Raport
Instytutu Automatyki i Informatyki Stosowanej
Politechniki Warszawskiej

Opracowanie i realizacja systemu do identyfikacji modeli obciążenia serwerów

Piotr Arabas, Michał Karpowicz, Ewa Niewiadomska-Szynkiewicz

E-mail: parabas@elka.pw.edu.pl, mkarpowicz@elka.pw.edu.pl ens@ia.pw.edu.pl

Raport wykonany w ramach projektu badawczego NCN, numer 2015/17/B/ST6/01885

Warszawa, marzec 2017

SPIS TREŚCI

1. Wprowadzenie

Raport jest poświęcony zagadnieniom energooszczędnych systemów teleinformatycznych. Uwaga koncentruje się na budowie modeli obciążeń serwerów obliczeniowych. Modele te będą wykorzystane w systemie zarządzania obliczeniami, którego celem jest efektywne wykorzystanie zasobów energetycznych centrów przetwarzania danych, a konkretnie zmniejszenie zużycia energii przez te jednostki obliczeniowe wchodzące w skład tych centrów.

Postęp wielu dziedzin nauki, m.in. fizyki i chemii, zależy od rozwoju nowatorskich energooszczędnych technologii teleinformatycznych umożliwiających przetwarzanie wielkich ilości danych. Według danych statystycznych obciążenie serwerów w centrach przetwarzania danych (data center), udostępniających usługi chmury obliczeniowej (cloud computing), kształtuje się na poziomie 10-50% ich możliwości [2]. Utrzymywanie zapasów mocy obliczeniowej związane jest z koniecznością zapewnienia gwarancji jakości oraz odporności usług na nagłe zmiany szybkości napływania zadań. Taka polityka sterownia wymusza jednak nadmierne zużycie mocy elektrycznej i pośrednio przyczynia się do wzmożonej emisji CO_2 do atmosfery. Zapotrzebowanie na energię jest również jednym z dominujących ograniczeń tempa rozwoju systemów HPC (high performance computing), przy obecnie stosowanych rozwiązaniach klaster exaflopowy pobierałby gigawaty mocy.

Nowoczesne elementy obliczeniowe, tj. procesory, układy pamięci i karty sieciowe, mogą pracować w zmiennych trybach (www.acpi.info) charakteryzujących się różnym poziomem zużycia energii. Przełączanie trybów pracy oraz odczyt sprzętowych wskaźników efektywności operacji umożliwiają intensywnie rozwijane abstrakcyjne interfejsy programistyczne (API) oraz sondy (moduły jądra) monitorujące pracę systemu operacyjnego. Istotnie, nowoczesne energooszczędne technologie obliczeniowe otwierają nowe możliwości prowadzenia badań nad założonymi procesami obliczeniowymi oraz strukturami sterowania klastrami serwerów.

W wyniku prac realizowanych w okresie styczeń-marzec 2017 opracowane zostały podstawy teoretyczne oraz wykonany system do identyfikacji eksperymentalnej modeli obciążeń i wydajności energetycznej jednostek obliczeniowych. Modele te będą wykorzystane do konstrukcji energooszczędnych algorytmów szeregowania zadań i alokacji zasobów oraz regulatorów pracy serwerów. Przewiduje się wykorzystanie mechanizmów usypiania i wybudzania urządzeń oraz skalowania napięcia i częstotliwości (DVFS) procesorów. Efektem końcowym będzie koncepcja i realizacja systemu komputerowego do energooszczędnego sterowania obciążeniem klastra oraz szybkością pracy jednostek obliczeniowych.

1.1. Modele serwerów aplikacji

Wykonane badania eksperymentalne miały na celu zdobycie nowej wiedzy o stochastycznym charakterze procesów obliczeniowych (serwerów aplikacji i maszyn wirtualnych) w złożonym systemie obliczeniowym. W badaniach wykorzystane zostanły narzędzia profilowania jądra systemu operacyjnego Linux umożliwiające odczyt rejestrów procesorów i metryk systemowych (RAPL, IPCM, MSR, *perf_events*) z wysoką częstotliwością próbkowania. Na potrzeby zadania identyfikacji zaimplementowane zostały generatory obciążenia serwerów (m.in. na bazie mgen, JMeter, benchIT, stress, itp.) umożliwiające wytwarzania strumieni zadanej charakterystyce widmowej (w miarę możliwości trwale pobudzających).

1.2. Modele obciążenia serwerów

Badania eksperymentalne zostały przeprowadzone z wykorzystaniem odpowiednio zaprojektowanych generatorów obciążeń (o zadanej charakterystyce widmowej) umożliwiających obserwacje dominujących zależności przyczynowo-skutkowych (pomiędzy licznikami operacji obliczeniowych) na poziomie jądra systemu operacyjnego (Linux). Wykorzystując metodykę wnioskowania statystycznego zidentyfikowane modele procesów zostały poddane analizie w dziedzinie czasu i częstotliwości. Na jej podstawie skonstruowane zostaną algorytmy predykcji oraz obserwatory stanu, które posłużą następnie do weryfikacji hipotez dotyczących własności stochastycznych procesów obliczeniowych. Ze względu na złożoność architektury sprzętowej (m.in. pipelining, multilevel cache) oraz interakcji wielu elementów obliczeniowych (m.in. rdzenie CPU, hierarchia jednostek pamięci) w warunkach losowych (szeregowanie zadań procesora przez system operacyjny) problem prognozowania obciążenia serwera wymaga wykorzystania zaawansowanych technik i nowatorskiego podejścia teoretycznego.

2. Modelowanie poboru mocy

2.1. Modele statyczne

Dla konstrukcji algorytmów sterujących konieczne jest posiadanie adekwatnego modelu poboru mocy. Model ten, powinien oddawać całkowity pobór mocy przez urządzenie, co wymaga wykonania dość skomplikowanych pomiarów w trakcie jego identyfikacji. W przypadku sterowania grupą heterogenicznych serwerów (np. farmą serwerów bazodanowych czy klastrem obliczeniowym), powinno się zidentyfikować modele dla wszystkich z nich, co wymaga wielokrotnego zestawienia eksperymentu pomiarowego z użyciem zewnętrznego miernika mocy.

Rozwiązaniem tego problemu może być wykorzystanie informacji udostępnianych przez nowsze wersje procesorów Intela za pomocą rejestrów msr [4]. W ten sposób, stosunkowo małym nakładem środków, można zdobyć szczegółowe informacje o poborze mocy przez procesor. Można przypuszczać, że pobór mocy przez cały serwer jest w pewien sposób skorelowany z poborem mocy przez procesor, który jest przecież głównym elementem maszyny. Wiele jednak wskazuje, że postać tej zależności może być różna w przypadku wykonywania przez serwer różnych zadań, angażujących w różnym stopniu zasoby i komponenty składowe. Celem przedstawionych w tym rozdziale prac było sprawdzenie czy korelacja taka istnieje i w przypadku pozytywnej odpowiedzi na to pytanie podjęcie próby wyznaczenia stosownego modelu.

Scenariusze pomiarowe

W celu określenia zależności miedzy mocą pobieraną przez serwer i mocą raportowaną przez procesor za pośrednictwem rejestrów msr konieczne jest przeprowadzenie jednoczesnych pomiarów obu wartości. Przyjęto, że badania będą prowadzone dla dwóch scenariuszy:

1 serwer obliczeniowy, wykonujący intensywne operacje arytmetyczne,

2 serwer przekodowujący strumień wideo.

Scenariusz pierwszy odpowiada sytuacji, gdy jedynym istotnie obciążonym elementem serwera jest procesor. Różnice w poborze mocy wynikają w tym przypadku głównie z możliwości dopasowania częstotliwości zegara procesora (czy też częstotliwości z jakimi taktowane są poszczególne bloki procesora) do natężenia wykonywanych zadań. W systemie Linux możliwe jest to dzięki działaniu w jądrze odpowiedniego sterownika (tzw. frequency governor) [18].

Drugi scenariusz stanowi próbę uwzględnienia obciązenia wprowadzanego przez operacje sieciowe – serwer przekodowujący strumienie wideo odbiera je na wybranym interfejsie sieciowym i wysyła przez drugi. Ponieważ przekodowanie strumienia wideo jest stosunkowo złożonym zadaniem obliczeniowym, to wydajność procesora stanowi wąskie gardło, tzn. prawdopodobnie nie jest możliwe pełne obciążenie interfejsów sieciowych ruchem. Tym niemniej stanowią one drugi, po procesorze i pamięci najbardziej obciążony komponent systemu.

Stanowisko pomiarowe

Na stanowisko pomiarowe składało się pięć komputerów klasy PC wyposażonych w procesory Intel i7, 8GB RAM, czteroportowe karty sieciowe Ethernet 1 Gb/s (Broadcom BCM5719), oraz typowy licznik energii elektrycznej z funkcją zdalnego odczytu .Użycie licznika energii zamiast wyspecjalizowanych mierników ogranicza rozdzielczość pomiarów do 1W, przy czym mogą one być wykonywane co około 8 s. Częstotliwość taka pozwala na zebranie dostatecznej liczy próbek przy czasie trwania eksperymentu rzędu pojedynczych minut. Pozostałe cztery komputery były wykorzystywane jako źródła i odbiorniki strumienia wideo.

Odczyt rejestrów wykonano za pomocą zmodyfikowanego programu power_gov (https://software.intel.com/en-us/articles/intel-power-governor). Modyfikacja polegała na umożliwieniu zapisywania pomiarów w pamięci, przez co zmniejszono obciążenie procesora i uczyniono eksperyment bardziej wiarygodnym. Po zakończeniu eksperymentu pomiarowego i powiadomieniu o tym programu power_gov poprzez wysłanie odpowiedniego sygnału, możliwe było zapisanie pomiarów na dysk. Wielką zaletą wykonywania pomiaru mocy za pomocą rejestrów msr jest możliwość ich częstego odczytu – w analizowanym przypadku robiono to co 10 ms.

W scenariuszu 1 konieczne jest obciążanie procesora zadaniem obliczeniowym o zmiennej intensywności. Zrealizowano to za pomocą zmodyfikowanego programu stress (http://people.seas.harvard.edu/\textasciitilde apw/stress/), który może wykonywać jedno lub więcej równoległych zadań arytmetycznych, pozwalając na obciążanie kolejno wszystkich rdzeni procesora. Wadą programu jest fakt, iż obciąża on procesor zawsze w tym samym stopniu. Problem ten rozwiązano wstawiając w wewnętrznej pętli obliczeniowej instrukcje opóźniające wstrzymujące wykonanie wątku.

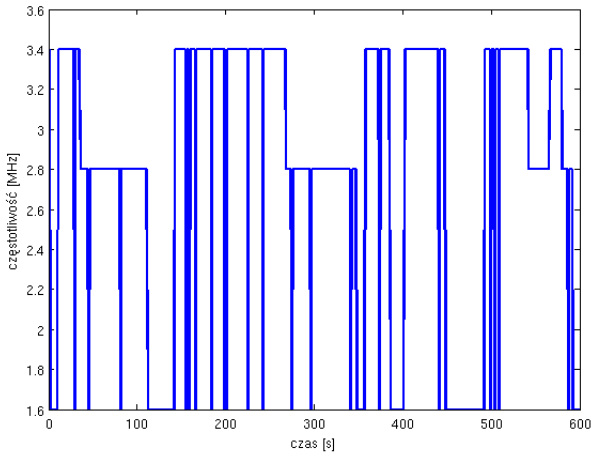
W przypadku scenariusza 2 do generowania strumienia wideo i następnie do jego przekodowania wykorzystano program mencoder (http://www.mplayerhq.hu). Wysyłanie i odbiór strumienia przez sieć był możliwy dzięki wykorzystaniu polecenia netcat. Użycie pary programów netcat pozwoliło zestawić rodzaj tunelu między komputerami biorącymi udział w eksperymencie. Konsekwencją takiego sposobu przekazywania ruchu jest pominięcie części mechanizmów związanych z przetwarzaniem nagłówków, co może wpływać na wykorzystanie mechanizmów przyspieszających karty (offloading).

Eksperymenty pomiarowe: serwer obliczeniowy

W celu zdjęcia charakterystyki odwzorowującej w możliwie pełny sposób zależność między mocą pobieraną przez procesor a całkowitym poborem mocy przez komputer wykonano szereg pomiarów przy różnych poziomach obciążenia procesora. Obciążenie, realizowane za pomocą zmodyfikowanego programu stres. W pierwszym przypadku wybrano 12 poziomów, pozwalających dla pojedynczego wątku, na obciążenie procesora na poziomie od około 3% do 100% (według wskazań programu top). Liczbę wątków zmieniano w zakresie od 1 do 8 – wynika to z faktu, że procesor i7 wyposażony jest w 4 rdzenie, na których może wykonywać po 2 wątki (Hyperthreading).

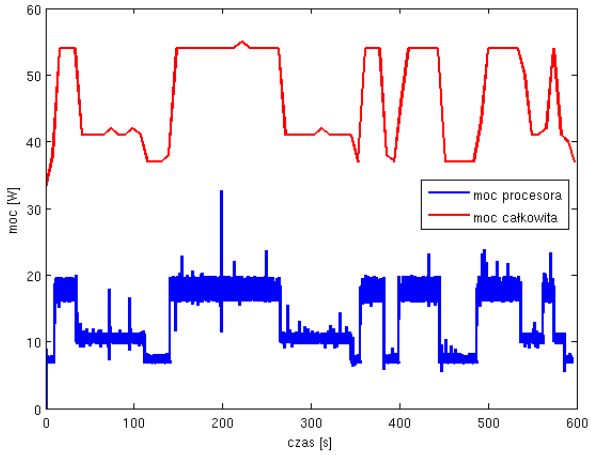
Przy tak prowadzonym eksperymencie system operacyjny dostosowuje częstotliwość taktowania procesora do obciążenia. Z charakterystyki standardowego sterownika on-demand wynikają stosunkowo częste skoki częstotliwości sięgające wartości maksymalnej (patrz. Rys.1). Pomiary zostały wykonane co 100 ms, co jest wartością znacznie niższą niż dynamika zmian

obciążenia (w tym wypadku przerwy między obliczeniami w programie stress trwały 160us).



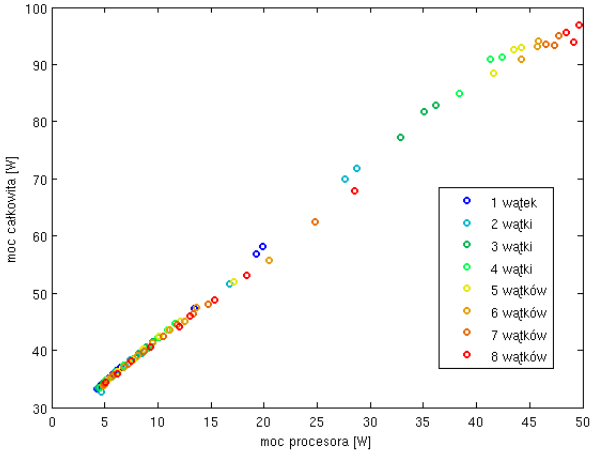
Rysunek 1: Zmiany częstotliwości zegara procesora przy obciążeniu procesem obliczeniowym.

Z faktu takiego sposobu sterowania zegarem wynika również skokowa postać wykresu poboru mocy przedstawionego na Rys. 1. Pomiar ten był wykonywany co 10 ms, co pozwala zauważyć iż dynamika tego procesu jest mniejsza niż zmiany obciążenia. Należy przypuszczać, że wynika to z okresu repetycji sterownika zegara procesora (100 ms) czego potwierdzeniem jest porównanie obu wykresów widocznych na Rys.2 przedstawiającym porównanie przebiegu całkowitego poboru mocy i mocy pobieranej przez procesor. Widoczna jest ich dużą zgodność – potwierdza to hipotezę o istnieniu korelacji między tymi wielkościami. Mimo iż moc całkowita jest mierzona co około 8s jej przebieg odwzorowuje co do obwiedni przebieg zapisu poboru mocy przez procesor.



Rysunek 2: Porównanie całkowitego poboru mocy i mocy pobieranej przez procesor dla obciążenia obliczeniowego.

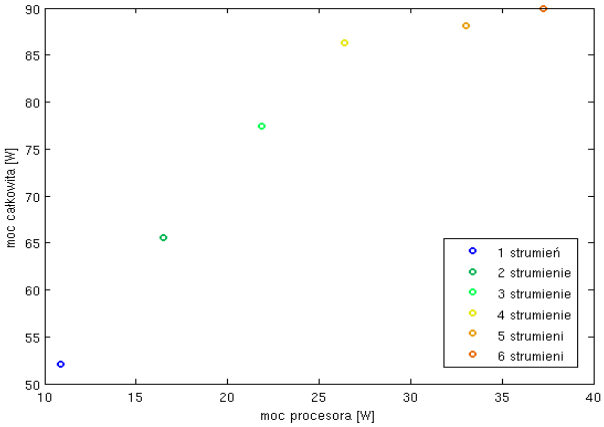
Zmienny charakter poboru mocy wymaga, aby w dalszych analizach posługiwać się uśrednionymi wartościami mocy. Wykres zależności uśrednionego całkowitego poboru mocy od mocy pobieranej przez procesor przedstawia Rys. 3. Bez szczegółowych analiz, tylko na podstawie wykresu, można stwierdzić, że zastosowanie modelu liniowego jest w tym wypadku dopuszczalne. Wiarygodność wyników podnosi fakt użycia stosunkowo dużej liczba punktów pomiarowych – przeprowadzono 96 eksperymentów, a pomiar mocy uśredniano z ponad 70 próbek dla pomiaru mocy całkowitej i około 60000 dla pomiaru mocy pobieranej przez procesor.



Rysunek 3: Zależność mocy całkowitej od mocy procesora dla obciążenia obliczeniowego.

Eksperymenty pomiarowe: serwer przekodowujący wideo

Przekodowywanie wideo jest zadaniem intensywnie obciążającym procesor, przez co możliwe jest przetwarzanie maksymalnie tylko 6 strumieni. Generowane przy tym obciążenie portów ruchem nie jest zbyt duże i wynosi około 20 Mb/s na strumień, czyli przy maksymalnym obciążeniu stanowi około 12% przepustowości portu. Wykres zmierzonej zależności przedstawia Rys. 4.



Rysunek 4: Zależność mocy całkowitej od mocy procesora dla przekodowywania strumienia wideo.

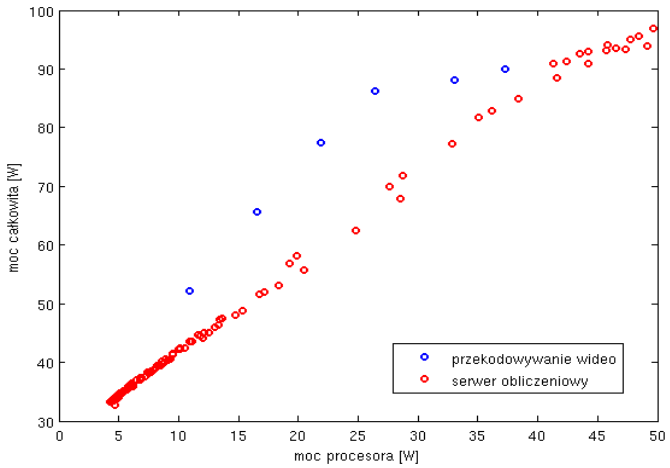
Przebieg charakterystyki poboru mocy różni się znacząco od przedstawionych wcześniej, połączenie punktów pomiarowych daje linie wklęsłą, o zmniejszającym się nachyleniu. Aby właściwie zinterpretować tę prawidłowość należy zauważyć, że przekodowywanie strumienia wideo, co prawda obciąża głównie procesor, ale różni się od typowych zadań obliczeniowych, gdyż wiąże się z dość ostrymi wymaganiami czasowymi. Zagięcie charakterystyki jest widoczne głównie dla 5 i 6 strumieni, czyli obciążenia, przy którym pojawiają się opóźnienia w przekodowywaniu ramek. Są one na tyle niewielkie, ze nie powodują zerwania transmisji, jest to jednak wynikiem zastosowania dość dużych buforów po stronie odbiorczej. Dla większej liczby strumieni niezakłócona transmisja nie jest już możliwa. Należy zauważyć zbieżność z liczbą rdzeni procesora (4) – sugeruje to, iż źródłem opóźnień mogą być problemy z przełączaniem wątków i dostępem do pamięci.

Dodatkowo, transmisja danych z pomocą tuneli zestawionych programami netcat obciąża procesor czynnościami wykonywanymi w scenariuszu rutera programowego przez kartę sieciową (offloading). Znajduje to odbicie w nachyleniu początkowej części charakterystyki, które jest wyższe niż w pierwszym scenariuszu.

Taki układ punktów pomiarowych wskazuje, że dla w miarę dokładnej aproksymacji przebiegu charakterystyki w pełnym zakresie, potrzebne jest zastosowanie funkcji nielinowej. Należy przypuszczać, że zastosowanie funkcji liniowej może dać dobre wyniki dla początkowej części charakterystyki – w zakresie, gdzie procesor nie jest całkowicie wykorzystany.

Eksperymenty pomiarowe: porównanie scenariuszy

Zestawienie charakterystyk zmierzonych dla scenariuszy serwera obliczeniowego i przekodowywania wideo przedstwia Rys. 5. Jego analiza wykazuje, że charakterystyka poboru mocy w scenariuszu przekodowywania wideo układa się powyżej punktów zmierzonych dla serwera obliczeniowego, przy czym jej początkowe nachylenie jest większe. Wskazuje to na zaangażowanie dodatkowych elementów systemu (karty sieciowej) przez transmisję danych. Zagięcie końcowego fragmentu charakterystyki można tłumaczyć mniejszym wykorzystaniem innych elementów niż procesor (karty sieciowej, pamięci) w sytuacji pojawienia się opóźnień w dekodowaniu ramek. Pobór mocy – tak przez procesor, jak i cały komputer jest znacząco większy niż w przypadku rutera programowego -- potwierdza to, że procesor ma decydujący udział w bilansie mocy.



Rysunek 5: Porównanie charakterystyk poboru mocy przez serwery: obliczeniowy i przekodowujący wideo.

Identyfikacja modelu

Wyniki opisanych wcześniej pomiarów wykazują znaczną regularność. Sugeruje to, że powinno być możliwe stworzenie stosunkowo prostych modeli wiążących całkowite zużycie energii przez komputer z poborem mocy przez procesor odczytanym z rejestrów msr. Dla scenariusza pierwszego wystarczający powinien być model linowy – nie odda on oczywiście wszystkich odchyłek widocznych na niektórych charakterystykach, można jednak spodziewać się, że dokładność aproksymacji będzie dostateczna dla większości zastosowań. Scenariusz 2 wymaga nieco bardziej złożonego podejścia. Wydaje się, że warto rozważyć dwa przypadki różniące się stopniem obciążenia maszyny. Dla mniejszej liczby strumieni (w zakresie 1-4) można użyć modelu liniowego. Dla modelowania zależności w pełnym zakresie konieczny jest model nieliniowy, przy czym stosunkowo nieskomplikowany przebieg charakterystyki jest argumentem za zastosowaniem funkcji drugiego stopnia.

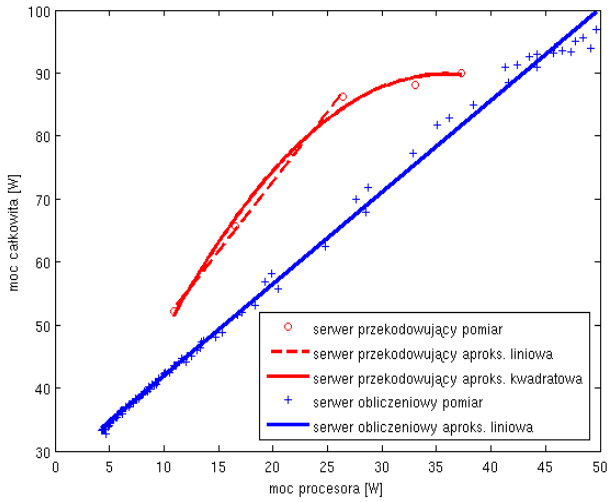
Proponowane modele są stosunkowo proste co powinno ułatwić porównanie i ocenę ogólnych właściwości omawianych przypadków. Ich zaletą jest zdolność do odfiltrowania i uśrednienia części zakłóceń oraz możliwość przeprowadzenia procedury identyfikacyjnej przy stosunkowo niewielkiej liczbie danych – ma to znaczenie szczególnie dla scenariusza 2.

Model drugiego stopnia stosowany w scenariuszu 2 można zapisać następująco:

$$p(w) = \alpha_0 + \alpha_1 w + \alpha_2 w^2,$$

gdzie: *w* jest mocą odczytaną z rejestrów msr procesora, *p(w)* jest całkowitym poborem mocy przez komputer a α_0, α_1, α_2 są zidentyfikowanymi współczynnikami modelu. Należy przy tym zauważyć, że model liniowy, użyty w przypadku pierwszego scenariusza, jak również dla zawężonego zakresu zmian mocy pobieranej przez procesor w drugim scenariuszu, można traktować jako szczególny przypadek tego modelu, w którym współczynnik α_2 jest równy zero.

Nachylenie linii dopasowanych do danych zebranych dla scenariusza przekodowywania wideo jest znacząco większe niż przy obciążeniu czysto obliczeniowym (patrz Rys. 6). Widoczne jest zagięcie charakterystyki dla 5-6 strumieni – może ono być oddane wyłącznie przez model nieliniowy i jak zostało wspomniane wcześniej wiąże się najprawdopodobniej z osiągnięciem przez system kresu wydajności. Znajduje to odbicie w mniejszym nachyleniu prostej dopasowanej w przypadku 1-4 strumieni.



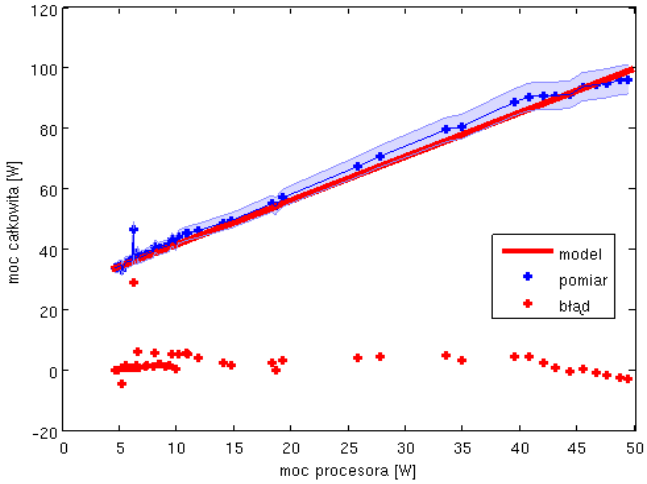
Rysunek 6: Identyfikacja modeli dla serwerów: obliczeniowego i przekodowującego wideo.

Weryfikacja modelu

Dla sprawdzenia poprawności identyfikacji przeprowadzono dodatkową serię eksperymentów pomiarowych. Umożliwiło to ocenę dokładności w nowych punktach, a przez to weryfikację poprawności założenia o kształcie modelu. Jako miarę dopasowania modelu wykorzystano średnią wartość bezwzględną błędu. Podobnie jak poprzednio przeprowadzono dwie serie eksperymentów pomiarowych polegających na: wykonywaniu obliczeń o zadanej intensywności (program stress) oraz przekodowywaniu wideo. Eksperymentów prowadzono dla nieco zmienionych parametrów obciążenia, w szczególności przekodowywanie wideo wykonano dla strumieni o przepływności zmniejszonej do około 80% w stosunku do wartości wykorzystywanych przy identyfikacji.

Graficzne przedstawienie wyników eksperymentu weryfikacyjnego dla przypadku serwera obliczeniowego zawiera Rys. 7. Jasnym kolorem oznaczono margines o szerokości 5% w stosunku

do zmierzonych wartości. Jak można zauważyć, poza pojedynczym punktem w początkowej części wykresu, linia odpowiadająca modelowi mieści się w tak określonym zakresie dokładności, przy czym średnia wartość bezwzględna błędu nie jest większa niż 3,1%. Można to interpretować jako potwierdzenie słuszności założenia o liniowości modelu, przy czym przyjęta dokładność (5%) powinna być wystarczająca dla większości zastosowań związanych z sterowaniem. Pojedyncza próbka wykazująca większą niż 5% odchyłkę jest najprawdopodobniej wynikiem błędu pomiarowego, czy też efektem uruchomienia się w czasie pomiaru jednego z procesów systemowych – np. indeksacji plików, który to proces intensywnie używa dysków a przez to może znacząco podwyższać całkowity pobór mocy. Falisty przebieg punktów pomiarowych w końcowym odcinku charakterystyki może jednak świadczyć o występowaniu pewnych nieregularności wynikających z nierównomiernego obciążenia elementów składowych maszyny.



Rysunek 7: Weryfikacja modelu poboru mocy serwera obliczenowego.

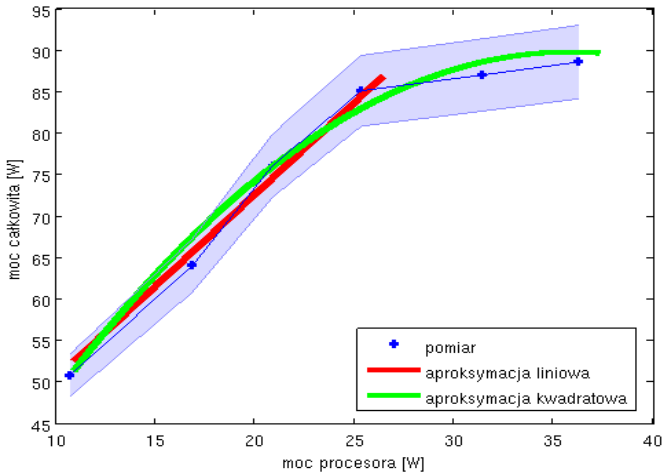
Przypadek serwera przekodowującego wideo jest, jak wskazano wcześniej bardziej skomplikowany. Modelowanie zależności całkowitego poboru mocy od mocy odczytywanej z rejestrów procesora w pełnym zakresie dopuszczalnych obciążeń wymaga użycia funkcji nieliniowej. Na Rys. 8 przedstawiono wyniki weryfikacji nastrojonego wcześniej modelu. Oznaczony kolorem błękitnym margines wynosi podobnie jak w przypadku serwera obliczeniowego 5%. Oba modele: kwadratowy pokrywający pełny zakres zmian i liniowy działający tylko dla mniejszych obciążeń mieszczą się w założonej dokładności. Układ punktów pomiarowych wskazuje, że zakres, który przyjęto dla modelu liniowego jest właściwy. Oznacza to, że punkt zagięcia charakterystyki całkowitego poboru mocy jest dobrze określony poprzez wartość mocy pobieranej przez procesor.

2.2. Modelowanie dynamiki przetwarzania danych

Dla zaprojektowania wydajnego sterownika procesora niezbędny jest model dynamiki przetwarzania danych. Ze względu na złożoność wykonywanych operacji i interakcję wielu elementów obliczeniowych w stochastycznym środowisku systemu operacyjnego jest to trudny problem inżynieryjny. Podstawowym wyzwaniem jest generowanie zmiennych obciążeń pobudzających wszystkie zaangażowane elementy obliczeniowe i w ten sposób umożliwiających identyfikację pełnego modelu dynamiki systemu.

W tym celu można wykorzystać metodykę zgodną z RFC2544 [6]. Poroponowany eksperyment polega na filtrowaniu przez testowany serwer wielu strumieni pakietów UDP, których średnia prędkość transmisji może być przedstawiona jako sygnał o zadanej charakterystyce widmowej. Poprzez odpowiedni dobór tej charakterystyki można zaprojektować eksperymenty pozwalające na

identyfikację modeli odpowiadających róznorodnym zastosowaniom – np. serwerom WWW, zaporom siecowym lub systemom wykrywania włamań.



Rysunek 8: Weryfikacja modelu poboru mocy serwera przekodowującego wideo.

Stanowisko pomiarowe

Do pomiarów wykorzystano serwer przetwarzający dwa strumienie ruchu UDP odbierane na niezależnych portach od dwóch niezależnych maszyn – generatorów ruchu. Badany serwer pracował z ustaloną częstotliwością zegara procesora, w zakresie od 1,6 do 3,40 GHz. Strumienie ruchu były generowane ze zmienną intesywnością $A=100r$ Mbps, r = 1,...,10. W celu obserwacji wydajności filtrowania pakietów została zaimplementowana specjalna sonda działająca z częstotliwością f_s=100Hz i przekazująca pomiary do jądra systemu operacyjnego.

Rejestrowano następujące dane:

1. obciążenie procesora wyrażone w procentach bieżącej częstotliwość taktowania,
2. obciążenie procesora wyrażone w MHz jako częstotliwość wykonania instrukcji,
3. pobór mocy procesora (RAPL MSRs),
4. liczba pakietów, które przeszły przez filtr pakietów jądra (statystyka ps_recv biblioteki libpcap),
5. liczba pakietów odrzuconych przez filtr pakietów jądra ze względu na wyczerpaną przestrzeń bufora (statystyka ps_drop biblioteki libpcap).

Konstrukcja sondy

Aby obserwować dynamikę operacji filtrowania pakietów konieczne jest zbieranie danych z wysoką częstotliwością próbkowania, co jest mozliwe tylko w przypadku sondy działającej na poziomie jądra systemu operacyjnego. Może ona być implementowana na wiele sposobów, patrz np. [7][8][9][10][11]. W opisywanym przypadku zdecydowano się na umieszczenie sondy w module jądra cpufreq [5]. Pozwoliło to rejestrować pomiary z częstotliwością f_s=100 Hz, równą częstotliwości z jaką działa sam moduł cpufreq. Pomiary zapisywane są poprzez mechanizm dziennika systemu (w pliku /var/log/messages). Najistotniejsze elementy kodu źródłowego sondy zamieszczono w dodatku (patrz Dodatek, Listing 1). Wprowadzono także dedykowany kanał netlink pozwalający zbierać pomiary dotyczące pracy biblioteki libpacp. Odpowiadają za to operacje input_observer_run() i output_observer_run() (patrz dodatek, Listing 2).

Konstrukcja sygnału wejściowego

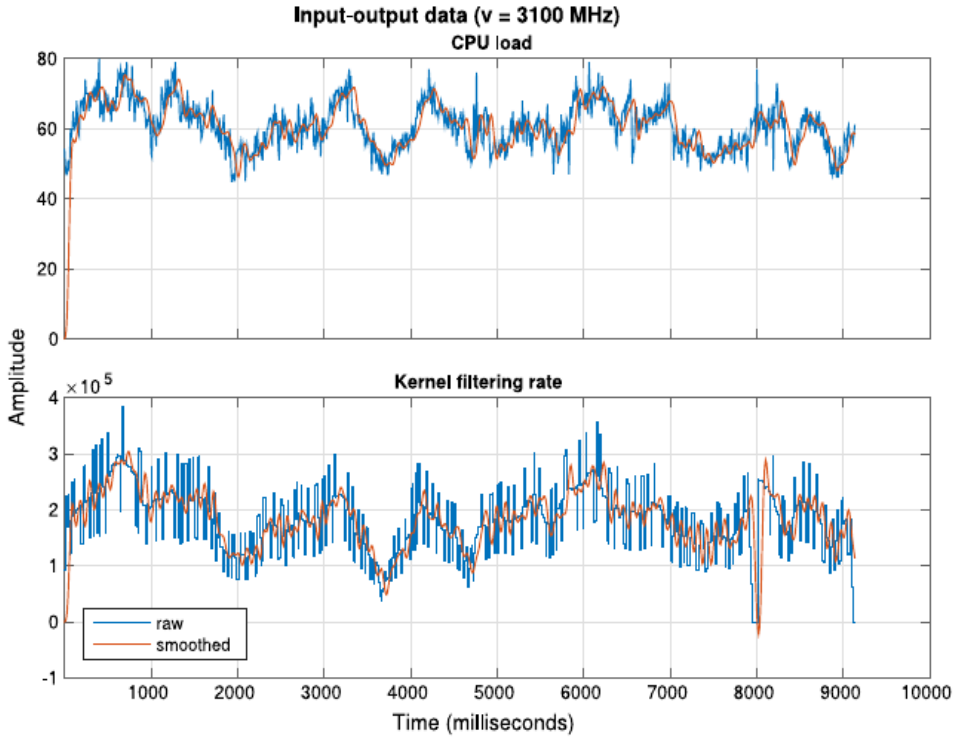
Dla właściwego zidentyfikowania modelu dynamiki przetwarzania istotna jest postać użytego sygnału wejściowego. W celu uwidocznienia zachowania systemu w możliwie szerokim zakresie częstotliwości wykorzystano sygnał będący złożeniem wielu fal sinusoidalnych. Ze znanych sposobów konstrukcji takich wejść (patrz np. [12][13]) w toku eksperymentów wybrano tzw. fazy Schroedera. Dokładniej, zastosowano następującą rodzinę sygnałów:

$$
\begin{aligned}
w(t) &= \sum_{k=1}^{N_f} A_k \sin(2\pi f_k t + \varphi_k), \\
t &= lT_s,\ T_s = 1/f_s,\ f_k = kf_s/N_t, \\
l &= 0, \ldots, N_t - 1,\ N_f = 30,\ N_t = 1000, \\
\varphi_k &= -k(k-1)\pi/N_f,\ k = 1, \ldots, N_f,
\end{aligned}
$$

gdzie N_f oznacza liczbę częstotliwości w sygnale, N_t liczbę próbek, *w(t)* scenariusz generowania ruchu. Należy pamiętać, że wybrany zakres częstotliwości może zostać ograniczony przez rozdzielczość pomiarów i możliwości oprogramowania generującego ruch. Innym sposobem może być odtwarzanie zapisu rzeczywistego ruchu z pliku .pcap po uprzednim przefiltrowaniu w celu wybrania odpowiednich częstotliwości.

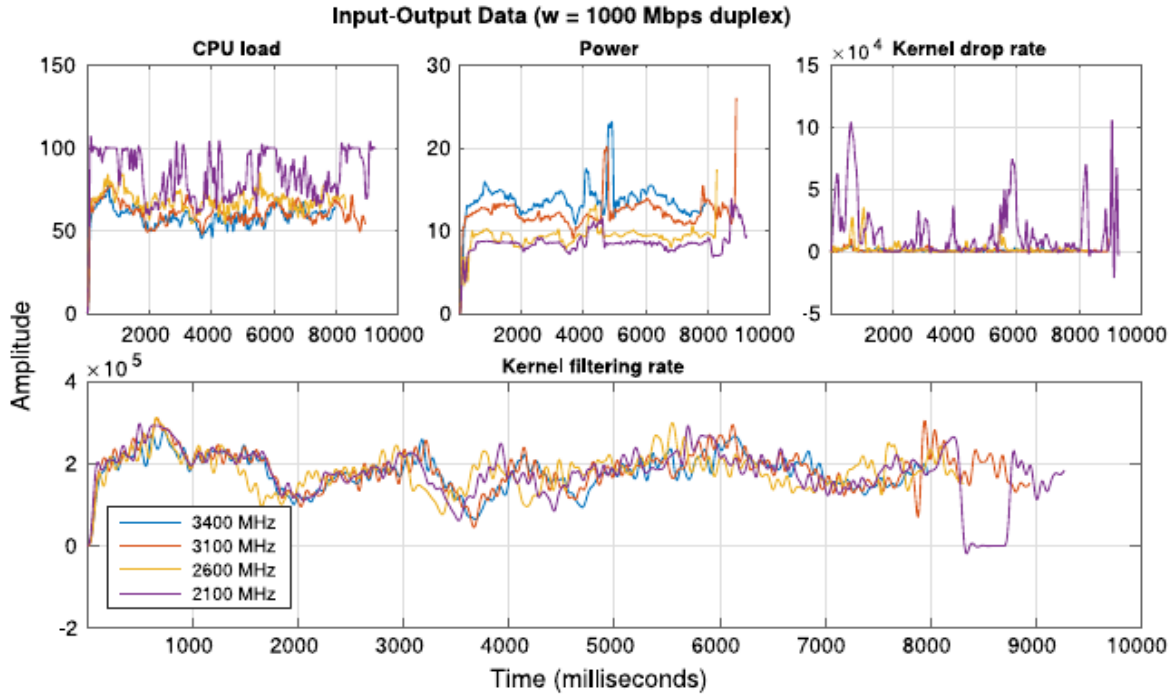
Identyfikacja i weryfikacja modelu

Typowy wynik doświadczenia, w którym serwer jest zmuszony do filtrowania pakietów przy ustalonej częstotliwości taktowania procesora przedstawia wykres Rys. 9. Obserowane obciążenie procesora charakteryzuje się mniejszą wariancją niż wywołujący je ruch. Można to wytłumaczyć buforowaniem zadań wykonywanych przez jądro.



Rysunek 9: Przykład dynamiki systemu: obciążenie procesora (wykres górny) i odpowiadjące nu natężenie ruchu sieciwego.

Porównanie dynamiki obciążenia, strat pakietów pakietów i poboru mocy przy różnych częstotliwościach taktowania procesora pokazuje Rys. 10. Zgodnie z oczekiwaniami, średnie obciążenie procesora i straty pakietów maleją wraz z wzrostem częstotliwości taktowania procesora, natomiast pobierana moc wzrasta. Sporadycznie można zaobserwować większe fluktuacje zarówno w obciążeniu procesora, jak i stratach pakietów. Efekt ten może być częściowo wyjaśniony przez nieprzewidywalne działania systemu, takie jak przełączanie kontekstów czy dostęp do pamięci.



Rysunek 10: Porównanie dynamiki procesora przy różnych częstotliwościach: obciążenie procesora, pobór mocy i poziom strat (na górze), oraz prędkość filtrowania pakietów (na dole).

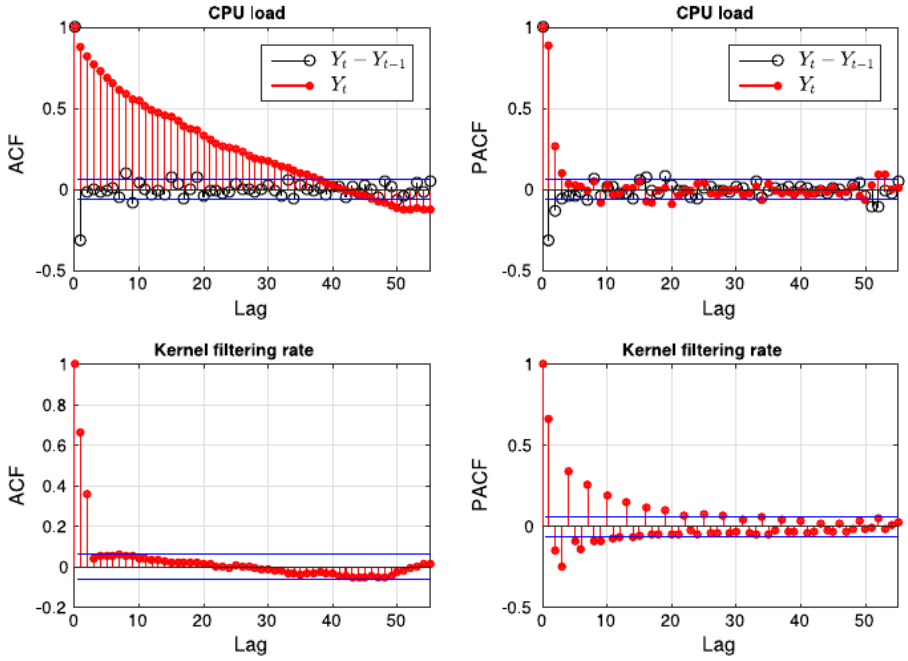
Rys. 11 przedstawia funkcję autokorelacji (ACF) i częściową funkcja autokorelacji (PACF) obserwowanego obciążenia procesora Y_t, i jego pierwszych różnic Y_t - Y_{t-1}. Wyniki te sugerują, że dynamika procesu może być opisana przez model średniej ruchomej niskiego rzędu. Podstawową obserwacją jest tutaj fakt, że skutki przetwarzania pakietów pozostają widoczne w obciążeniu procesora przez okres próbkowania.

Przykładowy model Boxa-Jenkinsa dla stałej częstotliwości, u=3.1GHz, przedstawiają następujące rónania:

$$
\begin{aligned}
y(t) &= \frac{B^*(q^{-1}|u)}{F^*(q^{-1}|u)}w(t) + \frac{C^*(q^{-1}|u)}{D^*(q^{-1}|u)(1-q^{-1})}e(t), \\
B^*(q^{-1}|u) &= 4.755 \cdot 10^{-6}q^{-2} + 4.677 \cdot 10^{-6}q^{-3}, \\
C^*(q^{-1}|u) &= 1 - 0.7905q^{-1} + 0.07769q^{-2}, \\
D^*(q^{-1}|u) &= 1 - 0.2198q^{-1} + 0.01524q^{-2}, \\
F^*(q^{-1}|u) &= 1 - 0.4874q^{-1},
\end{aligned}
$$

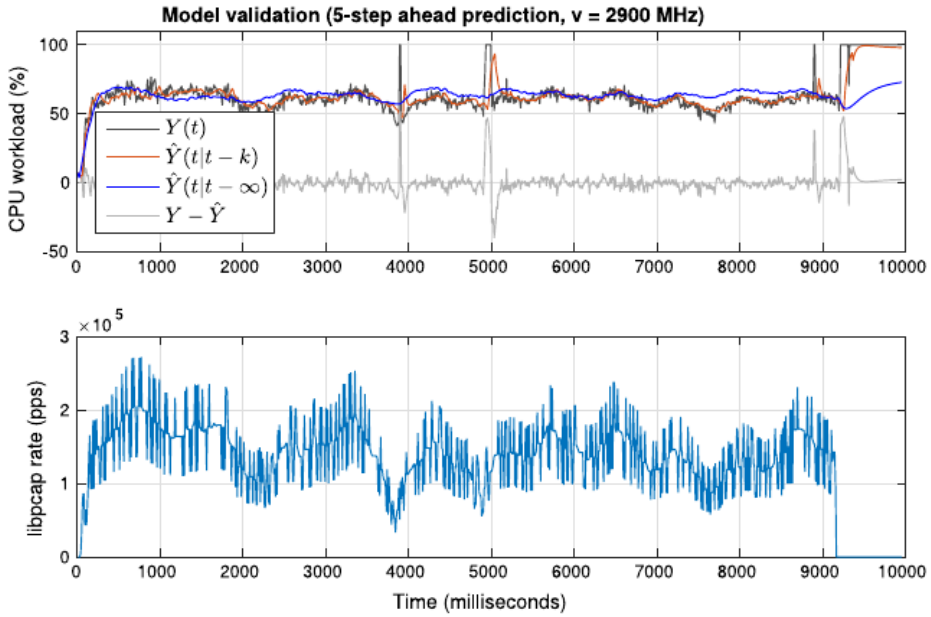
gdzie w oznacza wejście serwera, e biały szumy a q reprezentuje operatora opóźnienia. Innym przykładem jest może być model ARMAX uzyskany dla u=2. 6 GHz:

$$
\begin{aligned}
A^*(q^{-1}|u)y(t) &= B^*(q^{-1}|u)w(t) + C^*(q^{-1}|u)e(t),\\
A^*(q^{-1}|u) &= 1 - 0.9971q^{-1},\\
B^*(q^{-1}|u) &= 1.596^{-6}q^{-1},\\
C^*(q^{-1}|u) &= 1 - 0.135q^{-1}.
\end{aligned}
$$



Rysunek 11: Wykresy funkcji autokorelacji (z lewej) i częściowej autokorelacji (z prawej) obciążenia procesora (na górze) i natężenia filtrowanego ruchu (na dole).

Rys. 12 ilustruje wyniki eksperymentu weryfikacji modelu, w którym predykcja *Y(t|t–k)* wykonana na 5 kroków wprzód dokładnie odtwarza obciążenie procesora generowane przez operacje filtrowania pakietów. Wyniki symulacji obciążenia Y(t|t-∞), pokazują wygładzanie wejścia i całkowanie zakóceń w modelu, natomiast nagłe skoki obciążenia nie związane z operacjami filtrowania pakietów, nie mogą być symulowane przez zastosowany model liniowy.



Rysunek 12: Weryfikacja modelu Boxa-Jenkinsa.

3. Dodatek: Oprogramowanie – kod źródłowy

Zamieszczone w dodatku fragnenty kodu służą demostracji możliwości wzbogacenia jądra Linuksa o funkcjonalność sondy pozwalającej na śledzenie stanu aplikacji (Listing 1), przy czym dane pomiarowe mogą być odczytywane przy pomocy standardowego protokołu NETLINK (Listing 2). Listing 3 jest przykładem skryptu umożliwiającego monitorowanie zużycie energii z pomocą rejestrów MSR, zaś Listing 4 pokazuje jak te same operacje można wykonać z poziomu programu w języku C.

Listing 1: Kluczowe fragmenty kodu sondy

```
/*...*/
int
input_observer_run(struct sk_buff *skb, struct genl_info *info)
{/*...*/}
int
output_observer_run(struct sk_buff *skb, struct genl_info *info)
{/*...*/}
/*...*/
static enum hrtimer_restart
  sampling_function(struct hrtimer* unused){

  if(send_signal_to_app() < 0) {/*...*/}
  get_cpu_load();
  sampling_counter++;
  if(sampling_counter>=MAX_SAMPLES){
    return HRTIMER_NORESTART;
  }
  else {
    hrtimer_forward_now(&cpuload_probe, ktime);
    return HRTIMER_RESTART;
  }
}
/*...*/
int init_module(void){
  /*...*/
  ktime = ktime_set( 0, MS_TO_NS(sampling_rate) );
  hrtimer_init(&cpuload_probe, CLOCK_MONOTONIC, HRTIMER_MODE_REL);
  cpuload_probe.function = &sampling_function;
  hrtimer_start(&cpuload_probe, ktime, HRTIMER_MODE_REL);
  /*...*/
}
/*...*/
module_param(sampling_rate, long, S_IRUSR);
MODULE_PARM_DESC(sampling_rate, "Sampling rate");
module_param(app_pid, long, S_IRUSR);
MODULE_PARM_DESC(app_pid, "Application PID");
```

Listing 2: Przykład konfiguracji połączenia przez gniazda NETLINK

```
enum {
  SIGNAL_ATTRB_UNSPEC, SIGNAL_ATTRB_MSG, __SIGNAL_ATTRB_MAX,
};
#define SIGNAL_ATTRB_MAX (__SIGNAL_ATTRB_MAX - 1)

/* Protocol attribute validation policy */
static struct nla_policy
  app_observer_genl_policy[SIGNAL_ATTRB_MAX + 1] = {
  [SIGNAL_ATTRB_MSG] = { .type = NLA_U32 },
};

/* Generic netlink family specification */
#define VERSION_NR 1
static struct genl_family
  app_observer_gnl_family = {
    .id       = GENL_ID_GENERATE,
    .hdrsize  = 0,
    .name     = "APP_OBSERVER",
    .version  = VERSION_NR,
    .maxattr  = SIGNAL_ATTRB_MAX,
};

enum {
  APP_OBSERVER_UNSPEC,
  INPUT_OBSERVER_RUN,
  OUTPUT_OBSERVER_RUN,
  __APP_OBSERVER_MAX,
};
#define APP_OBSERVER_MAX (__APP_OBSERVER_MAX - 1)

int
input_observer_run(struct sk_buff *skb, struct genl_info *info);
/* Generic netlink operations */
struct genl_ops input_observer_gnl_ops = {
  .cmd    = INPUT_OBSERVER_RUN,
  .flags  = 0,
  .policy = app_observer_genl_policy,
  .doit   = input_observer_run,
  .dumpit = NULL,
};

int
output_observer_run(struct sk_buff *skb, struct genl_info *info);
/* Generic netlink operations */
struct genl_ops output_observer_gnl_ops = {
  .cmd    = OUTPUT_OBSERVER_RUN,
  .flags  = 0,
  .policy = app_observer_genl_policy,
  .doit   = output_observer_run,
  .dumpit = NULL,
};
/* ... */
```

Listing 3: Przykładowy skrypt moniturujący pobór mocy

```
#!/bin/bash

SAMPLING_RATE=1                  # seconds
MSR_PKG_ENERGY_STATUS="0x611"  # CPU energy
    counter
MSR_DRAM_ENERGY_STATUS="0x619" # DRAM energy
    counter

# Energy Status Units (ESU)
ESU=`echo "ibase=16;\
␣␣␣␣␣1/2^$(rdmsr␣-X␣0x606␣-f␣12:8)" | bc -l`
# Calculate number of CPU energy status
# counter incremants during sampling period
ESPKG=`a=$(rdmsr -X $MSR_PKG_ENERGY_STATUS);\
       sleep $SAMPLING_RATE; echo "ibase=16;\
␣␣␣␣␣␣␣$(rdmsr␣-X␣$MSR_PKG_ENERGY_STATUS)-$a"|bc`
# Calculate DRAM energy status
# counter incremants during sampling period
ESDRAM=`a=$(rdmsr -X $MSR_DRAM_ENERGY_STATUS);\
        sleep $SAMPLING_RATE; echo "ibase=16;\
␣␣␣␣␣␣␣␣$(rdmsr␣-X␣$MSR_DRAM_ENERGY_STATUS)-$a"|
    bc`
# Calculate power consumption [W]
CPUPOW=`echo "$ESPKG␣*␣$ESU" | bc -l`
DRAMPOW=`echo "$ESDRAM␣*␣$ESU" | bc -l`
echo CPU: $CPUPOW W
echo DRAM: $DRAMPOW W
```

Listing 4: Przykład funkcji monitorującej pobór mocy z wykorzystaniem rejestrów MSR

```
int read_msr(int cpu, unsigned int address,
    uint64_t *value)
{
  int   err = 0;
  char  msr_path[32];
  FILE *fp;

  sprintf(msr_path, "/dev/cpu/%d/msr", cpu);
  err = ((fp = fopen(msr_path, "r")) == NULL);
  if (!err) err = (fseek(fp, address, SEEK_CUR)
       != 0);
  if (!err) err = (fread(value, sizeof(uint64_t)
      , 1, fp) != 1);
  if (fp != NULL) fclose(fp);
  return err;
}
```

BIBLIOGRAFIA

[1] Koomey J.: Growth in data center electricity use 2005 to 2010, *A report by Analytical Press, completed at the request of The New York Times*, 2011

[2] Barroso L., André H. U: The case for energy-proportional computing, *IEEE computer* 40(12), 33–37, 2007

[3] Subramaniam B., Saunders W., Scogland T., Feng W.: Trends in energy-efficient computing: A perspective from the Green500, *Green Computing Conference (IGCC), 2013 International*, 1–8, 2013

[4] Intel: Intel R 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf, 2015.

[5] Pallipadi V., Starikovskiy A.: The ondemand governor, *Proc. of the Linux Symposium*, volume 2, 215–230, 2006

[6] S. Bradner, J. McQuaid, RFC 2544: Benchmarking methodology for network interconnect devices, 1999.

[7] M. Desnoyers, M.R. Dagenais, The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux, in: OLS (Ottawa Linux Symposium), vol. 2006, Citeseer, 2006, pp. 209–224.

[8] R. Matias, I. Beicker, B. Leitão, P.R. Maciel, Measuring software ageing effects through OS kernel instrumentation, in: IEEE Second International Workshop on Software Aging and Rejuvenation, IEEE, 2010, pp. 1–6.

[9] B. Lee, S. Moon, Y. Lee, Application-specific packet capturing using kernel probes, in: IFIP/IEEE International Symposium on Integrated Network Management, IEEE, 2009, pp. 303–306.

[10] M.-H. Wang, C.-M. Yu, C.-L. Lin, C.-C. Tseng, L.-H. Yen, KPAT: A kernel and protocol analysis tool for embedded networking devices, in: IEEE International Conference on Communications, IEEE, 2014, pp. 1160–1165.

[11] A. Khoroshilov, V. Mutilin, E. Novikov, I. Zakharov, Modeling environment for static verification of Linux kernel modules, in: Perspectives of System Informatics, Springer, 2014, pp. 400–414.

[12] R. Pintelon, J. Schoukens, System Identification: A Frequency Domain Approach, John Wiley & Sons, 2012.

[13] T. Soderstrom, P. Stoica, System Identification, Prentice Hall International, UK, 1989.