Optimizing Idle Power of HPC Systems: Practical Insights and Methods

Thomas Ilsche, Sebastian Schrader, Robert Schöne
ZIH, CIDS, TU Dresden, 01062 Dresden, Germany
thomas.ilsche@tu-dresden.de, sebastian.schrader@tu-dresden.de, robert.schoene@tu-dresden.de

***Abstract*—Energy costs are a critical consideration for operating High-Performance Computing (HPC) systems, with significant efforts dedicated to reducing the energy expenditure of active computations. However, compute nodes of HPC systems also spend a non-negligible amount of time idling, i.e., without performing any useful work. Optimizing idle power consumption presents an impactful and potentially more accessible opportunity for practical energy savings. This paper surveys factors that influence the idle power consumption of HPC systems. We employed various monitoring tools and power measurements in a structured practical approach to optimize idle power consumption during the installation phase of an HPC system. The results highlight the effectiveness of targeted idle power management strategies, demonstrating compound savings of approximately 90 W per node, and 57 kW for the full system in idle. This thoroughly discussed example can serve as a blueprint for similar optimizations in other HPC environments.**

***Index Terms*—High-Performance Computing, Idle power, Energy efficiency, Power management**

I. Introduction

Improving energy efficiency has been a primary target for High-Performance Computing (HPC) systems research over the recent years. This is driven by the cost of energy, environmental impact, and limited supply of power. Much of the research is focused on reducing the energy costs associated with computation, which represents the most significant portion of the overall energy expenditure. However, idle times, often overlooked in HPC systems, also contribute substantially to overall cost. Idle periods occur during transitions to more capable systems or simply over holidays when user job submissions decrease, reducing system utilization. Additionally, resource management systems need to reserve compute nodes for jobs spanning large parts of a system, leaving the nodes idle until the large job starts. While backfilling — scheduling smaller jobs to use these idle periods — is commonly used to improve utilization, it has practical limitations and remains an active topic of research [1], [2]. Moreover, planned and unplanned maintenance phases can leave compute nodes idle. For example, an inaccessible parallel file system can prevent the execution of jobs even when the compute nodes are functional.

Conventional server systems, e.g., web or database servers, operate on continuous utilization levels. During phases of low utilization, they process fewer requests, but rarely none at all. In contrast, nodes of HPC systems can be completely idle, perform no effective work at all during periods that may last from minutes to hours. This distinction in operational behavior influences the potential energy savings that are achievable by optimized configuration.

Compute jobs within HPC systems often include blocking communication phases, in particular due to load imbalances, that can be implemented as idle [3]. However, there is a latency/power trace-off when using idle mechanisms during communication phases. Moreover, this does not always apply to full compute nodes. On systems that allow node-sharing, nodes may also be partially utilized by jobs.

This paper focuses on effectively using processor idle states (C-states) rather than system suspend states (S-states) for managing power consumption of fully idle HPC system nodes. C-states, which manage power at the processor level, can be controlled transparently by the operating system. In contrast, S-states, which affect the entire node, require explicit control and external triggers or timers for resuming. The Slurm workload manager provides integrated mechanisms for suspending and resuming idle nodes [4]. Due to the substantial latencies, this is performed with certain delays and at configurable rates. While suspend has much higher energy saving potential than idle states, it can impose operational issues. Resuming a node reliably can be a practical challenge, e.g., due to network drivers and connections to parallel file systems resuming unreliably. In addition, due to the transition costs for suspend strategies, utilizing C-states efficiency is still relevant.

While it is unfeasible to generalize HPC operation, the aforementioned arguments motivate out focus on idle-power optimization. This paper discusses methods to analyze and optimize the use of C-states on fully-idle compute nodes to achieve optimal energy savings. This follows the optimizations performed on a 630-node CPU cluster at TU Dresden.

II. Related Work

Sundriyal et al. [5] leverage Linux suspend states to reduce the power consumption of idle nodes. They implement a strategy based on measuring the power consumption and using timed suspends and are therefore independent of a batch system. While they report energy savings of up to 87 %, the impact on computation efficiency from wake-up latencies is not described. Mämmelä et al. [6] describe energy aware scheduling approaches that combine different general scheduling algorithms with shutting down nodes for extended idle duration. They report a reduction of energy consumption by 6 % to 16 % with no significant increase in wait times.

Minartz et al. [7] discuss power saving modes for HPC systems and report that "For performance (measurement) reasons C-states are disabled on most HPC installations." We believe that now, 13 years after this publication, using C-states is common practice for HPC systems, but could not find a survey to confirm this. Minartz et al. also report energy savings from reducing the core frequency between 6 % and 11 %.

Gobriel et al. [8] describe an approach to increase idle times in the presence of background network traffic. They implement a prototype in a wireless network card that buffers broadcast messages, dispatching them in bursts. This strategy mitigates the impact of background noise, facilitating longer uninterrupted idle phases. They focus on mobile devices and report an achieved energy gain of 23 % and 41 % for wired and wireless environments respectively. To avoid introducing latency to critical communication, e.g., VoIP sessions, they use a heuristic to differentiate background traffic from active traffic, ensuring that the latter is not delayed by bulk processing.

III. Impacting Factors on Idle Power Consumption

ACPI [9] describes several measures to lower power consumption on idling systems. They can be implemented on processor level and still guarantee the system to be responsive to interrupts. On the one hand, performance states (P-states) slow down computation via Dynamic Voltage and Frequency Scaling (DVFS). On the other hand, power states (C-states) stop the execution of instructions to lower the power consumption until a certain event (e.g., timer, interrupt) occurs. To do so, the operating system calls a privileged instruction (e.g., `HLT` and `MWAIT` on x86). Depending on the definition of the requested state, the hardware disables (parts of) the core (if all threads of said core request a state). On Intel systems, typical core C-states are C1, C1E, and C6 [10, Table 2-12], which imply clock gating, clock gating+DVFS, and power gating, respectively. Please note, while C1E is usually requested per core, it is *"generally achieved at the package granularity"* [10].

When all cores reside in a deep C-state, there is almost no activity on the whole processor, e.g., no core requests data from the shared last level cache or the integrated memory controller iMC. Hence, non-core components can be disabled as well in this scenario. This is called a package C-state. Typical package C-states on Intel processors include PC2 and PC6 [10, Table 2-13]. PC6 can power gate non-core components or take them to retention voltage, or disable the iMC, setting the main memory to self-refresh. Since disabled processors cannot participate in communication with external devices or other processors, Intel introduced the PC2 state where external requests are handled, but still offcore-components can be disabled as well as cores.

Whenever a single core awakes, the non-core resources of its processor have to be enabled in addition to said core and, moreover, other processors might switch to PC2 in order to answer cache coherence requests. Therefore, the number of interrupts on any core defines the activity of all processors and consequently the power consumption on an idling system. To limit the number of interrupts of the operating system, Linux offers several configuration options. In the past, the regular scheduling-clock interrupts, that allow the scheduler to switch between processes running on a CPU, was a major source of interrupts that prevented efficient long-term idling. Nowadays, the default of `CONFIG_NO_HZ_IDLE` [11, Timers, NO_HZ] disables the scheduler ticks on idle CPUs. In addition Linux also provides options to lower reasons for regular kernel interrupts (e.g., `kcompactd` [11, Physical Memory], `khugepaged`[11, Transparent Hugepage Support], and rcu [12]). These enable administrators to extend potential idle phases of cores and processors, reducing idle power consumption.

On HPC systems there is another benefit of reducing interrupts: OS noise. In bulk synchronous parallel applications, a short interrupt of one local computation may slow down the entire parallel computation at synchronization points [13]. Therefore interrupts can have an amplified performance impact. The performance variations from non-deterministic interrupts are also problematic for performance measurements. Hence, processor cores used for computation should not be preempted while other (dedicated) cores handle interrupts. Linux supports this with the `CONFIG_NO_HZ_FULL` and `CONFIG_RCU_NOCB_CPU` flags [14] and the `irqaffinity` and `isolcpus` boot parameters [11, The kernel's command-line parameters]. While the methods to investigate and optimize OS noise and idle interrupts overlap, this work focuses on the latter.

We can distinguish two categories: software-visible noise and hardware noise. A specific infrastructure in the Linux Kernel can monitor *software-visible noise*. However, this `osnoise` tracer [13] uses an active workload rather than idle to detect software-visible noise caused by *"NMIs, IRQs, SoftIRQs, and any other system thread"* [11, OSNOISE Tracer]. In the context of idling systems, interrupts will have a CPU exit its idle state and work on a given task before it can return to idle. The occurrence of such work can be monitored with *ftrace* [11, Function Tracer] and tracepoints that provide a default instrumentation of the kernel [11, Event Tracing]. With this infrastructure, analysts can use different events, e.g., `nmi:nmi_handler` and `irq:irq_handler_entry`, to locate their occurrence and relate them to tasks. Several tools use tracepoint events, e.g., `powertop` via `libtracefs` or `lo2s` [15], to gather information on interrupts that indicate software visible noise. *Hardware noise* is handled by hardware components without the participation of the OS. This can be found, for example, in loosing cycles when the hardware serves System Management Interrupts, or when a PCIe device would use direct memory access to read data from main memory without the participation of a CPU. While the former could be detected with a tight computation loop as done by the `hwlat` tracer [11, Hardware Latency Detector], the latter is even harder to detect since no CPU is involved. Nevertheless, model specific registers [16, 15.5.4.2][17] and hardware performance monitoring units [16, 20.3.1.2] can be used to count the number of cycles package C-states were used or whether PCIe devices were actively communicating with the processor[1].

[1]e.g., UNC_IIO_* events on https://perfmon-events.intel.com/spxeon.html

IV. Idle Power Optimization on a CPU Cluster

The CPU cluster *Barnard* at TU Dresden consists of 630 compute nodes, each equipped with two Intel Xeon Platinum 8470 "Sapphire Rapids" processors. Each node contains 104 processor cores and 512 GiB DRAM. Hyper-Threading is enabled for a total of 208 hardware threads (logical CPUs as per the OS). The processor has a base frequency of 2.0 GHz, a maximum turbo frequency of 3.8 GHz, and a minimum frequency of 0.8 GHz. The operating system is configured to use the two available core C-states, C1 and C6 as per `/sys/devices/system/cpu/cpu0/cpuidle`. The compute nodes use the `acpi-cpufreq` idle driver. At the time of investigation, the system was running Red Hat Enterprise Linux release 8.7 with a Linux kernel version 4.18.0-425.19.2.el8_7.x86_64.

A. Power Measurement Capabilities

Barnard uses continuous power monitoring at two levels. For each node, the Baseboard Management Controllers (BMC) provide instantaneous power readouts via Redfish at a 1 Sa/s rate. Additionally, the data center infrastructure measures the input power for pairs of racks and one individual rack. This measurement uses Janitza UMG 96RM power analyzers and is read at 5 Sa/s using Modbus. The devices internally samples at a much higher rate to achieve the accuracy class of 0.5 %. Both measurements are processed by MetricQ infrastructure [18] and stored at full resolution.

While the processors also provide RAPL energy counters [19], they are of limited use for idle optimizations. Their accuracy at the idle point is not well understood. Non-linear effects due to interactions among energy saving mechanisms of the processor, voltage regulators, and other components in the system limit the meaningfulness of the CPU measurement domain. The actual optimization goal is to reduce the *system* power consumption. Moreover, on this system, RAPL is only available in-band and therefore requires active measurements that can perturb the idle nodes.

B. Quantifying Power Consumption for different C-States

To characterize the range of node power consumption, we consider a single Barnard node in several scenarios:

- FIRESTARTER, a tool designed to maximize power consumption [20], running on all hardware threads,
- FIRESTARTER running on a single hardware thread, with the rest of the hardware threads idle, all C-states enabled,
- the node runs in its normal idle configuration, except that for one CPU the core C-state C6 is disabled, which also prevents the system from entering deep package C-states,
- the normal idle configuration, core C6 enabled for all CPUs with no workload.

We run the four scenarios at the highest frequency possible using the `performance` governor and enabled turbo boost as well as the lowest frequency (800 MHz) using the `powersave` governor with disabled turbo boost. Note that these measurement were taken after the optimizations described in Section IV-E and on a single node.

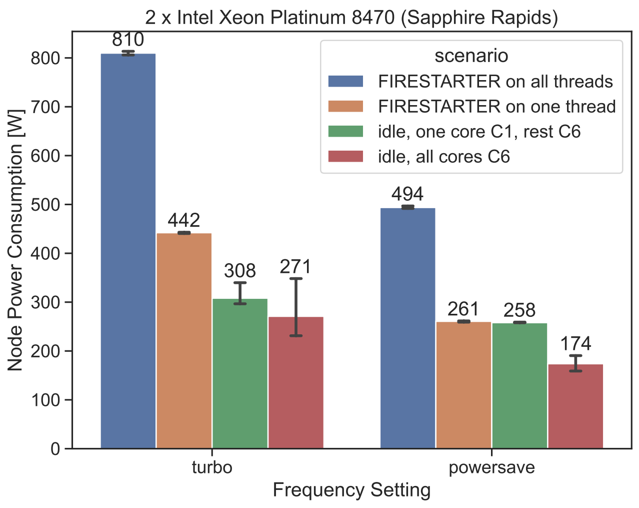

Figure 1: Power consumption on one node for different workload, C-state, and frequency configurations. Bars and labels show average power over a duration of 10 min, whiskers show the 5 and 95 percentiles.

Figure 1 shows the node power consumption in those configurations. Notably, with only a single active thread — one core out of 104 — the system consumes more than half of its maximum power. This high consumption for a single active thread applies to both frequency settings individually. Despite these numbers, and the only active core running at the highest possible turbo frequency, the power consumption of the core itself is likely relatively small. The large differences between the configurations where all but one thread are idle — ranging from 174 W to 442 W — can only be explained with the consumption of shared (non-core) resources. For instance, at turbo frequencies, a single active thread increases the node power consumption by 134 W compared to the configuration with a single core in a shallow idle state (C1). We suspect this difference to originate from the automatically regulated uncore frequency, which can be reduced even at shallow core/package C-states. Contrary, at the manually enforced lowest core frequency, the power consumption with a single active thread is almost equal to one thread in C1. This hints to a clockdown of the uncore frequency, which could be regulated based on workload demands [21] or core frequencies [22].

For all scenarios, including full idle, the power consumption is reduced when applying the `powersave` governor. This is unusual insofar as package C-states C1E and above automatically reduce frequency and voltage in addition to clock/power gating the cores. However, even though in the idle configuration C6 is *enabled* on all cores, the cores are waking up regularly and performing short bursts of work. In this configuration, the active times, where the frequency governor is relevant, are significant enough to cause this impact on power consumption. This observation is consistent with the high variance in power samples for the fully idle case especially with turbo frequencies, but also with the powersave governor.

On earlier systems, i.e., Intel's Skylake processor generation, we observe better package C-state residency and our measurements indicate that the full idle configuration is independent of the frequency governor. However, other, even earlier, related work [7] reported the idle power consumption to depend on the core frequency.

C. Core Frequency in Idle

In a first step to reduce the power consumption of fully idle nodes, the vendor installed Slurm epilog/prolog scripts. Whenever a job is scheduled on a node, the *performance* governor is activated as well as turbo frequencies. When the job finishes, turbo is disabled and the *powersave* governor is applied. Since nodes can be shared by multiple jobs, the prolog script uses a lock-file to detect whether another job is still running on the node.

With these scripts, the long-term average node power in idle decreased from 262.1 W to 184.0 W. Please note that the numbers differ from Figure 1, because here the average over all nodes are presented as opposed to a single node; Values vary significantly across nodes.

The `ondemand` and `schedutil` governors[2] select frequencies based on the current load. Using them independent of the job state of a node would also reduce the power consumption in idle. However, their use implies a latency to increase the frequency for full utilization and comes with an additional per-CPU overhead for calculating the desired frequency regularly, both undesirable in an HPC context.

D. Further Analysis

By observing idle power consumption of the racks, we noticed repetitive patterns in the power consumption as shown in Figure 2. Even at the sub-second level, the power of different rack-pairs showed a very strong correlation. To some extend, the node-level measurements also exhibited these patterns, but with a high degree over of overlapping noise from the less accurate instantaneous power measurements. This observation shows that there is still substantial potential for reducing idle power. The highly synchronized nature of the pattern indicated a cause that is either driven by local interrupts based on a global time or a interrupt to all nodes from an external event.

For a local investigation of idle activity on the node, we used PowerTOP. While the results presented here give hints for further optimization, please note that due to variations and limitation to a single node there is a significant degree of uncertainty in the values. PowerTOP reports package C-state utilizations of 60.9 % PC6 and 14.7 % PC2 with the exact same values for both processor packages. Figure 3 shows that, with few exceptions, the core C6 residencies are reported as 100.0 % or very close to it. Core residencies for C3 and C7 are listed, but always exactly 0 %.

Table I lists the most impactful sources of wake-ups. The `tick_sched_timer` has the highest event rate, but represents generic scheduling activity that offers no direct leverage for optimization. The `[rcu_sched]` process could possibly be optimized by tuning the RCU kernel boot parameters [12], but due to the complexity of this configuration and the possibility of side-effects, we did not follow up on this.

Arithmetically, the other sources each account only for a small fraction of <4 % of the total wake-ups. Nevertheless, `net_rx(softirq)` and `ipoib_reap_ah` (related to infiniband) indicate an impact of network activity, which is consistent with the observation of synchronized power patterns.

[2]https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

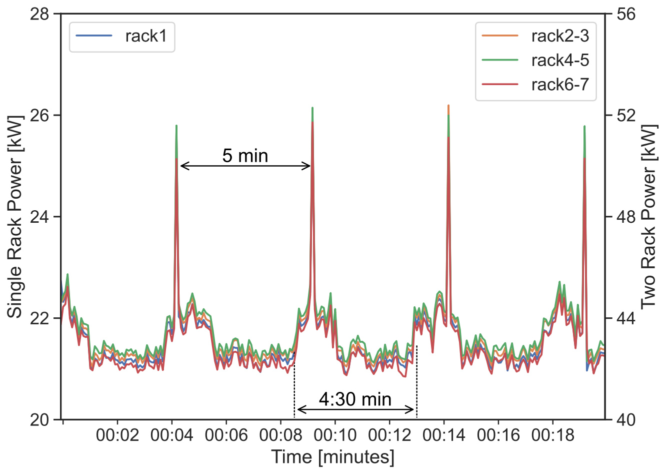

Figure 2: Power consumption of Barnard measured at rack input, averaged to 5 s bins.

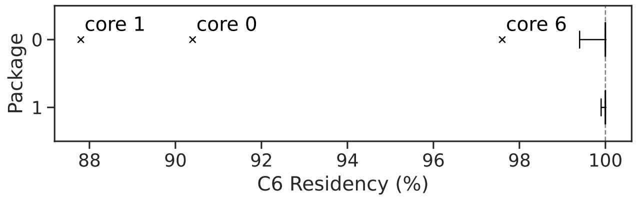

Figure 3: Boxplot with outliers of core C6 residency according to PowerTOP on Barnard before further optimization.

Table I: "Top 10 Power Consumers" as per PowerTOP, collected for 12 min. Total 131.4 wakeup/s.

Usage	Events/s	Category	Description
0.1 %	32.0	Timer	`tick_sched_timer`
0.0 %	8.7	Process	`[rcu_sched]`
0.0 %	4.8	Interrupt	`net_rx(softirq)`
0.0 %	4.8	kWork	`fb_flashcursor`
0.0 %	4.1	Process	`[xfsaild/dm-0]`
0.1 %	2.3	kWork	`ipoib_reap_ah`
0.0 %	2.3	Interrupt	`tasklet(softirq)`
0.2 %	1.0	Process	`powertop`
0.0 %	1.5	Process	`NetworkManager -no-daemon`
0.0 %	1.4	Interrupt	`mlx5_async63@pci`

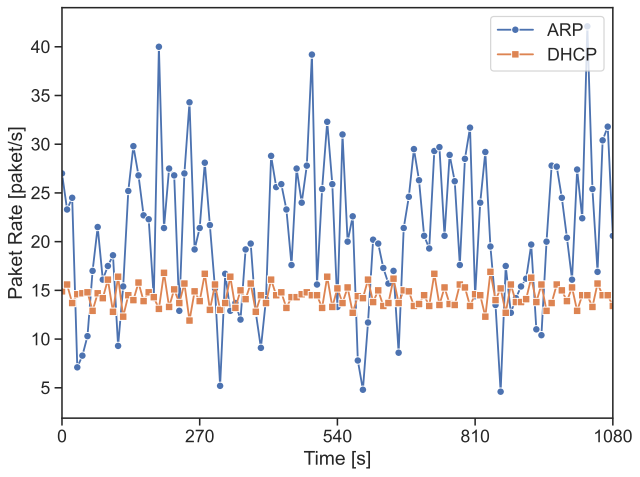

Figure 4: Packet rates exported from Wireshark, 10 s bins.

E. Optimizing Background Network Traffic

Following up on the suspicion of network activity having an impact on idle power consumption, we first confirmed that the nodes received ≈40 packets/s[3] on the Ethernet device. For further analysis, we captured a packet trace using `tcpdump`, which contained mainly ARP and DHCP broadcast packets[4]. The rate of ARP packets, shown in Figure 4, follows a pattern with a period of 270 s, matching the power consumption pattern revealed in Figure 2. However, the DHCP packet rate appears to be relatively constant with some noise. A cross-correlation between the packet rates and rack power consumption (both resampled on 1 s intervals) show a clear temporal connection with no delay for both ARP and DHCP packet rates.

These observations justified further analysis of packet sources and optimization in order to reduce idle power consumption. The DHCP packets were identified as discover queries originating from the Baseboard Management Controllers (BMC). The BMCs tried to configure an unused network interface as a result of a regression introduced during the early deployment stages. Since no answer was given, the requests were repeated every 64 s. After disabling DHCP on this BMC network interface, the idle power consumption was reduced from 184.0 W to 177.6 W per node.

In general, large layer 2 networks can experience significant background ARP (Address Resolution Protocol) traffic due to the need for nodes to resolve IP addresses to MAC addresses [23]. ARP uses Ethernet broadcasts that need to be processed by all nodes of a broadcast domain.[5] First, we reduced the amount of clients by fully deactivating unnecessary interfaces on the BMC. Secondly, we identified our Cumulus Linux based routers as the source of the periodic ARP packets. Cumulus Linux employs a `neighmgrd` daemon that tries to continuously refresh all neighbor entries within $1/4$ of the Linux kernel's base neighbor timeout We change the timeout[6] to drastically increase the interval from 270 s to 3570 s. Other network operating systems have similar mechanisms. When increasing the ARP/NDP timeout, the MAC aging-time on switches should also be adapted to ensure that is greater than the ARP/NDP timeout, in our case to 6 h. Since the HPC network changes less often than classical data center networks, the long timeouts have a low potential for problems. However, it does require some effort to consistently reconfigure a complex network with different generations of network components. Deactivating several interfaces reduced the node idle-power from 177.1 W[7] to 173.3 W, increasing the interval further reduced it to 172.2 W.

ARP/NDP traffic could be further reduced by leveraging the ARP/NDP suppression capability of Ethernet VPN (EVPN) [24]. Currently we employ EVPN only at our core routers at the top of a fat-tree topology. Extending the EVPN domain to more switches towards the edge of the network would limit the scope of the neighbor entry refresh to locally attached nodes instead of the whole network and allow suppressing ARP/NDP requests by end hosts directly by the closest switch. Unfortunately this was not feasible in our case.

F. Node Health Check

To further identify sources of activity on the idle compute nodes, we used `lo2s` [15]. This monitoring tool creates a timeline of various events, samples, and metrics in an OTF2 trace, which can be visualized with Vampir. In the system monitoring mode, `lo2s` records all scheduled activity on all CPUs, i.e, which task was executed at what time on which hardware thread. In addition, `lo2s` collects OS tracepoint events, in particular at `power::cpu_idle`. This tracepoint is triggered whenever the operating systems tells a hardware thread to enter an idle state, which idle state was selected by the idle governor, and when the idle state exited. Finally, the power measurements that have been collected externally via MetricQ are embedded in the trace (see Section IV-A). This trace comprises extensive information of idle activity while retaining the time dynamics.

To enable effective monitoring of idle with minimal perturbation, `lo2s` uses the `perf_event` monitoring infrastructure. The Linux kernel itself collects all events, without expensive switches to userspace monitoring code. There is also the possibility to include RAPL counters in the trace, however that requires to regularly read them with a userspace monitoring thread, imposing perturbation.

With `lo2s` we were able to immediately identify the activity running in 5 min intervals that first appeared in Figure 2 as the Node Health Check (NHC)[8] scripts. They are triggered by

[3]by evaluating `/sys/class/net/*/statistics/rx_packets`

[4]The initial `tcpdump` trace was created with the default promiscuous mode that includes packets not normally received by the node — in this case additional ICMPv6ND_NS packets. To avoid that `-no-promiscuous-mode` should be used for such an analysis.

[5]The IPv6 Neighbor Discovery Protocol (NDP) does not suffer from this problem due its use of solicited node multicast addresses instead of broadcasts.

[6]`net.ipv[4,6].neigh.default.base_reachable_time_ms = 14280000`

[7]The optimization described in Section IV-F was applied after DHCP and before ARP optimization. However, for readability reasons we continue with network related optimizations in this paper.

[8]https://github.com/mej/nhc

Table II: Average idle power consumption per node in various stages of optimization.

Initial State	`powersave` Governor	Disabled DHCP	NHC 15 min	ARP Reduction	Further ARP Reduction
262.1 W	184.0 W	177.6 W	177.1 W	173.3 W	172.2 W

Slurm in regular intervals for idle nodes to identify and mark faulty nodes. The runtime of each invocation varies, but can take up to 2.5 s. Note that the NHC processes were also included in the full PowerTOP "Software Info", but since each invocation has a different PID and it has very few "Wakeup/s", it is not listed in the top 10 power consumers. Instead of the effort to optimize the bash scripts, we chose to increase the interval to 15 min. As a trade-off this setting retains the functionality but further reduces the long-term idle power consumption from 177.6 W to 177.1 W per node.

G. Final State and Outlook

After the optimizations discussed in Section IV-E and Section IV-F, the package C6 residency increased from ≈60.9 % to ≈74.3 %[9] corresponding to an 11.8 W decrease of average idle node power. The nodes still show a significant number of intermittently scheduled kernel tasks across all cores as well as additional wake-up events on CPU 0 and 1. While this indicates, that there is still substantial potential for improvement, we could not identify further feasible worthwhile optimizations. Based on the PowerTOP descriptions `fb_flashcursor` and `NetworkManager -no-daemon`, we tried disabling the blinking of the framebuffer cursor[10] and the NetworkManager daemon, respectively. Neither led to a measurable change in power consumption on a single node test, thus no further change was deployed. Table II summarizes the power consumption after each optimization step. Since the optimizations affect each other, the relative improvements of each step depends on the order.

V. Conclusion and Future Work

This paper covers idle power of compute nodes: It discusses general factors and follows a exemplary optimization of HPC nodes. For the analysis, we show how tools such as PowerTOP and `lo2s` provide a structured way to identify opportunities for optimization. We identified three significant optimizations:

1) Reducing the core frequency explicitly in idle had a substantial impact on this system, contrary to previous experiences.
2) Reducing background network broadcasts lowered the rate of interrupts and extended sleep times.
3) Increasing the interval between idle maintenance tasks (Node Health Checks) further reduced average node idle power consumption.

In total, the three optimizations reduced the idle power by 89.9 W per node or 34 % overall. This constitutes a significant saving in operational costs of the system.

[9]Residency values according to two PowerTOP samples collected over 12 min on one node each.

[10]`echo 0 > /sys/class/graphics/fbcon/cursor_blink`

Besides the monitoring tools, we utilized external power measurements at node and rack level both for identifying disruptive activity and quantifying the impact of optimizations. The analysis of correlation between rack power and network event rates has shown, that a high temporal resolution, even beyond 1 Sa/s, can be useful in practice. During the installation phase, we were able to regularly observe the entire cluster in idle. For optimizations during production, where usually only individual nodes are idle, high-quality node-level power measurements would be beneficial.

While performing idle power analyses and optimizations during the acceptance test phase of an HPC system is a strong foundation for efficient operation, there is a substantial risk of regressions: Network and software configuration changes as well as system software or firmware updates can introduce regressions in idle power consumption. Due to variations and noise in measurements, it is much more challenging to detect inefficiencies of individual idle nodes. Possible opportunities for full-system idle analysis could be maintenance phases or partial outages, e.g. file system is unavailable but compute nodes are functional.

Overall, we argue, that monitoring, analyzing, and optimizing idle power consumption is an important part of the energy-efficient operation of HPC systems, alongside improving system utilization and application energy-efficiency. By incorporating these optimizations and maintaining vigilant monitoring, HPC system operators can achieve substantial energy savings, ensuring both operational cost-effectiveness and sustainability.

Acknowledgments

We sincerely thank the dedicated HPC operations team at TU Dresden, especially Danny Rotscher and Ulf Markwardt as well as the the technical team from Eviden, particularly Michael Hoehl, for their collaboration on the optimizations on the Barnard system.

References

[1] N. A. Simakov, M. D. Innus, M. D. Jones, R. L. DeLeon, J. P. White, S. M. Gallo, A. K. Patra, and T. R. Furlani, *A Slurm Simulator: Implementation and Parametric Analysis.* Springer International Publishing, Dec. 2017, pp. 197–217. DOI:10.1007/978-3-319-72971-8_10

[2] A. Wilkinson, J. Jones, H. Richardson, T. Dykes, and U.-U. Haus, *A Fast Simulator to Enable HPC Scheduling Strategy Comparisons.* Springer Nature Switzerland, 2023, pp. 320–333. DOI:10.1007/978-3-031-40843-4_24

[3] M. Knobloch, B. Mohr, and T. Minartz, "Determine energy-saving potential in wait-states of large-scale parallel programs," *Computer Science - Research and Development*, vol. 27, no. 4, pp. 255–263, Aug. 2011. DOI:10.1007/s00450-011-0196-7

[4] SchedMD, "Slurm power saving guide," https://slurm.schedmd.com/power_save.html.

[5] V. Sundriyal and M. Sosonkina, "Reducing idle power consumption in high performance systems," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI).* IEEE, Dec. 2017. DOI:10.1109/csci.2017.283

[6] O. Mämmelä, M. Majanen, R. Basmadjian, H. De Meer, A. Giesler, and W. Homberg, “Energy-aware job scheduler for high-performance computing,” *Computer Science - Research and Development*, vol. 27, no. 4, pp. 265–275, Aug. 2011. DOI:10.1007/s00450-011-0189-6

[7] T. Minartz, T. Ludwig, M. Knobloch, and B. Mohr, “Managing hardware power saving modes for high performance computing,” in *2011 International Green Computing Conference and Workshops*. IEEE, Jul. 2011. DOI:10.1109/igcc.2011.6008581

[8] S. Gobriel, C. Maciocco, and T.-Y. C. Tai, “Long idle: Making idle networks quiet for platform energy-efficiency,” in *2010 Fifth International Conference on Systems and Networks Communications*. IEEE, Aug. 2010. DOI:10.1109/icsnc.2010.60

[9] United EFI, Inc., “Advanced configuration and power interface (ACPI) specification,” 2016. [Online]. Available: https://www.uefi.org/sites/default/files/resources/ACPI_6_1.pdf

[10] W. S. Corey Gough, Ian Steiner, “Energy efficient servers - blueprints for data center optimization,” 2015. DOI:10.1007/978-1-4302-6638-9

[11] The kernel development community, “The linux kernel documentation,” (accessed 2024-06-26). [Online]. Available: https://www.kernel.org/doc/html/latest/

[12] P. E. McKenney, D. Eggemann, and R. Randhawa, “Improving energy efficiency on asymmetric multiprocessing systems,” Personal rdrop page of Paul E. McKenney (IBM Linux Technology Center), http://www.rdrop.com/users/paulmck/techreports/AMPenergy.2013.03.07a.pdf, IBM Linux Technology Center and ARM Ltd. Cambridge, Tech. Rep., 2013.

[13] D. B. de Oliveira, D. Casini, and T. Cucinotta, “Operating system noise in the linux kernel,” *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 196–207, Jan. 2023. DOI:10.1109/tc.2022.3187351

[14] P. McKenney, *Kernel configuration parameters for RCU*, Eklektix, Inc., 2019, (accessed 2024-06-26). [Online]. Available: https://lwn.net/Articles/777214/

[15] T. Ilsche, R. Schöne, M. Bielert, A. Gocht, and D. Hackenberg, “lo2s — multi-core system and application performance analysis for Linux,” in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 801–804. DOI:10.1109/CLUSTER.2017.116

[16] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 3B: System Programming Guide, Part 2*, Jun. 2024, (accessed 2024-06-26). [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/671427

[17] ——, *Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 4: Model-Specific Registers*, Jun. 2024, (accessed 2023-11-17). [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/671098

[18] T. Ilsche, D. Hackenberg, R. Schöne, M. Bielert, F. Höpfner, and W. E. Nagel, “MetricQ: A scalable infrastructure for processing high-resolution time series data,” in *2019 IEEE/ACM Industry/University Joint International Workshop on Data-center Automation, Analytics, and Control (DAAC)*, 2019, pp. 7–12. DOI:10.1109/DAAC49578.2019.00007

[19] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, “Power-management architecture of the intel microarchitecture code-named sandy bridge,” *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 3 2012. DOI:10.1109/MM.2012.12

[20] D. Hackenberg, R. Oldenburg, D. Molka, and R. Schöne, “Introducing FIRESTARTER: A processor stress test utility,” in *2013 International Green Computing Conference Proceedings*. IEEE, Jun. 2013. DOI:10.1109/igcc.2013.6604507

[21] R. Schöne, T. Ilsche, M. Bielert, A. Gocht, and D. Hackenberg, “Energy efficiency features of the intel skylake-sp processor and their impact on performance,” in *2019 International Conference on High Performance Computing & Simulation (HPCS)*, 2019, pp. 399–406. DOI:10.1109/HPCS48598.2019.9188239

[22] R. Schöne, M. Velten, D. Hackenberg, and T. Ilsche, “Energy efficiency features of the intel alder lake architecture,” in *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 95–106. DOI:10.1145/3629526.3645040

[23] K. Elmeleegy and A. L. Cox, “Etherproxy: Scaling ethernet by suppressing broadcast traffic,” in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 1584–1592. DOI:10.1109/INFCOM.2009.5062076

[24] J. Rabadan, S. Sathappan, K. Nagaraj, G. Hankins, and T. King, “Operational aspects of proxy ARP/ND in ethernet virtual private networks,” Internet Requests for Comments, RFC Editor, RFC 9161, January 2022.