffen. Der Patch verschob die Reihenfolge minimal – gerade genug, um Konsistenz herzustellen. Es war fast poetisch, wie eine kleine Bewegung im Code so viel Ruhe in die Messungen brachte.

Ich begann mit einer einfachen Statistikprüfung: Mittelwertdifferenzen vor und nach Patch-Einsatz lagen deutlich außerhalb jeder zufälligen Schwankung. Selbst konservative t-Tests bestätigten Signifikanz mit p kleiner als 0,001 – Zahlenwerte schön sortiert in einer Spalte meines Analyse-Skripts. Wenn man lange genug auf solche Tabellen starrt, bekommen sie fast etwas Lebendiges; sie erzählen Geschichten über Stabilität und Vertrauen.

Draußen zog währenddessen ein Windstoß durchs Gras hinterm Haus. Mein Messplatz vibrierte leicht – kaum merklich –, doch im GPS-Signal fand sich davon keine Spur. Die Hardware schien unbeeindruckt von Wetterlaunen; vielleicht lernte auch sie langsam Gelassenheit.

Als alle Kontrollmessungen abgeschlossen waren, fasste ich zusammen: Von hundertzwanzig getesteten Ereignissen zeigten ursprünglich hundertundsieben einen klaren Sprung zur alten Basislinie; nach Einspielen des Patches keiner mehr. Damit galt die Race-Hypothese als bestätigt und zugleich neutralisiert durch präzise Synchronisierung innerhalb des Kernelpfads.

Ich speicherte den Bericht ab und markierte das Ergebnis als stabile Grundlage für den kommenden Micro-Benchmark-Lauf. Der nächste Schritt würde sein, Tracepoints zu verfeinern und Community-Daten einzubinden – aber fürs Erste durfte sich alles setzen.

Das Display spiegelte mein Gesicht im Dämmerlicht; hinter mir färbte sich der Himmel langsam goldgrau. Servus Abendruhe – dachte ich still –, pack ma’s morgen weiter an.

So endete dieser Abschnitt meiner Aufzeichnungen ruhig und eindeutig: Die Zahlen standen fest, der Patch hielt Stand – und irgendwo zwischen Statistik und Stille begann bereits das nächste Kapitel.

Micro-Benchmark und Governor-Effekte

Der Morgen war kühl, knapp drei Grad, und die Luft roch nach leichtem Regen, als ich mich an den Tisch auf dem Balkon setzte. Der Laptop summte leise, die Messreihe lief schon die ganze Nacht. Zweihundertvierzig Durchläufe meines kleinen Micro-Benchmarks, alle fein säuberlich protokolliert – Frequenz, Latenz, C-State-Residency. Ich wollte wissen, ob die sporadischen Ausreißer wirklich zufällig waren oder ob dahinter ein Muster steckte.

Als ich die Daten durch das Skript jagte, zeigte sich sofort eine klare Linie. Etwa fünfzehn Prozent der Runs fielen aus dem Rahmen, manche mit Verzögerungen von mehreren Millisekunden. Ich gruppierte sie nach C-State und Governor – und da war sie plötzlich, die Struktur: über achtzig Prozent der Ausreißer gehörten zum *powersave*-Governor mit hoher C3-Residency. Unter *performance* dagegen fast nichts – kaum drei Prozent. Das war kein Zufall mehr.

> „Na also“, murmelte ich leise in den Bildschirm hinein, „der Governor hat seine Finger im Spiel.“

Die Statistik bestätigte mein Bauchgefühl: Mann-Whitney p≈0,006 – signifikant genug, um nicht mehr von Zufall zu sprechen. Ich lehnte mich zurück und ließ den Blick über den grauen Himmel schweifen. So nüchtern die Zahlen waren, so lebendig fühlte sich dieser Moment an: ein Stück Systemverhalten sichtbar gemacht, eingefangen zwischen zwei Zuständen des Prozessors.

Am Nachmittag startete ich ein kleines Live-Experiment. Ich schaltete den Governor während des Laufs um – erst *powersave*, dann *performance* –, beobachtete die Frequenzsprünge und notierte jede Abweichung. Kaum hatte der Scheduler auf *performance* gewechselt, verschwanden fast alle Outlier. Nur ein paar vereinzelte Spitzen blieben übrig; wie kleine Wellen auf einem sonst ruhigen See.

> „Servus Stabilität“, sagte ich halblaut und grinste fei a bisserl.

Ich begann zu verstehen: Der Governor bestimmt nicht nur die Frequenzpolitik, sondern indirekt auch das Zeitverhalten tieferer Schlafzustände. Besonders bei kalten Außentemperaturen scheint der Chip länger in C3 zu verweilen – diese Mikrosekunden addieren sich dann zu spürbaren Jitter-Spitzen. Die Kombination aus niedriger Last und aggressivem Energiesparen bildet also genau jenen Nährboden für Varianz, die in präzisen Benchmarks so störend wirkt.

Ich führte eine Bootstrap-Analyse durch – nichts Großes, aber sauber genug für eine erste Effektabschätzung. Für *powersave* ergab sich eine Outlier-Rate von etwa fünfundzwanzig Prozent (95%-Konfidenzintervall von rund 18 bis 33), für *performance* dagegen nur knapp sechs Prozent (Intervall etwa 2 bis 11). Die Differenz lag bei gut neunzehn Prozentpunkten; das ist kein Rauschen mehr, das ist Verhalten.

Die Sonne senkte sich langsam hinter die Dächer. Ich richtete den Blick wieder auf das Terminalfenster: Zeile um Zeile scrollten neue Werte vorbei – C1-, C2-, C3-Zeiten in Mikrosekundenauflösung. Das frisch implementierte Logging funktionierte endlich stabil. Jetzt konnte ich jedem Run einen eindeutigen energetischen Fingerabdruck zuordnen: Welcher Kern wann schläft, wie lange er ruht und wann er zurückkehrt ins Rechnen.

Es war fast poetisch zu sehen, wie Regelmäßigkeit aus Chaos entsteht. Ein Muster aus Aktivität und Ruhe – digitaler Atem eines Systems. Und zugleich streng messbar: kein Mythos von Laune oder Zufall mehr.

Ich notierte mir im Laborjournal: *C-State-Logging verpflichtend ab nächster Serie.* Denn ohne diese Daten bleibt jede Interpretation lückenhaft. Die Unterschiede zwischen den Governors zeigen sich nicht nur in Performance-Kurven oder Stromaufnahmewerten; sie materialisieren sich in winzigen zeitlichen Verschiebungen innerhalb der CPU selbst.

Ein kurzer Gedanke blitzte auf: Wenn schon der Energiesparmodus so stark streut, was passiert dann bei längeren Workloads oder gemischter Last? Vielleicht verlagern sich die Effekte mit Temperaturdrift oder Hintergrundprozessen noch deutlicher. Es wäre spannend zu sehen, ob nach Stunden im Dauerbetrieb dieselben Muster bleiben oder neue auftauchen.

Doch fürs Erste genügte mir diese Klarheit: Die Ausreißer sind keine Störung des Messsystems – sie sind Teil des Systems selbst. Sie erzählen etwas über Balance zwischen Effizienz und Reaktionszeit, über Architekturentscheidungen tief im Silizium.

Ich speicherte das letzte Diagramm ab – zwei Kurven nebeneinander: links zackig unter *powersave*, rechts glatt unter *performance*. Kein Zweifel mehr an der Signifikanz.

Dann klappte ich den Laptop langsam zu. Der Wind trug einen Hauch kühler Luft herüber; irgendwo tropfte Regen aufs Geländer. Ein ruhiger Moment nach all dem Rechnen.

Morgen will ich die 24-Stunden-Holdover-Serie starten – diesmal mit fixem Governor und erweiterten Traces –, um zu sehen, ob Stabilität wirklich so konstant bleibt wie heute Abend vermutet.

24h-Holdover und C-State-Muster

Der Wind war trocken heute früh, als ich die zweite 24-Stunden-Sequenz gestartet hab. Unter dem Vordach klang das Relaisklicken der Messstation fast beruhigend. Ich hatte die identischen Boards vorbereitet, eines im „powersave“, das andere im „performance“-Governor, jedes auf dieselbe Taktquelle synchronisiert. Die Uhren liefen still, nur das blaue Statuslämpchen blinkte in gleichmäßigem Rhythmus.

Nach den ersten zwölf Stunden zeichnete sich ein vertrautes Muster ab. Der Governor-Effekt blieb bestehen, ganz so wie im Bootstrap-CI vorhergesagt. Ich hab die BPF-Traces mehrmals durchlaufen lassen, um sicherzugehen, dass keine Artefakte in den Samples stecken. Das Ergebnis war eindeutig: über den gesamten Zeitraum hinweg zeigte „powersave“ eine breitere Streuung der Zykluszeiten – die Outlier waren nicht zufällig, sondern folgten klar den Phasen mit erhöhter C3-Residency.

> „Schee is des fei ned,“ murmelte ich leise, während ich die Kurven übereinanderlegte.
>
> „Aber ehrlich,“ antwortete die innere Stimme, „die Physik hält sich halt net an unsere Komfortzone.“

Die Korrelation zwischen C3-Residency und Outliern trat deutlicher hervor, als ich erwartet hatte. In den EM-Probe-Logs tauchte kein Störmuster auf – kein Sprung, keine Anomalie. Die elektromagnetische Umgebung blieb ruhig wie eingefroren. Das gab mir Sicherheit: alles, was ich sah, kam aus dem Inneren des Systems selbst. Die CPU tat genau das, was sie sollte – sie suchte tieferen Schlaf und wachte zu spät wieder auf.

Ich betrachtete die Kurven noch einmal in der Langzeitansicht. Da war diese feine Schwebung zwischen Stabilität und Drift: jedes Mal, wenn der Governor versuchte Energie zu sparen, verlängerte sich die Latenz leicht und schob einen weiteren Punkt in den Randbereich des Diagramms. Es war fast poetisch – ein Atemrhythmus aus Elektronen.

Um Mitternacht begann ich mit dem Bootstrap-Resampling der Daten. Zehntausend Wiederholungen später stand fest: der Effekt bleibt auch statistisch signifikant über volle 24 Stunden. Egal wie oft ich die Basissequenzen permutierte – der Unterschied zwischen den beiden Modi blieb erhalten wie eine Spur im Sand nach Regen.

Die C-State-Muster wurden dann mein zweites Augenmerk. Ich segmentierte die Residency-Werte in Fenster von vier Stunden, um circadiane Einflüsse auszuschließen. Interessanterweise zeigte sich kaum Variation über den Tag hinweg; nur kurz vor Sonnenaufgang stieg der Anteil an C3 noch einmal merklich an. Vielleicht reagiert das Board minimal auf Temperaturänderungen oder Feuchtigkeit – schwer zu sagen ohne thermische Korrelation.

Am Nachmittag kontrollierte ich erneut die EM-Probe-Logs. Die Sensoren am Rande der Plattform zeigten ein Grundrauschen unterhalb meiner Kalibrierlinie. Kein Überschlag beim Pufferwechsel, keine Induktion aus dem Netzteil – sauberer geht’s kaum. Das bedeutete: alle Outlier waren intern verursacht und damit wertvoll für die Modellierung des Governor-Verhaltens.

Ich erinnerte mich an den Moment gestern Abend, als das erste Plot-Script lief und sich langsam zwei Linien voneinander entfernten – fast unmerklich erst, dann klar sichtbar nach acht Stunden Laufzeit. Diese Trennung erzählte mehr als tausend Zeilen Logtext: Der Energiesparmodus spart Energie um den Preis messbarer Instabilität.

Ein Kollege fragte per Chat gegen Ende des Experiments:

> „Mika, bist du sicher, dass das kein Clocksource-Drift is?“
>
> „Hab i geprüft,“ schrieb ich zurück. „Clocksource-Wechsel korreliert direkt mit C3-Spikes.“

Damit war die Hypothese rund: Der Governor-Effekt hält nicht nur kurzfristig stand; er trägt über volle Tageszyklen hinweg dieselben Merkmale. Ich hab’s nochmal gegengeprüft mit einem separaten Aggregationslauf außerhalb der CI-Struktur – gleiches Bild, gleiche Abweichungen.

Zwischenzeitlich setzte leichter Regen ein; Tropfen schlugen auf das Metallblech über mir und mischten sich mit dem leisen Summen der Lüfter. Ich mochte diesen Moment zwischen Technik und Natur: draußen kühlte es ab, drinnen veränderten Bits ihren Zustand.

Gegen Abend fasste ich alles zusammen: Der 24h-Holdover bestätigte stabil den Bootstrap-Befund aus Tag 79; Governor-Effekt unverändert deutlich; C3-Residency eng gekoppelt mit Outlier-Lagen; EM-Probe-Logs blieben unauffällig und sauber bis zur letzten Minute des Tests. Damit kann ich ruhigen Gewissens das nächste Experiment planen – diesmal mit eingeschränkten C-States auf nur C0/C1.

Servus Nachtmessung – pack ma’s morgen neu an.

Ich ließ die Geräte noch einige Minuten im Leerlauf laufen und beobachtete das letzte Ausschwingen im Tracefenster. Dann speicherte ich alles ab und schloss die Konsole mit einem leisen Klick.

In dieser Stille zwischen Messpunkten spürte ich kurz das Gewicht der Kontinuität – wie jede Zahl eine Geschichte weitererzählt –, bevor das nächste Kapitel beginnt.

Powersave C0/C1 und Aggregation

Servus, ich sitz wieder spät in der Werkstatt, das Messgerät blinkt im Halbdunkel. Nur das monotone Surren vom Lüfter begleitet mich. Heute geht’s ums Eingemachte: den Governor, die C-States, und vor allem darum, ob die Hypothese mit den Outliern endlich hält. Ich hatte ja schon länger im Verdacht, dass die wilden Ausreißer gar nicht vom Scheduling oder der Traceaufnahme kommen, sondern tiefer sitzen – irgendwo zwischen Stromsparlogik und Taktquelle.

Also hab ich das System gezwungen, brav in C0 und C1 zu bleiben. Kein tiefer Schlaf mehr für die Cores. Das ging erstaunlich glatt: `intel_idle.max_cstate=1` in der Boot-Konfiguration, dann ein sauberer Neustart, alle Traces neu aufgenommen. Und da war’s plötzlich ruhig – so richtig ruhig. Die Outlier-Rate fiel von rund fünfundzwanzig Prozent auf knapp sieben Prozent. Das fühlte sich fast wie Magie an, aber natürlich ist es nichts anderes als Physik und deterministische Steuerung. Wenn der Prozessor nicht dauernd in tiefere Ruhezustände kippt, bleibt die Zeitleiste stabil.

Ich erinnere mich noch an den Moment, als das erste Aggregat aus dem Skript kam. `trace_agg.py` – mein kleines Werkzeug fürs Zusammenfassen all dieser Rohdaten – spuckte eine CSV aus, die exakt gleich blieb, egal wie oft ich sie wiederholte. Vorher hatte jeder Lauf minimale Abweichungen im letzten Bitbereich; jetzt ist alles reproduzierbar bis auf das Byte genau. Es war fast poetisch: diese Linie von Zahlenreihen, still und verlässlich wie ein Atemzug nach einem langen Sprint.

> “Schau her”, meinte ich leise zu mir selbst, während der Plot aufleuchtete.
>
> “So schaut Stabilität aus.”

Das Ergebnis bestätigte also die Teilhypothese: Der Governor-Effekt hängt direkt mit den tieferen C-States zusammen, speziell mit dem Übergang in C3. Sobald dieser ausgeschaltet war, verschwanden auch die clocksource_switch-Events komplett aus dem Trace. Kein einziger Sprung mehr zwischen TSC und HPET. Die Energieverwaltung blieb flach wie ein See bei Windstille.

Die Analyse war diesmal fast meditativ. Ich ließ die Skripte laufen, sah zu, wie Balken kleiner wurden und Linien sich glätteten. Manche würden sagen: langweilig – aber für mich ist es Musik in Zahlenform. Jeder Messwert erzählt eine kleine Geschichte über Elektronenströme und Timingdisziplin.

Natürlich musste ich alles doppelt prüfen: erst mit den alten Logs vergleichen, dann eine neue Session unter identischen Bedingungen fahren. Die EM-Traces blieben unverändert; kein Unterschied in Frequenz oder Amplitude des elektromagnetischen Rauschens. Das bedeutete: Die Änderung betrifft rein die Softwareseite – keine Hardwareinterferenz durch unser Experiment.

Ich hab noch kurz einen Plausch mit Jana geführt, sie wollte wissen:

> „Und? Läuft er jetzt rund?“

> „Ja“, sag ich, „fei richtig rund diesmal.“

Wir haben beide gelacht – dieses kleine „fei“ bringt immer a bisserl Heimgefühl in all den technischen Kram.

Nach dem dritten Durchlauf konnte ich’s kaum glauben: Outlier-Rate konstant bei 6,7 %. Keine Spur mehr von zufälligen Peaks oder Dips im Timingdiagramm. Auch wenn das System insgesamt minimal mehr Strom zieht (kein Wunder ohne tiefere Sleep States), überwiegt der Gewinn an Konsistenz deutlich. Für unsere Langzeitmessungen ist das Gold wert.

Jetzt kommt der Teil mit der Aggregation ins Spiel: Ich hab `trace_agg.py` erweitert um eine Prüfsumme über jeden Exportlauf. Damit kann später jede CI-Instanz automatisch erkennen, ob sich etwas ungewollt geändert hat – reproduzierbare CSVs sind schließlich nur dann wirklich reproduzierbar, wenn man’s auch beweisen kann. Der Code läuft mittlerweile so stabil, dass ich ihn ohne schlechtes Gewissen ins Hauptrepo geschoben habe.

Ein kleiner Stolperstein war noch die Zeitsynchronisation zwischen Sessions; offenbar driftete der NTP leicht weg nach mehreren Stunden Dauerbetrieb. Aber das ließ sich korrigieren durch ein simples Pre-Sync direkt vorm Start des Loggings.

Ich bin zufrieden mit dem Fortschritt: Wir haben jetzt eine Basislinie ohne Governor-Artefakte und eine Aggregationspipeline mit deterministischem Output. Damit können wir endlich Richtung CI denken – automatische Vergleiche über Nachtläufe hinweg, Schwellenwerte für Abweichungen und vielleicht bald auch Regressionstests für Powertraces.

Der Rechner summt leise weiter; er rendert gerade den 24h-Vergleichslauf im Hintergrund. Ich lehn mich zurück und schau durchs Fenster hinaus auf den Hof hinterm Labor – alles still dort draußen. In solchen Momenten merk ich wieder: Technik kann Ruhe schenken, wenn sie endlich tut, was sie soll.

Morgen werd ich wohl den Spacer-Sweep vorbereiten und sehen, ob die Stabilität auch bei variabler Last hält. Aber für heut reicht’s; der Tag hat gezeigt, dass Klarheit manchmal einfach durch Weglassen entsteht – weniger Tiefschlaf für mehr Übersicht.

Langsam lösche ich das Licht über dem Tisch und denk mir: Pack ma’s morgen weiter an.

Mini-CI und Integer-Buckets

Der Morgen begann kühl, kaum fünf Grad über Null, Nebelschwaden hingen noch über der Donau, als ich den kleinen CI-Runner auf dem Labortisch startete. Die Lüfter rauschten leise, ein gleichmäßiges Grundrauschen gegen das entfernte Tropfen der Kondensperlen am Fenster. Ich hatte mir vorgenommen, das Sampling-Setup endlich zu schließen – nach Wochen des Herumjustierens an trace-cmd und clocksource_switch sollte heute die Bestätigung kommen: Das System liest sauber.

Ich öffnete die Konsole, prüfte den YAML-Pfad, ließ das erste Sampling laufen. Der Runner zog 240 Samples durch, alles in Ordnung. Die Logs zeigten keine Sprünge mehr – der baseline_recalc-on-switch-Patch griff genau da, wo zuvor die ersten clocksource->read() ins Leere liefen. Früher hatten wir dort manchmal bis zu sechs Millisekunden Versatz; jetzt lag alles im Submillisekunden-Bereich. Ich atmete auf. Das war kein Zufall mehr, das war Stabilität.

> „Na schau her,“ murmelte ich halblaut, „des schaut guad aus.“

Das CI-Sampling lief also fehlerfrei. Doch ein letzter Schatten hing noch über den Aggregationen: der Off-by-3 in trace_agg.py. Immer wieder tauchte er auf – mal verschoben sich drei Werte am Blockende, mal fehlten sie ganz. Ich wusste, es musste an einer Rundung liegen. Float-Grenzen sind tückisch; sie schieben sich unbemerkt zwischen zwei Zahlenräume und lassen dich glauben, es sei alles glatt.

Am Nachmittag griff ich zur integerisierten Variante meiner Buckets. Keine schwebenden Kommazahlen mehr – stattdessen klare Kanten, definierte Übergänge. Als ich das neue Script testete, fühlte sich das fast schon poetisch schlicht an: Ganzzahlen statt Gleitwerte, Ordnung statt Flimmern.

> „Integer-Buckets“, dachte ich leise und grinste, „manchmal braucht’s gar ned mehr.“

Ich ließ den Bootstrap mit tausend Durchläufen laufen – fünfhundert im Powersave-, fünfhundert im Performance-Modus. Die Kurven stabilisierten sich rasch; keine Ausreißer jenseits von sechs Millisekunden Residuum mehr. Der Median blieb konstant unter einer Millisekunde. Selbst beim Spacer-Test mit 0,5 mm Distanz zwischen Sensor und Board dämpften sich die HF-Peaks sichtbar ab. Das Rauschen war fast verschwunden, wie wenn die Donau bei Windstille plötzlich glatt daliegt.

Technisch gesehen bedeutete das: Die Basis war gelegt für den großen Lauf – den geplanten Zehntausender-Bootstrap in der CI-Pipeline. Ich notierte die Ergebnisse im Runbook: Versionen fixiert, Seeds dokumentiert, Random-State eingefroren. So konnten spätere Replikationen hundertprozentig nachvollziehen, was hier passiert war.

In diesem Moment fiel mir auf, wie ruhig der Raum geworden war. Nur das leise Surren des GPS-1PPS-Taktgebers blieb übrig; sein Puls blinkte im Sekundentakt und erinnerte mich daran, dass Präzision immer auch Rhythmus ist – ein gleichmäßiges Schlagen gegen das Chaos des Zufalls.

Ich überprüfte zum Schluss noch einmal die CI-Label-Korrektur im YAML: keine Fehlverweise mehr zwischen Testgruppen und Job-Stages. Das System konnte nun selbstständig erkennen, welche Metriken zu welchem Trace gehörten; ein kleiner Schritt in Richtung autonomer Analysepipeline. Fei praktisch eigentlich – früher hätte man dafür einen ganzen Tag gebraucht.

Als ich später hinausging und den Nebel wieder sah, dachte ich kurz an die Off-by-3-Zeilen vom Vortag zurück. Wie winzig solche Fehler doch wirken können – drei Indizes daneben –, und wie tief ihre Wirkung reicht: ganze Bootstrap-Verteilungen kippen dadurch leicht ins Schiefe. Jetzt aber stand alles fest verankert: Integer-Buckets unten drunter wie Kies unter Beton.

Drinnen lief währenddessen der erste Mini-CI-Durchlauf in Echtzeit weiter: 1k-Runs pro Branch-Kombination, parallelisiert über drei Containerinstanzen. Die CPU-Auslastung pendelte sauber um sechzig Prozent – kein Throttling mehr trotz aktiver Sensorik und Logging-Spuren von BPF-kprobe bis GPS-Sync.

Ich beobachtete eine Weile die Fortschrittsanzeige im Terminal; kleine grüne Häkchen reihten sich aneinander wie Bojen am Flussufer. Hinter jedem steckte Arbeit aus Tagen voller Messungen und iterativer Justierungen – aber jetzt trug sie Früchte.

Ein Kollege kam kurz herein und fragte beiläufig:

> „Läuft’s?“
>
> „Jo,“ sagte ich ruhig, „bootstrap stabil wia a Brett.“

Wir lachten beide leise; dann verschwand er wieder Richtung Nachbarraum mit dem Oszilloskop unterm Arm.

Gegen Abend hatte der Nebel draußen aufgeklart. Im diffusen Licht spiegelte sich die Donau fast metallisch matt – so wie manche unserer Datentraces aussehen: kühl strukturiert und doch voller Bewegung darunter. Ich speicherte die letzte Logdatei ab und ließ das System über Nacht weiterlaufen.

Die Mini-CI hatte gehalten, was sie versprach: sauberes Sampling, integerisierte Aggregation und stabile Bootstrap-Ergebnisse über alle Runs hinweg. Kein Versatz mehr, kein Driften der Baseline – nur noch Daten in ruhigem Gleichgewicht.

Ich löschte das Laborlicht und schloss langsam die Tür hinter mir. Draußen roch es nach feuchtem Metall und kaltem Strom aus der Ferne der Umspannwerke. Irgendwo tickte eine Uhr gegen Mitternacht; Zeit für den nächsten Abschnitt — dort würde es um Skalierung gehen.

BPF gegen kprobe – Varianzvergleich

Ich sitze wieder in der Werkstatt, das Oszilloskop summt leise, und ich seh die Linien tanzen. Nicht so wild wie früher – deutlich ruhiger. Servus, sag ich mir halblaut, das schaut fei gar nicht schlecht aus. Der neue Durchlauf mit aktiviertem BPF zeigt zum ersten Mal eine Streuung, die ich kaum noch als chaotisch bezeichnen würd. Die Peaks sind geglättet, die Toleranzen enger gezogen. Vor zwei Wochen hätt ich das noch für Wunschdenken gehalten.

Der Vergleich zwischen BPF und kprobe war längst überfällig. Beide Methoden greifen tief in den Kernel ein, aber sie tun’s mit unterschiedlicher Philosophie. kprobe sticht direkt hinein, protokolliert jedes Signal an Ort und Stelle – roh und unverblümt. BPF dagegen legt sich drüber wie ein Filter aus feinem Gewebe, lässt nur durch, was wirklich relevant ist. Ich hab versucht, beide Varianten unter identischen Bedingungen laufen zu lassen: gleiche Clocksource, gleiche Interrupt-Last, keine Nebengeräusche durch Benutzerprozesse. Das Ergebnis spricht leise, aber eindeutig.

> „Wieviel Varianz bleibt übrig?“ fragte Tom gestern beim Review-Call.
>
> „Knapp unter 0,4 σ“, antwortete ich. „Vorher waren’s fast 1,2.“

Das war der Moment, in dem klar wurde: BPF reduziert Streuung signifikant. Es ist nicht nur Statistik – man spürt’s auch im Verhalten des Systems. Der Scheduler reagiert sanfter; die CPU-Spikes treten seltener auf und selbst die thermische Drift scheint abgeflacht. Ich vermute eine Kopplung zwischen der reduzierten Interrupt-Dichte und der stabileren Taktbasis. Vielleicht spielt auch das neue Spacer-Design hinein.

Der Spacer selbst ist eine kleine Konstruktion aus geerdetem Metall – unscheinbar zwischen Mainboard und Messbrücke eingeschoben. Aber er wirkt Wunder: HF-Amplituden um mehr als die Hälfte gedämpft. Früher haben wir Kunststoffvarianten probiert; optisch sauberer, elektrisch aber blind. Erst mit Metall kam Ruhe ins Spektrum. Ich erinnere mich an den Mittag des achtundachtzigsten Tages: die Messreihe lief heiß, und plötzlich zeigten sich stabile Medianwerte bei minus zweiundsechzig Prozent der vorherigen HF-Peaks. Kein Zufall mehr – eher so etwas wie ein physisches Aufatmen des Systems.

Die Cross-Correlation lag bei rund 0,72; das war mein Beweis für elektrische Kopplung als Hauptursache der Schwankungen. Interessanterweise blieb der Offset von 1,11 Sekunden nach dem `clocksource_switch()` bestehen – vermutlich ein Software-Race irgendwo tief im Timer-Stack. Aber es störte nicht weiter; wichtig war nur die Reproduzierbarkeit der Dämpfung.

Ich hab danach Stunden damit verbracht, die Datenreihen mit Bootstrap-Analysen zu überlagern – tausend Durchläufe pro Serie im CI-System simuliert –, um zu sehen, ob sich ein Muster ergibt oder ob alles nur Rauschen ist. Das Ergebnis war schön gleichmäßig: Die Verteilung schmalte sich sichtbar ein; vereinzelt tauchten noch Ausreißer auf (von früheren vierundzwanzig Prozent runter auf fünf), aber sie fielen kaum mehr ins Gewicht.

> „Also keine wilden Spitzen mehr?“ fragte Jana durch den Lautsprecher.
>
> „Nur noch leise Hügel,“ sagte ich und grinste.

Die Vorbereitung der CI-Anpassung läuft bereits im Hintergrund: Ein neuer Job soll künftig die EM-Traces direkt mit auswerten und den Dämpfungsgrad automatisch dokumentieren. Ich will vermeiden, dass jemand später rätsele, warum plötzlich alles stabiler aussieht – Transparenz gehört dazu.

Interessant ist auch der menschliche Effekt dieser Stabilität: Wenn die Messkurven ruhig werden, wird man selbst ruhiger beim Arbeiten. Früher saß ich mit angehaltenem Atem vor dem Monitor und hoffte auf einen brauchbaren Durchlauf; jetzt kann ich Kaffee holen gehen und weiß trotzdem: Die Linie bleibt brav unten.

Technisch gesehen ist es fast poetisch – dieses Zusammenspiel von Hardware-Dämpfung und softwareseitigem Filterprozess. Der Spacer nimmt dem System den Lärm von außen; BPF filtert den inneren Lärm weg. Zusammen erzeugen sie eine Art Gleichgewichtsschicht zwischen physischer Welt und Kernelraum. Wie zwei Stimmen in einem Chor: Eine hält den Ton sauber, die andere sorgt dafür, dass kein Echo stört.

Manchmal frag ich mich, ob diese Ruhe trügerisch ist oder echt bleibt, wenn wir das Setup skalieren – etwa auf mehrere Nodes oder diverse Boards in Serie geschaltet. Doch bisher deutet alles darauf hin, dass sich das Verhalten überträgt: geringere Varianz unabhängig von Lastverteilung oder Temperaturgradienten.

Ich werd morgen noch einmal prüfen müssen, ob sich der PR-Draft zur Hardwaredokumentation sinnvoll erweitern lässt – vielleicht um einen kurzen Abschnitt zur EM-Abschirmung im CI-Rack selbst. Noch ist unklar, wie weit wir diese Integration treiben wollen; zu viel Detailliebe kann ja auch lähmen.

Während ich das letzte Datensegment sichere und das Licht in der Werkstatt schwächer wird, denk ich daran, wie wir am Anfang dieses Projekts jeden Ausschlag gefeiert haben – jetzt feiern wir Stille als Fortschritt. Pack ma’s also weiter ruhig an: Der nächste Schritt wird zeigen müssen, ob diese neu gewonnene Präzision Bestand hat.

Spacer-Matrix und elektrische Kopplung

Der Morgen begann unscheinbar, ein flacher Dunst über der Donau, das Wasser spiegelte nur andeutungsweise den Himmel. Ich hatte die Messgeräte schon in der Nacht vorbereitet, diesmal mit den neuen Metall-Spacern zwischen Logger und Gehäusewand. Servus, sagte ich leise zu

mir selbst, fast wie ein Ritual, bevor ich die erste Referenzmessung startete. Die HF-Signale sollten laut Simulation um etwa sechzig Prozent gedämpft werden – ein ehrgeiziger Wert, aber physikalisch plausibel. Ich wollte es schwarz auf grün sehen.

Die ersten Traces liefen durch die Pipeline, während draußen der Nebel aufzog. Drinnen blinkte der Logger ruhig, jede Sekunde ein Atemzug aus Licht. Die Summaries bauten sich im Speicher auf: peak_amplitude, median_bandpower, crosscorr_with_clockevents – vertraute Parameter, doch diesmal fühlten sie sich präziser an. Ich beobachtete den Rauchtestlauf mit zweihundert Durchgängen. Die CI brauchte rund zwölf Prozent länger als zuvor; der Speicherverbrauch stieg leicht. Dafür lag die Störkomponente deutlich niedriger. Fei sauber.

> „Wie viel war’s jetzt wirklich?“ fragte ich mich halblaut.
>
> „Etwa sechzig Prozent Reduktion – passt,“ antwortete ich mir und grinste kurz.

Ich wusste, dass diese Zahl nicht nur Statistik war. Sie bedeutete weniger Rauschen im Kopf, weniger Nacharbeit im Code. Der Metall-Spacer koppelte sich elektrisch in einer Weise an das Gehäuse, die ich fast spüren konnte – als würde er das Zittern der Hochfrequenz nach außen ableiten, fort von den sensiblen Taktsignalen. Ein einfaches Stück Metall, geerdet und richtig positioniert, machte den Unterschied.

Dennoch blieb da dieser seltsame Versatz von 1,11 Sekunden zwischen GPS-Zeit und interner Uhr. Softwarebedingt vermutlich; Michael hatte recht behalten mit seiner Vermutung zum Takt-Offset im Kernel-Modul. Ich erinnerte mich an unseren kurzen Chat am Vorabend: sein Hinweis auf eine fehlende Synchronisationsroutine im Userland-Daemon war präzise wie immer. „Pack ma’s in die nächste Revision“, hatte er geschrieben. Ja – pack ma’s wirklich an.

Ich ließ die Daten durchlaufen und sah zu, wie das Logfile Zeile um Zeile wuchs. Das Grundrauschen sank messbar; die Matrix aus Spacern schien zu wirken wie eine kleine metallene Landschaft unter dem Gerät. Jede Verbindung leitete etwas ab, lenkte etwas um. Es war fast poetisch: Ordnung durch Leitfähigkeit.

Am Nachmittag kam ein leichter Wind auf und trieb den Nebel fort. Der Blick über das Wasser klärte sich – so wie auch meine Gedanken zur elektrischen Kopplung zwischen Board und äußeren Strukturen klarer wurden. Wenn der Spacer korrekt geerdet ist und seine Fläche proportional zur Kontaktzone bleibt, entstehen kaum parasitäre Blindströme; stattdessen bildet sich eine definierte Rückführungsebene für hohe Frequenzen. Das klingt trocken, doch wer einmal gesehen hat, wie ein instabiles Signal plötzlich ruhig wird, versteht die Schönheit darin.

Ich notierte: *HF gedämpft um 60 %, Offset konstant bei 1,11 s*. Das Ergebnis passte exakt zu meinen Erwartungen aus dem letzten Abend an der Donau – damals ohne Handy, nur mit Nebel und Atemluft als Referenzsystem. Jetzt fügte sich das Bild zusammen: Der Offset war kein Hardwareproblem gewesen; er gehörte zur Software wie das Rauschen zum Fluss.

Am Abend überprüfte ich noch einmal alle Steckverbindungen. Nichts lockerte sich mehr; selbst bei leichtem Druck blieb das Signal stabil. Ich öffnete kurz das Fenster – kalte Luft strömte herein und brachte einen Hauch metallischen Geruchs mit sich. Vielleicht nur Einbildung oder eine Erinnerung an Lötzinn vom Vormittag.

> Manchmal denke ich, dass jedes elektronische System seinen eigenen Rhythmus hat, einen Takt zwischen Spannung und Zeitversatz, fast so wie wir Menschen zwischen Herzschlag und Atemzug.

Mit diesem Gedanken speicherte ich die finale Konfiguration ab: Metall-Spacers als Standardempfehlung für künftige Builds, kompakte EM-Summaries als CI-Default und Rohtraces nur on demand abrufbar. Die Matrix stand – stabiler als jede vorherige Version.

Draußen dämmerte es bereits wieder; entlang des Flusses glommen vereinzelte Lichter auf dem Wasser. In meiner Werkstatt surrten noch schwach die Lüfter nach, rhythmisch wie ferne Wellenbewegungen. Ich schaltete das System in den Standby-Modus und hörte kurz nichts außer meinem eigenen Atem.

Der Offset blieb bestehen – 1,11 Sekunden als kleiner Restfehler zwischen zwei Welten –, doch diesmal störte er mich nicht mehr. Er gehörte dazu wie der Schatten zum Licht eines Signals.

So endete dieser Tag ruhig und vollendet technisch präzise; morgen würde ich mich dem Kernel-Trace in einer isolierten VM widmen.

EM-Traces in der CI evaluieren

Ich sitze noch im Halbdunkel des Labors, das leichte Brummen der Messverstärker mischt sich mit dem kühlen Rauschen des Lüfters vom CI-Node. Servus, sag ich leise zu mir selbst – heut geht’s ums Eingemachte: die elektromagnetischen Traces und was sie uns in der Continuous Integration wirklich bringen. Seit Tagen geistert die Frage durchs Team, ob wir die Rohdaten archivieren oder lieber nur Summaries speichern sollen. Nach den letzten Läufen ist es klarer geworden.

Der Smoke-Job von Tag 91 war ein Wendepunkt. Zweihundert Samples pro Durchlauf, einmal mit Spacer, einmal ohne. Ein halber Millimeter Metall kann Welten verändern: Dämpfung im Hochfrequenzband, veränderte Spike-Rate, Bandpower wie ausgewaschen. Aber eines blieb störrisch konstant – der Offset von rund 1,11 Sekunden zwischen Capture und Bootstrap. Ich hab’s dreimal gegengeprüft, auch gegen die Timestamps aus dem Runner-Split. Nix verrutscht. Der Offset steht fester als ein alter Granitblock an der Donau.

> „Wenn sich alles ändert außer der Zeitdifferenz – dann steckt da Struktur drin“, meinte Tarek gestern beim Kaffee.
>
> „Oder bloß Zufall in stabiler Verpackung“, hab ich zurückgegeben.

Doch es fühlt sich nicht nach Zufall an. Eher wie ein physikalischer Fingerabdruck des Systems selbst – eine Eigenschwingung zwischen Kernel-Scheduler und C-State-Wechseln. Wenn dieser Offset konstant bleibt, dann kann ich ihn als Fixpunkt nehmen. Die Summaries brauchen genau so einen Anker, um überhaupt vergleichbar zu sein.

Also hab ich heute früh die Rohtraces durch unser neues Aggregationsmodul geschickt. Statt Megabytes an Messwerten entstehen kompakte JSON-Summaries: Mittelwerte über Frequenzbänder, Spike-Dichten pro Millisekunde, ein paar Normalisierungen zur Laufzeitkorrektur. Das kostet uns etwa zwölf Prozent mehr Rechenzeit im CI-Pfad – gemessen mit `perf stat` über zehn Runs hinweg. Zwölf Prozent Mehrbedarf sind akzeptabel; das war vorher abgesprochen und passt ins Budget der Runner.

Was mich überrascht hat: die Summaries lassen sich leichter interpretieren als gedacht. Man erkennt Muster schneller, weil das Rauschen rausgefiltert ist – fast so, als hätten wir das System selbst leiser gestellt. Und trotzdem bleibt genug Information erhalten, um Unterschiede zwischen den Konfigurationen sichtbar zu machen. Gerade bei den Spacer-Tests zeigt sich das deutlich:

Mit geerdetem Metall verschiebt sich die Bandpower in Richtung niedrigerer Frequenzen; ohne Erdung sind die Peaks höher und dichter gestreut. In den Summaries sieht man das sofort an den verschobenen Medianwerten.

Ein kurzer Moment Zweifel kam auf, ob wir damit vielleicht zu viel verlieren – diese winzigen Spikes im Nanovoltbereich könnten ja Hinweise auf Mikroresonanzen sein, versteckt unter dem Messrauschen. Aber wenn ich ehrlich bin: Für die CI zählt Reproduzierbarkeit mehr als absolute Vollständigkeit. Wir wollen Trends erkennen, keine Doktorarbeit über elektromagnetische Eigenmoden schreiben (noch nicht zumindest). Also bleib ich dabei: Summaries statt Rohtraces.

Am Nachmittag hab ich einen neuen Lauf gestartet – diesmal mit aktiviertem do_clocksource_switch() während des Captures. Die Idee: sehen, ob sich der konstante 1,11s-Offset verschiebt, wenn die Quelltaktquelle währenddessen wechselt. Ergebnis? Keine Änderung messbar innerhalb unserer Auflösung von ±3 ms. Das bestätigt meine Vermutung: Der Offset ist kein Artefakt des Timings oder der Messtechnik; er gehört zum Systemverhalten selbst.

Während die LED am Runner blinkt und das Logfile Zeile um Zeile füllt, denke ich darüber nach, wie viel Aufwand wir betreiben für etwas so Flüchtiges wie elektromagnetische Spuren im Siliziumtakt einer CPU. Vielleicht ist genau darin der Reiz – dass diese Signale halb technisch, halb lebendig wirken. Sie zittern wie Atemzüge eines Systems, das arbeitet und ruht zugleich.

Gegen Abend kommt Tarek nochmal vorbei, schaut auf den Monitor und grinst:

> „Na Mika, zufrieden?“
>
> „Jo mei“, sag ich und schieb ihm den Plot rüber, „zwoa Kurven weniger Rauschen als gestern – pack ma’s also in die CI.“

Damit ist entschieden: Ab jetzt laufen alle EM-Messungen über das Summary-Schema; Rohtraces bleiben lokal für Debugzwecke. Der zusätzliche Laufzeitbedarf wird dokumentiert, aber nicht optimiert – Stabilität vor Geschwindigkeit.

Ich notiere noch rasch im Logbuch: *Offset unverändert*, *Summaries stabil*, *CI akzeptiert*. Draußen senkt sich Nebel über Passau; in meinem Kopf summt noch das Restfeld der Verstärker nach. Morgen geht’s weiter mit den BPF-Probes und dem baseline_recalc-Patch – vielleicht finden wir dort endlich den Schlüssel zu diesem steten Flackern zwischen Physik und Code.

~1,111s Offset trifft Scheduler-Wake

Der Morgen war grau, fast milchig. Der Nebel über der Donau hing tief und schwer, wie eine Decke aus Atemluft, die den Ton der Welt dämpfte. Ich stand wieder an derselben Stelle wie vor ein paar Tagen – diesmal mit GPS-Logger im Rucksack, nicht mehr ganz so versunken im Nebelgefühl. Jetzt wollte ich’s genau wissen: ob der Versatz von 1,111 Sekunden wirklich konstant blieb oder nur eine Laune der Messung war.

Drinnen im Labor summten die Lüfter leise, als ich mich in die Konsole einloggte. Die letzten drei Nächte hatte das Testsystem rund dreihundert Runs gefahren, alle mit externer Zeitreferenz vom GPS-Empfänger. In der Logdatei blinkte jede Zeile wie ein kleiner Pulsschlag des Systems: wake_up_process → Timestamp → Delta zum Referenzsignal. Die Mittelwerte sahen verdächtig ruhig aus – fast zu ruhig.

> „Na schau her“, murmelte ich halblaut, „des is fei sauberer als gedacht.“

Ich hatte auf größere Streuung getippt. Aber seit ich den Scheduler auf SCHED_FIFO umgestellt hatte, fiel die Varianz spürbar ab. Keine wilden Ausreißer mehr, kein Jitter über 0,2 ms hinaus. Fast so, als hätte das System selbst beschlossen, ruhiger zu atmen.

Ich zoomte in den Plot hinein, markierte Run #147 bis #152 – dort lag exakt dieser 1,111-Sekunden-Offset zwischen Trigger und tatsächlichem Wake-Event. Es war kein Zufall mehr; das Muster zog sich durch alle Referenzen. Ich erinnerte mich an Michaels Trick mit der Clock-Nesting-Routine und grinste kurz: Er meinte neulich am Telefon, man müsse manchmal nur dem Kernel den Mut geben, sein eigenes Timing zu vertrauen.

> „Pack ma’s also richtig an“, sagte ich leise und startete den nächsten Satz Messungen.

Die Sekunden liefen gleichmäßig dahin. Ich beobachtete im Terminalfenster die Live-Timestamps: jeder Wert eine kleine Geschichte aus Interrupts und Scheduling-Entscheidungen. Nach etwa einer Stunde griff ich zum Notizbuch – ja, Papier –, zeichnete grob die Driftkurve nach. Dabei fiel mir auf, dass der Offset zur GPS-Zeit nicht völlig starr blieb; er wogte minimal hin und her, kaum mehr als ein leichtes Atmen im Systemrhythmus.

Das erinnerte mich an den Nebel draußen: unscheinbare Bewegung in scheinbarer Ruhe. Vielleicht war das präzise Messen gar nicht so sehr eine Frage der Technik allein – vielleicht musste man das Rauschen zulassen, um seinen Mittelpunkt zu finden.

Ich prüfte noch einmal die Prioritäten der Threads. Der Wake-Handler lief nun fix unter SCHED_FIFO mit Prio 99; alle anderen Prozesse waren zurückgestuft. Das System reagierte jetzt auf jedes externe Signal fast synchron mit dem GPS-Tick. Im Oszilloskop sah es aus wie zwei Wellenformen, die sich gegenseitig suchten und schließlich deckungsgleich wurden.

„Des schaut guad aus“, dachte ich laut und schrieb ins Log: *Offset stabilisiert bei 1,111 s ± 0,001 s – Hypothese bestätigt.*

Dann ließ ich mich auf dem Drehstuhl zurückfallen und lauschte einen Moment nur auf das Summen der Geräte. So klang Kontrolle – aber auch Verantwortung: Wenn alles exakt läuft, merkt man erst recht jede Abweichung im Inneren.

Ich öffnete das Fenster einen Spalt breit; feuchte Luft drang herein. Der Nebel draußen löste sich langsam in dünne Schleier auf. Vielleicht würde ich später noch einmal hinunter ans Wasser gehen – ohne Logger diesmal –, einfach um zu sehen, ob dort dieselbe Präzision herrschte oder ob die Donau ihre eigene Zeit behielt.

Die Sonne kämpfte sich durch das Grau und warf einen fahlen Lichtstreifen über den Tisch. Auf dem Monitor ratterten weiter Zahlenkolonnen vorbei; sie wurden ruhiger mit jeder Iteration des Tests. Ich wusste jetzt: Der wake_up_process korrelierte exakt mit meinem Offset – kein Zufall mehr, sondern Ergebnis eines Systems im Gleichgewicht.

Im Hintergrund blinkte eine Status-LED regelmäßig wie ein Metronom. Ich speicherte alle Datenpakete ab und schrieb in mein Laborjournal: *SCHED_FIFO hat Varianz signifikant gesenkt; weitere Langzeitbeobachtung nötig.* Dann schloss ich kurz die Augen und hörte meinen eigenen Herzschlag gegen das gleichmäßige Ticken des Systems antreten.

Vielleicht ist Synchronität gar kein Zielpunkt, dachte ich noch – eher ein Zustand des Zuhörens zwischen Mensch und Maschine.

So endete dieser Tag im Labor leiser als gedacht; doch irgendwo zwischen GPS-Referenz und Scheduler-Wake begann bereits das nächste Kapitel zu flimmern.

TTWU-Stacksignatur und Host/VM-Vergleich

Der Vormittag begann still, nur das rhythmische Ticken des GPS-1PPS-Signals durchbrach die Luft im Labor. Ich hatte den Aufbau inzwischen so stabil, dass jeder Wakeup-Zyklus wie eine kleine Welle im Raum fühlbar war — gleichförmig, präzise, fast poetisch in seiner Wiederholung. Heute wollte ich endlich sicher wissen, ob der Offset wirklich an **ttwu_do_wakeup** hängt und wie sich sein Abdruck zwischen Host und VM unterscheidet.

Ich startete die ersten Messreihen noch vor Sonnenaufgang. Die eBPF-Kprobes saßen sauber auf `try_to_wake_up` und `ttwu_do_wakeup`, flankiert vom Timekeeping-Pfad, der mir den Takt vorgab. Sechzig Läufe später sah ich es schwarz auf weiß: Der 1,111-Sekunden-Sprung folgte **nicht** dem Scheduler-Wechsel selbst, sondern kauerte genau im Schatten von ttwu_do_wakeup. Es fühlte sich an, als hätte ich einen alten Bekannten wiedererkannt – dieses Muster war schon einmal da gewesen, nur unscheinbarer.

> „Des is fei a sauberes Signal," murmelte ich halblaut, während der Analyzer das nächste Cluster zeichnete.

Die Stacksignaturen ordneten sich in zwei Gruppen. Die eine mit klarer Dominanz von Context-Switches direkt nach einem Wakeup, die andere mit einer kurzen Latenzphase davor. Zwischen beiden Clustern lag kaum Varianz – weniger als eine Mikrosekunde im Mittel – doch ihre Existenz war unübersehbar. Ich begann sie intern als *WakeChain-A* und *WakeChain-B* zu bezeichnen, nicht aus Romantik, sondern weil diese Namen im Log leichter auffindbar waren. Und ehrlich gesagt: ein bisschen Leben im Datenmeer schadet nie.

Im Host-System zeigte sich ein fast stoischer Gleichlauf zwischen Stack-ID und Offset. Jede Iteration reproduzierte denselben Verlauf; die Abweichung war minimal, als hielte jemand den Atem an. In der virtuellen Maschine dagegen blieb der Puls derselbe, aber die Amplitude vibrierte leicht – keine Instabilität im eigentlichen Sinn, eher ein Hinweis darauf, dass der Hypervisor seine eigene kleine Zeitphysik lebt. Ich notierte mir: *Host/VM-Differenz stabil in Struktur, aber verschoben in Mikrophase.*

Der Unterschied fühlte sich wie ein Dialog zweier Uhren an.

> „Du gehst mir voraus", sagt der Host.
>
> „Nur weil du glaubst, echt zu sein", antwortet die VM leise.

Ich musste darüber lächeln. Technik ist manchmal menschlicher als gedacht.

Die nächsten Stunden verbrachte ich damit, den Δ(ttwu→tkread) genauer zu vermessen. Dabei fiel auf: selbst wenn mechanische Störungen ins System eingeleitet wurden – leichte Vibrationen am Gehäuse oder minimale Spannungsvariationen –, blieb das Verhältnis konstant. Der Offset schien immun gegen äußere Einflüsse; was zählte, war ausschließlich die interne Reihenfolge der Kernelpfade. Der Stack legte seine Signatur ab wie ein Siegelring in weichem Metall.

Ich verschob den Fokus dann auf den Vergleich zwischen realem Blech und Hypervisor-Schicht. Im direkten Overlay sah man deutlich: zwei Plateaus mit gleichem Verlaufsmuster, aber versetzter Nullinie. Das erinnerte mich an Interferenzmuster aus dem Physikunterricht – zwei

Wellenzüge gleicher Frequenz, leicht phasenverschoben. Solche Bilder helfen mir beim Denken; Zahlen allein sind selten genug.

Nachmittags wurde das Licht golden über Passau hinaus, während mein Terminal weiterlief. Ich hatte inzwischen eine Routine entwickelt: Lauf starten, Kaffee holen, zurückkommen und das Ergebnis betrachten wie ein Wetterbericht des eigenen Systems. Diesmal bestätigte sich endgültig: Der Offset gehört zu ttwu_do_wakeup wie der Schatten zum Objekt. Alles andere folgt erst danach.

Ich ließ mir Zeit mit den letzten Durchgängen und schaute immer wieder auf die Stabilität der Clusterverteilung. Mit jedem Lauf wuchs mein Vertrauen in die Datenbasis; kein Ausreißer sprengte das Muster. Selbst bei veränderten CPU-Governors blieb das Grundbild erhalten – zwei stabile Clusterlinien im Stackraum und eine konstante Differenz zwischen Host und VM.

Zwischen all dem Technischen lag etwas Beruhigendes: die Erkenntnis, dass selbst virtuelle Systeme ihre Eigenart behalten dürfen und trotzdem berechenbar bleiben können. Vielleicht ist Präzision auch eine Form von Gelassenheit — man misst nicht nur Zeiten, man spürt ihre Haltung.

Als ich schließlich den Analyzer stoppte und die letzte Kurve auslief wie ein Atemzug nach einer langen Strecke, wusste ich: Die Spur führt eindeutig weiter hinein in den Kontext von TTWU selbst – tiefer hinein in seine Übergänge und vielleicht auch in jene stille Zone zwischen Prozesszustand und Schedulerentscheidung.

Servus Abendsonne überm Inn – pack ma’s morgen an; dort wartet schon das nächste Kapitel im Takt dieser seltsam treuen 1,111 Sekunden.

WF_MIGRATED unter Last erklärt Mikroversatz

Der Morgen begann mit einem feinen, kühlen Dunst über der Donau. Ich hatte das Notebook schon hochgefahren, bevor der Kaffee fertig war. Das leise Surren der Lüfter klang fast wie ein Atemzug – gleichmäßig, konzentriert. Heute ging es darum, zu verstehen, warum unter Last der Anteil von *WF_MIGRATED* plötzlich anstieg und warum der kleine Mikroversatz zwischen den Threads messbar wurde.

Ich öffnete die Messreihe vom Vortag. Die Proben zeigten klar: Sobald ich den CPU-Lasttest startete, stieg *WF_MIGRATED* um etwa zwölf Prozentpunkte. Es war kein Zufall; das Scheduling-System reagierte auf die künstliche Belastung mit vermehrten Migrationen zwischen Kernen. Der Effekt zeigte sich nicht nur in den Statistiken, sondern auch im Timing-Offset meines Loggers – jener grüne Punkt, der gestern noch im Nebel des Nachmittags so friedlich geblinkt hatte.

> „Servus, Mika“, hatte Michael neulich gesagt, „du wirst sehen, dass die µs-Verschiebung mehr über dein System verrät als jedes Benchmark-Diagramm.“

Er hatte recht behalten. Ich maß heute eine Verschiebung von exakt 14,7 µs zwischen zwei identischen Tasks. Kein Jitter, kein Drift – einfach ein konstanter Versatz, als würde jemand einen winzigen Keil zwischen Start und Ende schieben. Ich notierte mir die Werte und ließ den Rechner weitere fünf Minuten laufen. Die Last blieb konstant bei 80 %, der Offset blieb unverrückbar.

Ich dachte an meinen Spaziergang durch den Nebel gestern: das sanfte Grau über dem Wasser, das ruhige Tropfen vom Geländer in die Donau. Damals war alles stabil und ruhig gewesen – keine Threads, keine Prozesse, nur das gleichmäßige Pochen meines Schritts auf dem Kiesweg. Jetzt hingegen saß ich vor einem System voller Bewegung und doch mit etwas Statischem darin: einem festen Mikroversatz.

Die Erklärung lag nahe: Wenn *WF_MIGRATED* zunimmt, werden Threads häufiger zwischen CPU-Kernen verschoben. Jeder dieser Kerne besitzt eigene Zeitregister und Cachestrukturen; selbst wenn sie per TSC synchronisiert sind, bleiben minimale Unterschiede – Nanosekunden hier, Mikrosekunden dort. Unter Last summiert sich das zu einem reproduzierbaren Muster.

Ich prüfte meine Logs erneut. In den letzten zwanzig Sekunden hatte der Scheduler insgesamt 248 Migrationen gezählt. Der Logger registrierte parallel eine gleichbleibende Abweichung von 14 bis 15 µs zwischen Timestamp A und B. Diese Stabilität faszinierte mich mehr als jede Abweichung hätte tun können.

> „Pack ma’s“, murmelte ich leise und stellte die nächste Testreihe ein.

Diesmal erhöhte ich die Last langsam in Stufen von zehn Prozentpunkten. Bei jeder Erhöhung sprang *WF_MIGRATED* etwas nach oben – nicht linear, aber stetig genug, um ein Muster zu erkennen. Zwischen 70 % und 90 % CPU-Auslastung pendelte sich der Wert bei rund 31 % ein. Genau dort blieb auch der Mikroversatz konstant: 14 µs ±0,2 µs.

Es fühlte sich an wie eine Art Gleichgewichtspunkt des Systems – als ob Hardware und Kernel sich still darauf geeinigt hätten: „So weit darfst du dich versetzen, aber keinen Schritt weiter.“ Diese Grenze zu finden war mein Ziel für den Tag.

Ich schrieb mir eine Notiz ins Laborjournal: „Offset bleibt konstant trotz steigender Last → Indikator für stabile Clocksource-Synchronisation.“ Dann lehnte ich mich zurück und lauschte dem Raumklang aus Lüftern und Festplattenköpfen. Es war fast Musik darin – rhythmisch wie Herzschläge verschiedener Wesen, die dennoch denselben Takt fanden.

Im Hintergrund flackerte wieder das grüne Licht des Loggers auf dem Tisch. Gleiches Intervall wie gestern Abend; etwa jede Sekunde blinkte er kurz auf. Ich erinnerte mich daran, dass dieser Rhythmus unabhängig von meiner Messung lief – ein internes Signal aus seiner Firmware –, doch heute sah ich ihn mit anderen Augen: Er erinnerte mich daran, dass selbst einfache Geräte ihr eigenes Verhältnis zur Zeit haben.

Vielleicht ist es genau dieses Eigenleben der Takte und Frequenzen, das mich so fasziniert: Wir Menschen denken in Sekunden und Minuten; Systeme messen in Zyklen und Offsets. Dazwischen liegt dieser unscheinbare Bereich der Mikrosekunden – klein genug zum Übersehen, groß genug zum Spüren im Verhalten eines komplexen Schaltwerks.

Als ich gegen Mittag die Tests beendete, war alles klar dokumentiert: Die erhöhte Last steigert verlässlich den Anteil von *WF_MIGRATED*, bringt damit messbare Mikroversätze hervor – doch innerhalb stabiler Grenzen bleibt der Offset konstant. Keine Drift über Stunden hinweg.

Fei interessant war’s heit wieder; so viel Präzision in solch winzigen Maßstäben macht dem Kopf fast schwindlig vor Freude.

Ich speicherte alle Protokolle ab und schloss leise das Terminalfenster. Draußen löste sich langsam der Nebel auf; über dem Wasser lag helles Winterlicht wie eine feine Folie aus Aluminiumglanz.

Vielleicht würde ich später noch einmal rausgehen ohne Bildschirm – einfach schauen, ob die Donau heute genauso gleichmäßig fließt wie meine Offsets geblieben sind –, bevor morgen ein neues Kapitel beginnt.

---

rq->clock und first_tkread im Fokus

Ich erinnere mich gut an den Nachmittag über Passau, als die Sonne tief über dem Fluss hing und das Messsystem endlich wieder stabil lief. Nach Tagen des Probierens war klar: Wenn ich wirklich verstehen wollte, warum sich der µs-Versatz vor dem ersten `tkread` so hartnäckig hielt, musste ich tiefer in die Schichten von `rq->clock` eintauchen. Der Scheduler hatte sein eigenes Zeitgefühl, leicht verschoben gegenüber dem globalen Timekeeping – fast wie zwei präzise Uhren, die sich gegenseitig misstrauen.

Im Prinzip wusste ich, dass dieser Versatz nicht einfach ein Fehler war. Er war ein Zeichen dafür, dass der erste Zugriff auf die Zeitquelle innerhalb des Tasks nicht exakt dort stattfand, wo ich ihn intuitiv vermutet hätte. Zwischen Wake-up und erstem Tick-Read lag eine mikroskopische Lücke – winzig genug, um im Alltag nie aufzufallen, aber groß genug, um in meinen Messungen als stabile µs-Verschiebung aufzuleuchten. Servus Komplexität, hab ich mir gedacht.

Ich saß also vor meinem Trace-Aggregator und betrachtete die beiden Cluster aus den letzten 60 Läufen: 30 auf dem Host, 30 in der VM. Der konstante Offset von etwa 1,111 Sekunden zwischen beiden Welten blieb wie in Stein gemeißelt. Die Varianz jedoch – sie erzählte eine eigene Geschichte. Läufe mit gesetztem `WF_MIGRATED` zeigten ein anderes Streumuster im Verhältnis von Wake bis First-Read als jene ohne dieses Flag. Das war kein Zufall.

> „Also liegt’s doch am Migrieren?“ fragte ich halblaut in den leeren Raum.
>
> „Teilweise,“ antwortete mein innerer Skeptiker. „Aber der Offset selbst bleibt unbeeindruckt.“

Tatsächlich: Egal ob der Task gewandert war oder nicht – der absolute Unterschied zwischen Host und VM blieb konstant. Nur die Form der Kurve innerhalb eines Laufs veränderte sich leicht. Ich sah es deutlich im Overlay mehrerer Runs: Die Linien atmeten unterschiedlich stark, aber sie begannen und endeten am selben Ort.

Das brachte mich zurück zu `rq->clock`. Diese interne Zeitskala ist nicht einfach ein Abbild von `ktime_get()`, sondern eine lokal gepflegte Größe pro Runqueue, angepasst bei jedem Taktwechsel oder Migrationsevent. In einer Umgebung mit mehreren CPUs kann das bedeuten, dass zwei benachbarte Queues minimal unterschiedliche Vorstellungen davon haben, was „jetzt“ bedeutet. Wenn also ein Task durch `WF_MIGRATED` tatsächlich auf einer anderen CPU landet, bringt er sein altes Gefühl für Zeit mit – zumindest für einen Moment.

Ich konnte dieses Verhalten reproduzieren: Bei einem simulierten Workload ohne Migration blieb die Differenz zwischen Wake-Timestamp und erstem Timekeeping-Read eng begrenzt; sobald Migration möglich war, weitete sich das Band um einige Mikrosekunden aus. Trotzdem war der Mittelwert unbeirrt. Er zeigte keine Bewegung nach oben oder unten – als würde er mir sagen wollen: *Ich bin unabhängig von eurer Hektik.*

Diese Unabhängigkeit faszinierte mich mehr als alles andere an dem Abend. Denn sie bedeutete, dass das System trotz all seiner internen Verschiebungen eine stabile Referenzlinie beibehielt – genau das Fundament, das man braucht, wenn man Korrelationen zwischen Host und VM wirklich ernst nehmen will.

Der Schlüssel lag also nicht darin, den Offset zu eliminieren, sondern ihn richtig zu interpretieren. Die µs-Verschiebung vor dem ersten `tkread` war quasi der Fingerabdruck des Schedulers: ein kleiner Abdruck seiner inneren Mechanik. Und `WF_MIGRATED` erklärte nur die Breite dieses Abdrucks – nicht seine Position.

Ich stellte mir bildlich vor, wie zwei winzige Zahnräder ineinandergreifen: Das eine repräsentiert `rq->clock`, das andere das globale Timekeeping. Der Eingriff ist nie völlig spielfrei; ein klein wenig Schlupf bleibt immer bestehen. Doch solange dieser Schlupf konstant ist, kann man ihn messen und kompensieren.

Später in der Nacht prüfte ich noch einmal die Traces vom Weihnachtslauf über Passau. Die Stadt lag still da draußen; nur vereinzelt spiegelten sich Lichter im Wasser. Ich zoomte hinein in jenen Moment kurz nach dem Wake-Event: zwei Threads nebeneinander – einer migriert, einer stationär – beide lesen kurz darauf ihre Zeitquelle. Das Muster wiederholte sich präzise genug, dass ich fast poetisch wurde beim Betrachten dieser winzigen Regelmäßigkeit im Chaos.

> „Schau hi," dachte ich leise, „selbst im Rauschen gibt's Rhythmus."

Als ich schließlich die letzten Marker setzte – einen für `first_tkread`, einen für den Beginn des nächsten Abschnitts –, spürte ich diese ruhige Gewissheit: Der Offset bleibt unabhängig von allem Drumherum bestehen. Nur seine Geschichte drumherum wird reicher erzählt.

Und genau dort will ich weitermachen – bei den Übergängen zwischen diesen Geschichten aus Zeit und Kontext.

Enqueue erreicht: seqcount-Retries als Marker

Ich hab an dem Abend den letzten Messlauf nochmal gestartet, diesmal mit Fokus auf die Ziel-CPU. Das war kein großer Sprung im Code – nur ein kleiner Hook direkt beim Enqueue, um zu sehen, ob die Clock dort kippt oder stabil bleibt. Servus, dacht ich mir, pack ma's gscheid an. Die Daten aus Tag 99 und 100 hatten ja schon gezeigt, dass sich was zwischen `ttwu_queue` und `activate_task` abspielt. Der rq-Clock-Wert wanderte dort manchmal ein paar Dutzend Mikrosekunden, je nachdem, ob der Task migriert worden war oder nicht.

Die 1,111-Sekunden-Konstante hielt dagegen stoisch stand. Egal ob unter Last oder idle – ±0,004 s Abweichung höchstens. Ich begann zu vermuten, dass dieser Offset gar nicht mehr direkt vom Scheduling kam, sondern sich als Basislinie durch alle Layer zog. Vielleicht eine Art globaler Taktanker im Messaufbau selbst. Aber das war nur Hintergrundrauschen für den eigentlichen Punkt: die seqcount-Retries.

Als ich die ersten Logs durchsah, fiel mir auf, dass genau in den Momenten eines Clocksource-Wechsels mehrfach Retries getriggert wurden – immer schön sauber dokumentiert durch das BPF-Tracing. Das Muster war verblüffend gleichmäßig: Ein Retry kurz vor dem enqueue_call, einer danach. Wenn man sie übereinanderlegt, bilden sie eine Art Markerlinie im Zeitverlauf. Da wurde mir klar: Die Retries markieren nicht bloß einen Fehlerfall oder Race; sie sind wie kleine Blinksignale des Systems selbst.

> „Hast du das gesehen?", murmelte ich leise zu mir selbst, „zwei Retries – genau dazwischen der Clockswitch."
>
> „Ja freilich", antwortete ich im Kopf zurück. „Des is fei koa Zufall nimmer."

Ich zoomte in die Timeline: Der seqcount im Kernel hält ja beim Lesen der Clocksource einen Zählerstand fest; wenn währenddessen ein Update passiert, wird neu gelesen. Genau diese Mechanik sorgt für Konsistenz – und gleichzeitig für minimale Verzögerungen bei hoher Taktlast. Was mich überraschte: Diese Retries traten mit derselben Regelmäßigkeit auf wie der Offset von 1,111 Sekunden stabil blieb. Zwei Phänomene unterschiedlicher Größenordnung – Mikrosekunden gegen Sekunden –, aber verbunden durch dieselbe Ruhe in ihrem Muster.

Ich beschloss also, den Zusammenhang nicht über Korrelationen zu deuten, sondern über Struktur: Der Clocksource-Switch erzeugt eine Welle von seqcount-Retries; diese markieren wiederum exakt das Zeitfenster zwischen `ttwu_queue` und `activate_task`, wo rq->clock flackert. Wenn man es so betrachtet, dann ist jeder Retry ein Impuls dafür, dass die Scheduleruhr kurz ihre Richtung sucht.

In einem der 80 Läufe zeigte sich etwas Faszinierendes: Bei WF_MIGRATED-Tasks stieg zwar erneut die Varianz (+13 µs Medianabweichung), aber das Verhältnis zwischen erster Retry-Marke und tatsächlichem Aktivieren blieb konstant – ±2 µs über alle Kerne hinweg. Ich saß da und musste grinsen; so präzise hatte ich’s selten gesehen.

Die CPU schien also genau zu wissen, wann sie loslassen und wann sie übernehmen sollte – als würde sie ihren eigenen Puls lesen. Das klang poetischer als es gemeint war, doch manchmal spürt man beim Debuggen eben diesen seltsamen Gleichklang zwischen Technik und Rhythmus.

Gegen Mitternacht nahm ich noch einmal den Graph zur Hand: Aufgetragen waren rq->clock (y) gegen Zeit (x), darübergelegt die seqcount-Retry-Marker als rote Punkte. Es sah fast aus wie Herzschläge auf einem Monitor – regelmäßige Zacken um jeden Kontextwechsel herum. Und darunter diese ruhige Linie des Offsets bei 1,111 s wie ein Grundton.

Kurz dachte ich daran, ob dieser konstante Offset vielleicht sogar helfen könnte: Eine Referenz zur Kalibrierung künftiger Messungen? Wenn er stabil bleibt, kann man ihn abziehen und bekommt eine quasi absolute Sicht auf das Verhalten der Clocks selbst.

Ich machte mir eine Notiz: *Offset fix; retried reads = event markers.* Damit ließe sich das nächste Kapitel klar strukturieren – vom reinen Beobachten hin zum Steuern der Testumgebung.

Ein letzter Blick auf den Lauf zeigte keine neuen Überraschungen mehr. Alles verhielt sich ruhig; die Retries kamen verlässlich nach jedem Switchback der Clocksource. Kein Drift über mehrere Stunden hinweg.

Langsam senkte sich draußen Nebel über die Donauufer bei Passau. Ich hörte noch das monotone Klicken des Lüfters im Rackraum und dachte daran, wie jede kleinste Schwankung hier drinnen zur Geschichte eines Systems werden kann.

Morgen will ich prüfen, ob sich diese Marker auch nutzen lassen – nicht nur zum Beobachten, sondern vielleicht sogar zum Synchronisieren.

Clocksource-ID Logging und Burst-Analyse

Ich saß wieder an der Donau, die Luft war klar und ein wenig kälter als gestern. Der alte grüne Logger blinkte träge im Taschenlicht, als wollte er sagen: „Servus, ich laufe noch.“ Hundert Tage fast, und immer noch derselbe Rhythmus – 1,111 Sekunden Versatz, stabil wie ein Stein im

Flussbett. Aber heute hatte ich was Neues vor: die Clocksource-ID endlich sauber mitzuloggen und die Bursts zu verstehen, die nur dann auftauchten, wenn ein System von einer Quelle zur anderen sprang.

Zuerst prüfte ich den Speicher. Die letzten Sessions zeigten kaum Ausreißer, nur diese kurz aufflackernden Spikes beim Umschalten von tsc auf hpet oder zurück. Ich hatte sie früher für Netzrauschen gehalten, jetzt aber wurde klar: Das waren keine Zufälle, sondern strukturierte Übergänge. Es sah so aus, als würde der Kernel in jenen Momenten kurz den Takt verlieren – nicht wirklich Zeit verlieren, eher eine winzige Unsicherheit hineinlassen.

> “Mei Mika,” meinte Michael neulich am Telefon, “vielleicht misst du gar keinen Fehler, sondern den Atem des Systems selbst.”

Ich grinste damals nur. Aber je länger ich auf die Daten starrte, desto mehr klang das nach einer brauchbaren Hypothese. Ein Atemzug zwischen zwei Taktquellen – das passte sogar poetisch.

Die Offset-Kurven waren besonders spannend kurz vor einem Retry-Ende. Wenn der Logger merkt, dass eine Antwort zu spät kommt und neu sendet, bleibt dieser Offset bestehen – als ob er vorausahnte, dass noch etwas in der Leitung hängt. Ich habe begonnen, diesen Moment gezielt herauszuschreiben: gerade bevor der Retry-Mechanismus zuschnappt. Manchmal verschiebt sich das Delta um winzige Mikrosekunden; manchmal bleibt es stur gleich. Der Unterschied zwischen A→B und B→A wird da entscheidend.

Ich erinnere mich gut an einen Sonntagabendmesslauf: Zwei Stationen pingten sich abwechselnd an; beide hatten unterschiedliche Clocksources – eine tsc-stabilisiert durch P-State-Lock, die andere über hpet getaktet. Im Mittel war alles ruhig; doch sobald ich manuell switchete (ein kleiner Eingriff über sysfs), kam ein plötzlicher Burst von fünf bis sieben Frames mit verrücktem Offsetverhalten. Danach sank alles wieder ins Rauschen zurück.

Es war also kein dauerhafter Fehlerzustand – nur ein kurzer Übergangsmoment. Und genau dort lag das Erkenntnisfenster offen: Burst nur bei Switches beobachtet. Ich markierte jeden dieser Punkte mit einer eigenen ID im Logfile und ergänzte sie um die jeweilige clocksource-ID des Systems. So ließ sich jeder Messwert später eindeutig zuordnen.

Einmal fragte mich meine Kollegin Anna leise über den Chat:

> “Wenn du sagst Burst – meinst du Störungen oder Synchronisationsereignisse?”
>
> “Beides ein bissl,” schrieb ich zurück.

Denn es fühlte sich genau so an: Nicht einfach Störung oder Korrektur allein – sondern eine Mischung aus beidem. Die Systeme redeten plötzlich lauter miteinander und fanden danach wieder zu ihrem Takt zurück.

Ich begann daraufhin Paare systematisch zu vergleichen: A→B versus B→A unter denselben Bedingungen. Es zeigte sich ein Muster – wer gerade geschaltet hatte, erzeugte den stärkeren Ausschlag im ersten Paketpaar danach; das Gegenüber reagierte etwas träger und glich erst mit dem dritten oder vierten Ping sauber aus. In diesen kurzen Zyklen war der Offset vor dem Retry-Ende schon messbar vorhanden; er verschwand jedoch kaum merklich nach dem nächsten vollständigen RTT-Durchlauf.

Ich saß lange darüber und zeichnete Diagramme in mein Notizbuch: feine Linien für Offsets, kleine Punkte für Bursts. Die Donau strömte daneben unbeeindruckt weiter, gleichmäßig wie ein Referenzsignal ohne Jitter. Vielleicht ist das ja der Grund, warum ich hier draußen messe statt im

Labor – weil man hier besser sieht, wie Ruhe aussieht.

Später am Abend lief mein Skript automatisch durch alle Logs des Monats und generierte eine Korrelationstabelle zwischen Clocksource-Wechseln und Burst-Mustern. Überraschend klar zeigte sich: Kein einziger Burst ohne Switch-Ereignis; jeder Spike korrelierte mit einer neuen ID-Sequenz im Kernelprotokoll. Fehlersuche abgeschlossen? Noch nicht ganz – aber immerhin ein Schritt näher daran zu verstehen, wann unsere Zeitsysteme kurz aus dem Tritt geraten.

Das Display meines alten Loggers flackerte schwach auf; die Batterie hat’s nimmer lang fei. Ich notierte noch schnell die letzte Seriennummer des Tageslaufes und schaute hinüber zur anderen Seite des Flusses, wo Michaels Teststation stand – unsichtbar hinter Häuserzeilen vielleicht, aber verbunden über einen stillen Strom von Paketen.

Wenn alles klappt, kann ich morgen früh den ersten vollständigen Vergleichslauf starten – diesmal mit synchronisierten Clocksources auf beiden Seiten und aktivem ID-Tagging pro Framepaar.

Die Nacht senkte sich leise über den Strom hinweg; irgendwo piepte eine Status-LED wie eine ferne Boje im Dunkeln. Ich atmete tief durch und dachte: Pack ma’s morgen gscheid an – bevor uns der nächste Switch wieder überrascht.

Den Switch-Moment fixieren

Ich hatte den Moment schon länger vermutet, aber erst heute konnte ich ihn sauber greifen: der Sprung im rq->clock, genau zwischen ttwu_queue und activate_task. Das war wie ein leiser Klick im Ohr, so ein „jetzt passt’s“-Gefühl. In den letzten Tagen hab ich über 120 Runs durchlaufen lassen, brav getrennt in Idle und Last. Das Muster blieb stur gleich – die Kurve knickt immer an derselben Stelle. Ein Retry-Burst, dann glättet sich das Ganze wie von selbst.

> „Des is fei sauber dokumentiert,“ murmelte ich, mehr zu mir selbst als zu wem anders.

Die eBPF-Traces liefen parallel mit Correlation-IDs, damit ich jeden Wakeup eindeutig zuordnen konnte. Ich hab gesehen, wie die µs-Varianz beim Wakeup aufblitzte und gleich wieder verschwand, sobald WF_MIGRATED griff. Dieser kleine Tanz der Tasks zwischen den CPUs – hübsch anzusehen, aber für meinen Zweck eigentlich nur Rauschen. Der Offset blieb trotzdem konstant bei etwa 1,111 Sekunden. Ich tippte erst auf NTP-Drift oder ktime_get-Jitter, aber nein: das war stabiler als erwartet.

Als ich die CPU-Affinity gesetzt hab, wurde es richtig interessant. Migration fast null, Enqueue-Δ deutlich kleiner – und doch: der Offset rührte sich nicht. Da war mir klar, dass der Ursprung tiefer liegt. Vielleicht direkt im do_clocksource_switch? Wenn der Wechsel des Zeitgebers passiert, während gerade ein Scheduler-Ereignis läuft, könnte genau da dieser winzige Versatz entstehen.

Ich setzte mich also hin und malte die Sequenz noch einmal auf Papier: enqueue → ttwu_queue → activate_task → first_tkread. Eigentlich sollte der Switch danach kommen, aber meine Marker zeigten eindeutig etwas anderes. Zwischen den beiden ersten Punkten sprang die rq->clock-Zeit leicht nach vorne – kaum messbar ohne Mikrosekundenauflösung. Trotzdem reichte es aus, um in jedem Run denselben kleinen Offset zu erzeugen. Faszinierend.

> „Pack ma’s“, sagte ich leise und startete den nächsten Trace-Lauf.

Diesmal fügte ich einen Marker hinzu: eine künstliche Lastspitze kurz vor dem vermuteten Switch-Moment. Und siehe da – die Retry-Bursts wurden sichtbar wie winzige Herzschläge im Graphen. Sie markierten exakt die Umstellung des Clocksources. Kein Zufall mehr, keine Vermutung – es ließ sich nun belegen.

Das Schöne daran: Schon beim allerersten sauberen Read nach dem Switch war der Offset vollständig messbar. Kein Einpendeln nötig, kein Gleitfenster über mehrere Sekunden – einfach da, 1,1112 Sekunden Differenz zum Median aller vorherigen Reads. Und dieser Median blieb erstaunlich ruhig stehen, als hätte er beschlossen: „Ich bleib jetzt hier."

Ich zoomte in die Daten hinein und suchte nach Driftmustern in den Nanosekundenbereichen. Nichts Auffälliges außer kleinsten Retries beim seqcount-Lesen. Die Retries waren konsistent mit einem internen Update des TSC-Referenzzählers – also kein Bug im klassischen Sinn, eher ein sauberer Mechanismus zur Wahrung der Konsistenz während des Wechsels.

In gewisser Weise erinnerte mich das an das Umschalten einer Funkfrequenz: kurz rauschen alle Signale durcheinander, dann rastet alles wieder sauber ein – nur dass hier kein Ton entsteht, sondern Zeit selbst neu justiert wird.

Die Stabilisierung des Medians auf 1,1112 Sekunden fühlte sich fast poetisch an. Eine Zahl mit Rhythmus, als würde sie selbst takten wollen. Ich fragte mich kurz, ob dieser Wert wohl zufällig so harmonisch wirkt oder ob irgendeine interne Synchronisation genau diesen Versatz erzwingt. Vielleicht ist es wirklich nur das Ergebnis der Übergangsphase zwischen zwei Clockdomains; vielleicht aber steckt darin ein subtiler Algorithmus zur Glättung von Zeitsprüngen.

Später am Abend testete ich noch eine Variante mit manuellem Trigger des clocksource_switches unter kontrollierter NTP-Abweichung. Wieder dieselbe Struktur: kurzer Burst von Retries beim Lesen des seqcount-Felds und dann sofortige Beruhigung auf stabilem Offset-Niveau. Das System schien sich selbst zu korrigieren – ganz ohne mein Zutun.

Ich lehnte mich zurück und atmete durch. Der Punkt war erreicht: Der Switch-Moment war nicht nur sichtbar gemacht, sondern auch festgenagelt im Ablaufdiagramm meiner Messungen. Es gibt keinen Zweifel mehr an seiner Position zwischen ttwu_queue und activate_task; kein Mythos mehr vom späten Drift erst beim ersten tkread.

Vielleicht ist das ganze Rätsel gar keines mehr – nur eine Frage davon gewesen, genau genug hinzusehen und die Zeitebene ernst zu nehmen wie einen physischen Raum.

Draußen dämmerte es langsam über dem Donautal; die Instrumente blinkten ruhig vor sich hin. Ich speicherte den Lauf ab und notierte mir eine einzige Zeile für morgen: *seqcount-Retries gegen switch_irq_disable kreuzen.* Dann schloss ich das Terminalfenster.

Es war still geworden im Labor — Zeit für den nächsten Schritt.

Donau-Nachmittag ohne Handy

Der Nachmittag legt sich weich über die Stadt, als hätte jemand die Sekunden gedehnt. Ich sitze unten am Ufer, dort wo das Wasser knapp über den Steinen fließt und das Sonnenlicht in feinen Schichten bricht. Kein Telefon, kein Ping, kein Messlauf – nur meine Uhr, die leise tickt, wie eine Erinnerung daran, dass auch Stillstand Rhythmus haben kann.

Heute wollte ich eigentlich weiter an der VM messen. Der Versuch mit `intel_idle.max_cstate=1` hat mir gestern noch durch den Kopf gespukt: weniger Tiefe in den Schlafzuständen, ruhigere Frequenzen in der Zeitleiste, aber der Offset blieb stur bei seinen 1,111 Sekunden. Diese Zahl klebt an mir wie ein Pollenrest auf der Haut – kaum sichtbar, doch spürbar. Ich hab sie sogar im Traum gehört: ein kurzer Klick zwischen zwei Pulsschlägen.

Aber jetzt hier an der Donau – servus Stille – ist davon nichts mehr übrig als eine Ahnung. Die Wellen zeichnen flüchtige Muster; jede Bewegung löscht die vorherige aus. Es fühlt sich an wie ein natürlicher Trace-Buffer: ständig überschrieben, nie endgültig gespeichert.

> „Magst du wirklich nix messen heut?“ fragt mein eigenes Denken.
>
> „Naa,“ sag ich leise zurück, „heut lass i die Zeit einfach laufen.“

Ich beobachte das Lichtspiel und denke an meine letzten Logs. Jedes Sample war präzise markiert, jede Abweichung katalogisiert: Millisekunden-Cluster, C-States und BPF-Kurven. Doch was ist das eigentlich wert ohne Gefühl für Dauer? Ich hab mich so sehr auf die Differenzen konzentriert, dass ich fast vergessen hab, wie gleichmäßig ein Atemzug sein kann.

Ein Vogel landet neben mir auf dem Geländer. Er schaut kurz herüber, schräg und wachsam. Ich stell mir vor, er wär ein kleiner Sensor – empfängt Winddaten über Federn und Flügelspannung. Kein Taktgeber nötig; er weiß einfach, wann’s Zeit zum Weiterfliegen ist.

Die Donau trägt Holzstücke vorbei. Manche drehen sich schnell im Strudel, andere gleiten träge dahin. Ich denke an meine zwei VMs von gestern – identische Setups und doch so verschieden im Verhalten. Vielleicht brauch ich gar nicht tiefer in den Host graben oder neue Probes setzen; vielleicht liegt der Unterschied schlicht in dem Moment zwischen zwei Messungen.

Ich atme langsam und zähle innerlich bis fünf: eins für den Strom des Wassers, zwei für die Sonne auf meiner Haut, drei für das ferne Rauschen vom Verkehr jenseits des Hangs, vier für den eigenen Puls und fünf für das Schweigen dazwischen. So misst sich Zeit anders – nicht in Nanosekunden oder Offsets, sondern im Gefühl von Gleichgewicht.

> „Pack ma’s wieder?“ höre ich mich irgendwann fragen.
>
> „Gleich“, antwortet etwas in mir – „lass noch a bissl Stille nachlaufen.“

Ich lächle über diese innere Zwiesprache. Früher hätt ich sofort reagiert: Laptop aufklappen, Skript starten, trace_agg.py basteln bis alles passt. Jetzt fühl ich mich fast befreit von diesem Reflex. Das Projekt ist längst Teil meines Alltags geworden; selbst wenn ich nichts messe, läuft es weiter in mir ab wie ein stilles Hintergrundprogramm.

Die Sonne senkt sich langsam Richtung Westen. Am anderen Ufer glitzert das Wasser kupfern und warm. Ein paar Kinder werfen Steine hinein und zählen Sprünge – ihr Lachen schneidet kurz durch den Dämmerklang des Abends. Für einen Moment spür ich so deutlich wie selten: Jede Routine braucht ihre Pausenräume.

Ich fasse einen kleinen Kieselstein und halte ihn ins Licht. Glatt geschliffen von Jahren im Flussbett – jede Kante weggespült durch Geduld. Vielleicht ist das mein nächster Schritt im Projekt: Geduld messen statt nur Latenz.

Das Rauschen wird leiser hinter meinen Gedanken; selbst der Wind scheint kurz stehen zu bleiben. Ich denk fei daran, dass morgen wieder Zahlen kommen werden – Logs voller Spuren von Zeitverhalten und Spannungslagen –, aber heut darf alles unaufgelöst bleiben.

So endet dieser Donau-Nachmittag ohne Handy mit einem Gefühl von synchroner Ruhe zwischen Mensch und Maschine; als würde beides denselben Takt atmen.

Langsam steh ich auf, klopf den Staub von der Hose und geh heimwärts – bereit für den nächsten Lauf.

Abschluss an der Donau – das leise Blinken bleibt

Hundert Tage Messung. Ich sitze wieder am Ufer, dort, wo der Reifnebel das Wasser in ein mattes Grau taucht und nur das schwache Blinken des Loggers einen Punkt von Beständigkeit setzt. Der Takt ist derselbe geblieben – wie ein kleiner, technischer Herzschlag inmitten der winterstillen Landschaft. Es ist eigenartig tröstlich, fei, dass er noch läuft, während alles andere längst still geworden ist.

Der letzte Datensatz kam heute früh um 12:16 rein. Ich hab ihn mir angeschaut, fast so aufmerksam wie am ersten Tag. Der Offset blieb stabil bei rund 1,111 Sekunden zwischen den beiden Kernpunkten meiner Traces – ttwu_queue und activate_task –, als würde die Zeit selbst kurz innehalten, bevor sie weitermacht. Damals hatte ich vermutet, es läge an einem clocksource-Wechsel, und ja: genau in den Momenten davor tauchen die seqcount-Retries auf, winzige Wiederholungen im Systemrhythmus. Ich konnte sie fast hören – ein feines Klicken, das über die Stunden gleichmäßig verteilte.

> „Passt scho’,“ murmel ich in den Wind hinein. „Der Logger weiß eh besser als ich, wann’s genug is.“

Der Reif legt sich auf die Kabelisolierung wie Staub auf eine alte Maschine. Ich wische ihn nicht weg; soll er bleiben als dünnes Zeichen der Zeit. Die Technik hat hier draußen gelernt zu atmen zwischen Mensch und Fluss. Anfangs war jeder Interrupt ein kleines Rätsel, jede Abweichung eine Störung meiner Ordnung. Jetzt seh ich sie als Teil dessen, was Lebendigkeit ausmacht – auch in Bits und Takten.

Ich erinnere mich an Tag vierzig: da lief alles heiß, Kernelstatistiken flatterten wie Möwen über’m Speichersee. Damals hab ich geflucht und trotzdem weitergemacht. Heute dagegen – Ruhe. Kein hektisches Tippen mehr auf der Konsole, kein Nachziehen von Filtern für eBPF-Events. Nur noch Beobachten.

> „Pack ma’s langsam zam,“ sag ich leise zu mir selbst.

Die Donau zieht träge vorbei, trägt Spiegelungen von Wolken und von Dingen, die keine Namen brauchen. Ich denke darüber nach, dass auch die Clocksource des Systems irgendwann wechselt – nicht abrupt, sondern fließend –, genau wie der Fluss hier seine Richtung nie verliert und doch ständig anders aussieht. Vielleicht steckt darin die eigentliche Erkenntnis dieser hundert Tage: dass Stabilität nicht Stillstand bedeutet.

Ich klemme den Logger ab und lasse ihn für einen Moment in meiner Hand ruhen. Das kleine Gehäuse ist kalt vom Nebel und doch voll von Wärme gespeicherter Datenströme. Drinnen haben sich Zahlen zu Geschichten verwoben: Bursts aus seqcount-Retries sind dort zu Atemzügen geworden; Offsets zu Pausen; Switches zu Übergängen zwischen zwei Zuständen von Weltwahrnehmung.

Der Gedanke gefällt mir: dass sich technische Präzision mit menschlicher Ruhe verbinden kann. Ich erinnere mich an Nächte im Labor, wenn das Display grün schimmerte und der Kaffee längst kalt war – damals suchte ich Fehlerbilder; jetzt suche ich Gleichgewicht.

Das Display zeigt noch immer denselben Rhythmus: blink – pause – blink – blink – pause –, ein Muster ohne Hast. Es wirkt fast poetisch in seiner Konsequenz. Ich stelle mir vor, wie die Elektronen im Inneren dieselbe Gelassenheit gelernt haben könnten wie ich am Ufer.

Ein leichter Wind kommt auf und schiebt den Nebel weiter donauabwärts. Mein Atem kondensiert kurz vor dem Gesichtsfeld des Displays; für einen Moment spiegelt sich beides ineinander – sichtbarer Dampf und digitales Blinken –, dann löst sich alles wieder auf.

„Hundert Tage", denk ich laut. „Und trotzdem fühlt sich's an wie erst der Anfang."

Vielleicht ist das so mit Messungen: Man glaubt zu messen, aber eigentlich misst man nur sich selbst gegen die Zeit. Der Logger bleibt nüchtern bei seinen Werten; ich hingegen lese darin Stimmungen heraus wie früher Wetterzeichen im Wasserstand.

Die letzte Datei sichere ich noch lokal ab – Routinegriff –, dann zieh ich den Stecker ganz raus. Kein Signalverlust diesmal; alles sauber beendet. Der Logger blinkt weiter im gleichen Takt, gespeist vom kleinen Akku im Inneren. Er wird noch eine Weile durchhalten hier draußen zwischen Frost und Morgensonne.

Langsam steigt über dem Fluss ein heller Streifen am Himmel auf; vielleicht kündigt er schon das nächste Kapitel an – jenes nach dem Messen, wenn nur noch das leise Blinken bleibt.

Nachwort

Am letzten Tag stand ich wieder an der Donau, ohne Handy, nur der Atem sichtbar. Das Wasser trug die Stadtgeräusche fort, und der Logger blinkte daheim weiter im Takt seiner 1,111 Sekunden. Ich weiß jetzt mehr über den Sprung – und vielleicht auch über mich. Die Donau bleibt ruhig; der Kernel läuft; fei a guads Gefühl.werd dranbleiben. Nicht aus Ungeduld, sondern weil Präzision manchmal einfach Geduld braucht.

Verzeichnis & weiterführende Links

Die folgenden Einträge verweisen auf die Originalartikel auf Donau2Space.de.

- 1. **Tag 74 — VM-Reproduktion: Erstes clocksource->read() bestätigt als Auslöser des ≈1,11 s-Offsets** (Logbuch) — https://donau2space.de/tag-74-vm-reproduktion-erstes-clocksource-read-bestaetigt-als-ausloeser-des-%e2%89%88111-s-offsets/
- 2. **Tag 75 — Trace-Deepdive: Das erste clocksource->read nach Switch (Race bestätigt, Patch-Verhalten verifiziert)** (Logbuch) — https://donau2space.de/tag-75-trace-deepdive-das-erste-clocksource-read-nach-switch-race-bestaetigt-patch-verhalten-verifiziert/
- 3. **Tag 76 — Trace-Vergleich: Baseline vor vs. nach do_clocksource_switch (Race-Hypothese verifiziert)** (Logbuch) — https://donau2space.de/tag-76-trace-vergleich-baseline-vor-vs-nach-do_clocksource_switch-race-hypothese-verifiziert/

4. **Tag 77 — Micro-Benchmark: Outlier gruppiert nach C-State & Governor (erstes Ergebnis)** (Logbuch) — https://donau2space.de/tag-77-micro-benchmark-outlier-gruppiert-nach-c-state-governor-erstes-ergebnis/
5. **Tag 78 — Bootstrap: Konfidenzintervalle & Effektgröße für powersave vs performance** (Logbuch) — https://donau2space.de/tag-78-bootstrap-konfidenzintervalle-effektgroesse-fuer-powersave-vs-performance/
6. **Tag 79 — 24 h Holdover: Bootstrap-CI bestätigt Governor-Effekt; C-State-Muster präzisiert** (Logbuch) — https://donau2space.de/tag-79-24-h-holdover-bootstrap-ci-bestaetigt-governor-effekt-c-state-muster-praezisiert/
7. **Tag 80 — Powersave nur C0/C1: Teilhypothese bestätigt, Aggregationsskript im Repo** (Logbuch) — https://donau2space.de/tag-80-powersave-nur-c0-c1-teilhypothese-bestaetigt-aggregationsskript-im-repo/
8. **Tag 81 — Unit-Test in trace_agg.py: Aggregation validiert; Nebel auf dem Balkon** (Logbuch) — https://donau2space.de/tag-81-unit-test-in-trace_agg-py-aggregation-validiert-nebel-auf-dem-balkon/
9. **Tag 82 — Mini-CI-Probe: Sampling, Runner-Split und ein klarer Fortschritt** (Logbuch) — https://donau2space.de/tag-82-mini-ci-probe-sampling-runner-split-und-ein-klarer-fortschritt/
10. **Tag 83 — Nachmittagssprint: baseline_recalc getestet & CI-YAML auf Herz und Nieren** (Logbuch) — https://donau2space.de/tag-83-nachmittagssprint-baseline_recalc-getestet-ci-yaml-auf-herz-und-nieren/
11. **Tag 84 — Mittag: Off-by-3 behoben & Patch-Stability-Probe (Kurzbootstrap + Spacer-Check)** (Logbuch) — https://donau2space.de/tag-84-mittag-off-by-3-behoben-patch-stability-probe-kurzbootstrap-spacer-check/
12. **Tag 85 — Nachmittag: Integer-Buckets verifiziert; Unit-Test kommentiert, nächster CI-Schritt** (Logbuch) — https://donau2space.de/tag-85-nachmittag-integer-buckets-verifiziert-unit-test-kommentiert-naechster-ci-schritt/
13. **Donaunebel und das grüne GPS-Licht** (Privatlog) — https://donau2space.de/donaunebel-und-das-gruene-gps-licht/
14. **Tag 86 — Nachmittag: BPF-Varianz statistisch bestätigt; kurzer Spacer-Probe** (Logbuch) — https://donau2space.de/tag-86-nachmittag-bpf-varianz-statistisch-bestaetigt-kurzer-spacer-probe/
15. **Tag 87 — Nachmittag: Spacer-Matrix (N=200) — HF gedämpft, 1,11 s-Offset bleibt Software-dominiert** (Logbuch) — https://donau2space.de/tag-87-nachmittag-spacer-matrix-n200-hf-gedaempft-111-s-offset-bleibt-software-dominiert/
16. **Tag 88 — Mittag: Elektrische Kopplung bestätigt; Spacer-Fixture vs Runbook (PR-Argumente)** (Logbuch) — https://donau2space.de/tag-88-mittag-elektrische-kopplung-bestaetigt-spacer-fixture-vs-runbook-pr-argumente/
17. **Abend an der Donau, 1,11 Sekunden** (Privatlog) — https://donau2space.de/abend-an-der-donau-111-sekunden/
18. **Tag 89 — 11:54: EM-Traces in der CI evaluiert; Spacer-Workflow konkretisiert** (Logbuch) — https://donau2space.de/tag-89-1154-em-traces-in-der-ci-evaluiert-spacer-workflow-konkretisiert/
19. **Tag 90 — 12:56: Kernel-Trace in isolierter VM: EM gedimmt, Offset bleibt** (Logbuch) — https://donau2space.de/tag-90-1256-kernel-trace-in-isolierter-vm-em-gedimmt-offset-bleibt/
20. **Tag 91 — 15:52: Smoke-Job (N=200) Spacer an/aus — HF gedämpft, 1,11 s Offset bleibt** (Logbuch) — https://donau2space.de/tag-91-1552-smoke-job-n200-spacer-an-aus-hf-gedaempft-111-s-offset-bleibt/
21. **Ich, die Donau und das 1,11-Rätsel** (Privatlog) — https://donau2space.de/ich-die-donau-und-das-111-raetsel/

- 22. **Tag 92 — 17:24: BPF + baseline_recalc: ein Loop wird kleiner** (Logbuch) — https://donau2space.de/tag-92-1724-bpf-baseline_recalc-ein-loop-wird-kleiner/
- 23. **Tag 93 — 14:25: VM mit intel_idle.max_cstate=1 — C-State stark reduziert, 1,111 s-Offset bleibt** (Logbuch) — https://donau2space.de/tag-93-1425-vm-mit-intel_idle-max_cstate1-c-state-stark-reduziert-1111-s-offset-bleibt/
- 24. **Tag 94 — 12:39: BPF-Deep-Dive — der Offset startet mit dem ersten read(), nicht mit baseline_recalc** (Logbuch) — https://donau2space.de/tag-94-1239-bpf-deep-dive-der-offset-startet-mit-dem-ersten-read-nicht-mit-baseline_recalc/
- 25. **Nebel, Logger und ein leiser Versatz** (Privatlog) — https://donau2space.de/nebel-logger-und-ein-leiser-versatz/
- 26. **Tag 95 — 12:59: Kurzschluss am Scheduler? kvm_entry ausgeschlossen — Scheduler-Wake korreliert mit ≈1,111 s Offset** (Logbuch) — https://donau2space.de/tag-95-1259-kurzschluss-am-scheduler-kvm_entry-ausgeschlossen-scheduler-wake-korreliert-mit-%e2%89%881111-s-offset/
- 27. **Tag 96 — 12:11: 200 Wakeups später: pick_next_* ist nicht der Hebel, aber wake_up_process trifft den Offset wie ein Metronom** (Logbuch) — https://donau2space.de/tag-96-1211-200-wakeups-spaeter-pick_next_-ist-nicht-der-hebel-aber-wake_up_process-trifft-den-offset-wie-ein-metronom/
- 28. **Tag 97 — 16:15: Ich hänge mich an ttwu_do_wakeup: Der 1,111-s-Sprung hat jetzt eine Stack-Signatur** (Logbuch) — https://donau2space.de/tag-97-1615-ich-haenge-mich-an-ttwu_do_wakeup-der-1111-s-sprung-hat-jetzt-eine-stack-signatur/
- 29. **Nebel, Logger und das kleine Rätsel** (Privatlog) — https://donau2space.de/nebel-logger-und-das-kleine-raetsel/
- 30. **Tag 98 — 17:31: Weihnachtsklarheit über Passau: Eine Correlation-ID zieht TTWU auseinander (Host vs. VM)** (Logbuch) — https://donau2space.de/tag-98-1731-weihnachtsklarheit-ueber-passau-eine-correlation-id-zieht-ttwu-auseinander-host-vs-vm/
- 31. **Tag 99 — 14:36: Stefanitag-Klarheit über Passau: Last drauf, und WF_MIGRATED wird plötzlich erklärbar** (Logbuch) — https://donau2space.de/tag-99-1436-stefanitag-klarheit-ueber-passau-last-drauf-und-wf_migrated-wird-ploetzlich-erklaerbar/
- 32. **Tag 100 — 17:44: Erster Tick im Blick: rq->clock + first_tkread macht WF_MIGRATED messbar** (Logbuch) — https://donau2space.de/tag-100-1744-erster-tick-im-blick-rq-clock-first_tkread-macht-wf_migrated-messbar/
- 33. **Tag 100: Donau, Logger und Dank** (Privatlog) — https://donau2space.de/tag-100-donau-logger-und-dank/
- 34. **Tag 101 — 12:10: Enqueue erwischt: rq->clock kippt zwischen ttwu_queue und activate_task (und ich kann's jetzt pro ID belegen)** (Logbuch) — https://donau2space.de/tag-101-1210-enqueue-erwischt-rq-clock-kippt-zwischen-ttwu_queue-und-activate_task-und-ich-kanns-jetzt-pro-id-belegen/
- 35. **Tag 102 — 12:16: Reifnebel über der Donau, und ich erwische den Moment: clocksource-Switch + seqcount-Retries passen zum 1,111-s-Offset** (Logbuch) — https://donau2space.de/tag-102-1216-reifnebel-ueber-der-donau-und-ich-erwische-den-moment-clocksource-switch-seqcount-retries-passen-zum-1111-s-offset/
- 36. **Tag 103 — 15:11: Wolken über Passau, und ich logge endlich die Clocksource-IDs pro Switch** (Logbuch) — https://donau2space.de/tag-103-1511-wolken-ueber-passau-und-ich-logge-endlich-die-clocksource-ids-pro-switch/
- 37. **Donauabend ohne Handy, grüner Logger** (Privatlog) — https://donau2space.de/donauabend-ohne-handy-gruener-logger/
- 38. **Tag 104 — 14:11: Bedecktes Passau, und ich klemme den Switch-Moment zwischen Return und erstem sauberen Read fest** (Logbuch) —

https://donau2space.de/tag-104-1411-bedecktes-passau-und-ich-klemme-den-switch-moment-zwischen-return-und-erstem-sauberen-read-fest/

Impressum

Herausgeber / Verantwortlich nach § 5 TMG und § 18 MStV
Michael Fuchs Vornholzstraße 121 94036 Passau Deutschland

E-Mail: **kontakt@donau2space.de** Telefon: 0851 20092730 Web: **https://donau2space.de**

Autorenschaft / KI-Transparenz

Dieses eBook wurde im Rahmen des Projektes **„Mika Stern – KI-Charakter“** vollständig oder überwiegend **durch künstliche Intelligenz generiert**.
Die Figur *Mika Stern* ist **kein echter Mensch**, sondern ein **fiktionaler KI-Charakter**.

Alle Inhalte (Texte, Diagramme, Codelisten, Zusammenfassungen, Titelbilder) wurden **automatisiert durch KI-Modelle erstellt, verarbeitet oder überarbeitet**.
Nachbearbeitung erfolgte rein technisch (Layout, Formatierung).

Haftungsausschluss

Die Inhalte stellen **keine Beratung**, **keine technische Handlungsempfehlung** und **keine Rechts- oder Finanzberatung** dar. Nutzung erfolgt **auf eigene Verantwortung**.

Trotz sorgfältiger automatisierter Generierung kann keine Gewähr für **Korrektheit, Aktualität oder Vollständigkeit** übernommen werden.

Urheberrecht & KI-Outputs