Trendy zamiast migawek: vmstat, iostat, sar, pidstat

Poziom: średnio **Dotyczy: wszystkie dystrybucje (sysstat dla iostat/sar/pidstat)**

Czas: ~20 min

`free` i `ps` dają zdjęcie stanu w jednej chwili. `vmstat`, `iostat`, `sar` i `pidstat` pokazują trend — kilka próbek w odstępach czasu, z których dopiero widać, czy problem narasta, czy to jednorazowy szczyt. Ta strona uczy czytać ich kolumny i unikać najczęstszego błędu: brania pierwszej próbki na poważnie.

Skróty użyte na tej stronie (7)

Skrót	Co znaczy
CPU	Central Processing Unit: procesor
HDD	Hard Disk Drive: tradycyjny dysk twardy z wirującymi talerzami i głowicą, wolniejszy od SSD, ale tańszy za gigabajt, typowo używany jako warstwa pojemnościowa
NVMe	Non-Volatile Memory Express: protokół komunikacji z dyskami SSD podłączonymi przez złącze PCIe, znacznie szybszy niż starsze SATA i SAS, typowy dla szybkich dysków systemowych
PID	Process ID: numer procesu w systemie
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SLES	SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL
SSD	Solid State Drive: dysk półprzewodnikowy

Fiszka

TYLKO ODCZYT

Pobieranie próbek co 2 sekundy: vmstat 2

```
vmstat 2
```

Bez drugiego argumentu `vmstat` powtarza pomiar co podaną liczbę sekund bez końca — przerywamy `Ctrl+C`. Pierwsza linia to zawsze średnia od startu systemu, nie próbka bieżąca — patrz *Koniecznie* niżej.

Przykład:

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0  524288 2150000  184000 5210000    0    0     2     1   50   30  5  2 92  1  0
 0  0  524288 2148000  184000 5211000    0    0     4     0   48   28  4  1 94  1  0
```

vmstat = virtual memory statistics: pamięć, CPU i dyski w czasie

TYLKO ODCZYT

Pobranie pięciu próbek i zakończenie: vmstat 2 5

```
vmstat 2 5
```

Drugi argument ogranicza liczbę powtórzeń — kończy się samo, wygodne w skryptach diagnostycznych i logach, bez ryzyka zapomnianej pętli w tle.

vmstat = virtual memory statistics: pamięć, CPU i dyski w czasie

TYLKO ODCZYT

Wypisanie sum i liczników od startu systemu: vmstat -s

```
vmstat -s
```

Zamiast kolejnych próbek w czasie, pokazuje pojedynczy zestaw skumulowanych liczników — ile razy łącznie coś się wydarzyło od ostatniego rozruchu.

Przykład:

```
      16308000 K total memory
       8598000 K used memory
       6120000 K active memory
       4890000 K inactive memory
       2150000 K free memory
        184000 K buffer memory
```

```
        5210000 K swap cache
        4194300 K total swap
         524288 K used swap
        3670012 K free swap
         842001 non-nice user cpu ticks
          12044 nice user cpu ticks
         398210 system cpu ticks
       15920044 idle cpu ticks
          88012 IO-wait cpu ticks
           2200 pages paged in
          48810 pages paged out
            410 pages swapped in
           1280 pages swapped out
```

vmstat = virtual memory statistics: pamięć, CPU i dyski w czasie

TYLKO ODCZYT

Sprawdzenie statystyk dysków bez iostat: vmstat -d

```
vmstat -d
```

Skrócony, surowszy odpowiednik `iostat` — liczby skumulowane od startu, nie procenty ani czasy oczekiwania. Przydatny, gdy `sysstat` nie jest zainstalowany, a potrzebujemy choćby zgrubnego obrazu.

Przykład:

```
disk- ------------reads------------ ------------writes----------- -----IO------
       total merged sectors      ms  total merged sectors      ms    cur    sec
sda    88410   1200  9840200  102400  205100   8400 41200400  380200      0    920
```

vmstat = virtual memory statistics: pamięć, CPU i dyski w czasie

TYLKO ODCZYT

Kolumny `vmstat` — pełny opis

Sześć grup kolumn opisuje sześć różnych obszarów jądra naraz, w jednym wierszu:

Kolumna	Grupa	Znaczenie
`r`	procs	procesy w kolejce gotowych do uruchomienia (czekają na CPU)
`b`	procs	procesy w nieprzerwalnym śnie, zwykle czekające na I/O
`swpd`	memory	ilość pamięci w swapie (KB)
`free`	memory	wolna pamięć (KB)
`buff`	memory	bufory jądra
`cache`	memory	page cache
`si`	swap	swap in — strony wczytywane z powrotem z dysku do RAM-u (KB/s)
`so`	swap	swap out — strony wypychane z RAM-u na dysk (KB/s)
`bi`	io	bloki odebrane z urządzenia blokowego (na sekundę)
`bo`	io	bloki wysłane na urządzenie blokowe (na sekundę)
`in`	system	liczba przerwań na sekundę
`cs`	system	liczba przełączeń kontekstu na sekundę
`us`	cpu	% czasu CPU w trybie użytkownika
`sy`	cpu	% czasu CPU w trybie jądra
`id`	cpu	% czasu CPU bezczynnego
`wa`	cpu	% czasu CPU czekającego na zakończenie I/O
`st`	cpu	% czasu „skradzionego" przez hipernadzorcę (maszyny wirtualne)

□□ **CPU** = Central Processing Unit: procesor

□□ **TYLKO ODCZYT**

Sprawdzenie rozszerzonych statystyk dysków: iostat -x 2

```
iostat -x 2
```

Wymaga pakietu `sysstat` (na wszystkich trzech głównych rodzinach dystrybucji nazywa się tak samo). `-x` (extended) dodaje kolumny czasu oczekiwania i nasycenia, bez których

surowe `iostat` mówi niewiele więcej niż `vmstat -d`.

Przykład:

```
Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  r_await  w_await
aqu-sz  %util
sda              10.50    5.20    512.00    256.40     0.50     1.20     2.50
5.00    0.08    8.50
```

□□ TYLKO ODCZYT

Ukrycie urządzeń bez ruchu w statystykach: iostat -xz 2

```
iostat -xz 2
```

`-z` (zero) ukrywa wiersze urządzeń, które w danym interwale nie miały żadnej aktywności — na serwerze z kilkunastoma dyskami/wolumenami to różnica między czytelną tabelą a stroną pełną zer.

□□ TYLKO ODCZYT

Kolumny `iostat -x` — pełny opis

Kolumna	Znaczenie
`r/s`, `w/s`	operacje odczytu/zapisu na sekundę (IOPS)
`rkB/s`, `wkB/s`	przepustowość odczytu/zapisu w KB/s
`await` (lub `r_await`/`w_await`)	średni czas oczekiwania operacji, łącznie z czasem w kolejce (ms)
`aqu-sz`	średnia długość kolejki żądań do urządzenia
`%util`	procent czasu, w którym urządzenie miało choć jedno żądanie w toku

□□ **IOPS** = Input/Output Operations Per Second: liczba operacji odczytu i zapisu na sekundę, podstawowa miara szybkości magazynu, którą podaje też Health Service S2D

□□ TYLKO ODCZYT

Sprawdzenie statystyk CPU: sar -u

```
sar -u 2 5
```

Statystyki procesora co 2 sekundy, 5 razy — te same kategorie co kolumny `cpu` w `vmstat` (`us`, `sy`, `id`, `wa`), ale w osobnym, przeznaczonym do tego narzędziu.

Przykład:

```
Linux 5.14.0 (srv01)   17.09.2026   _x86_64_  (8 CPU)

10:15:01     CPU     %user     %nice   %system   %iowait    %steal     %idle
10:15:03     all      5.20      0.00      2.10      1.30      0.00     91.40
10:15:05     all      4.80      0.00      1.90      0.90      0.00     92.40
```

CPU = Central Processing Unit: procesor · **sar** = system activity report: historyczne dane o obciążeniu CPU, pamięci i dysków zbierane w czasie

TYLKO ODCZYT

Sprawdzenie statystyk pamięci: sar -r

```
sar -r 2 5
```

Odpowiednik `free`, ale w postaci ciągu próbek w czasie zamiast jednego zdjęcia — widać trend zamiast punktu.

Przykład:

```
10:15:01    kbmemfree   kbavail kbmemused  %memused kbbuffers  kbcached
10:15:03      2150000   7412000   8598000     52.72    184000   5210000
10:15:05      2140000   7405000   8608000     52.78    184000   5211000
```

sar = system activity report: historyczne dane o obciążeniu CPU, pamięci i dysków zbierane w czasie

TYLKO ODCZYT

Sprawdzenie statystyk dysków: sar -d

```
sar -d 2 5
```

Odpowiednik `iostat -x`, w tym samym systemie zbierania danych co reszta `sar` — wygodne, gdy chcemy mieć CPU, pamięć i dyski w jednym, spójnym źródle danych historycznych.

Przykład:

```
10:15:01       DEV       tps  rkB/s  wkB/s  areq-sz  aqu-sz  await  %util
10:15:03    dev8-0     15.70  512.0  256.4     4.89    0.08   2.50    8.50
```

CPU = Central Processing Unit: procesor · **sar** = system activity report: historyczne dane o obciążeniu CPU, pamięci i dysków zbierane w czasie

TYLKO ODCZYT

Sprawdzenie statystyk interfejsów sieciowych: sar -n DEV

```
sar -n DEV 2 5
```

To samo podejście zastosowane do kart sieciowych — pakiety i bajty na sekundę per interfejs, w czasie.

Przykład:

```
10:15:01     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s
10:15:03      eth0     420.10    380.50    512.20    488.90
```

sar = system activity report: historyczne dane o obciążeniu CPU, pamięci i dysków zbierane w czasie

TYLKO ODCZYT

Odtworzenie danych historycznych z konkretnego dnia: sar -f /var/log/sa/saXX

```
sar -f /var/log/sa/sa15
```

`sysstat` domyślnie zapisuje próbki co 10 minut do plików binarnych `saDD` (gdzie `DD` to dzień miesiąca) w `/var/log/sa/`. `-f` pozwala odtworzyć dowolny wcześniejszy dzień — to jest jedyny z tu opisanych sposobów pokazania „co się działo wczoraj o 3 w nocy", pod warunkiem że usługa zbierająca dane była wtedy uruchomiona.

Przykład:

```
Linux 5.14.0 (srv01)   15.09.2026   _x86_64_  (8 CPU)

00:00:01        CPU     %user     %nice   %system   %iowait    %steal     %idle
00:10:01        all      3.10      0.00      1.40      0.20      0.00     95.30
03:00:01        all     42.80      0.00     18.20     31.40      0.00      7.60
```

sar = system activity report: historyczne dane o obciążeniu CPU, pamięci i dysków zbierane w czasie

ZMIENIA SYSTEM

Włączenie zbierania danych historycznych: systemctl enable --now sysstat

```
systemctl enable --now sysstat
```

Bez tego `sar` odpowie na pytanie o przeszłość pustką — usługa `sysstat` (na RHEL/Rocky pakiet `sysstat`, na Debianie/Ubuntu też `sysstat`, ale tam trzeba dodatkowo włączyć zbieranie w `/etc/default/sysstat`) uruchamia `sa1`/`sa2` przez cron/systemd timer, które faktycznie zapisują próbki do plików w `/var/log/sa/`.

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **systemctl** = system control: sterowanie usługami systemd

TYLKO ODCZYT

Sprawdzenie I/O per proces w czasie: pidstat -d 2

```
pidstat -d 2
```

`iostat` i `sar -d` pokazują ruch per *urządzenie*. `pidstat -d` (z tego samego pakietu `sysstat`) pokazuje ruch dyskowy per *proces* — odpowiedź na pytanie „kto konkretnie generuje ten zapis", której `iostat` nie da.

Przykład:

```
Linux 5.14.0 (srv01)   17.09.2026   _x86_64_  (8 CPU)

10:20:01    UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
10:20:03    999      1203      0.00    412.50      0.00       2  postgres
10:20:03      0      2044     88.20      0.00      0.00       0  java
```

◫◫ TYLKO ODCZYT

Sprawdzenie CPU jednego konkretnego procesu w czasie: pidstat -u -p NumerPID 2

```
pidstat -u -p NumerPID 2
```

> (gdzie `NumerPID` to identyfikator procesu z wyniku `ps aux` albo `pidstat`)

Odpowiednik `top` dla jednego wybranego PID-u, w formacie łatwym do zalogowania i porównania w czasie — bez interaktywnego ekranu, który trudno przekierować do pliku.

Przykład:

```
10:22:01      UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
10:22:03        0      2044   18.00    2.00    0.00    0.00   20.00     3  java
10:22:05        0      2044   45.00    5.00    0.00    0.00   50.00     3  java
```

◫◫ **CPU** = Central Processing Unit: procesor

◫◫ TYLKO ODCZYT

Obserwacja obciążenia dysku na żywo: iotop

```
iotop -o
```

Interaktywny odpowiednik `pidstat -d`, aktualizowany na żywo, z sortowaniem po zużyciu I/O. `-o` (`--only`) pokazuje wyłącznie procesy, które faktycznie coś czytają lub piszą w danej chwili — bez tego lista jest zaśmiecona procesami o zerowym ruchu. Wymaga uprawnień administratora, bo czyta liczniki I/O jądra dla wszystkich procesów.

◫◫ TYLKO ODCZYT

Wszystkie flagi

Polecenie	Co robi
`vmstat N` / `vmstat N C`	próbki pamięci, swapu, I/O i CPU co N sekund, C razy (albo bez końca)

Polecenie	Co robi
`vmstat -s`	skumulowane liczniki od startu systemu
`vmstat -d`	zgrubne statystyki dysków bez `sysstat`
`iostat -x N`	rozszerzone statystyki dysków, z `await` i `%util`
`sar -u` / `-r` / `-d` / `-n DEV`	CPU / pamięć / dyski / sieć, z pakietu `sysstat`
`sar -f plik`	odtworzenie danych historycznych z pliku `saDD`
`pidstat -d` / `-u`	I/O i CPU per proces w czasie

> **Uważaj**
>
> - Pierwsza próbka `vmstat` i `iostat` to średnia od startu systemu — zawsze ją odrzucaj, patrz tylko na kolejne.
> - `%util` bliskie 100% na dysku SSD/NVMe nie znaczy automatycznie „urządzenie w pełni nasycone" — patrz *Jak to działa*.
> - `sar` bez wcześniej włączonej usługi zbierającej dane (`sysstat`) nie ma żadnej historii do pokazania — `-f` odpyta pusty plik.

CPU = Central Processing Unit: procesor · **SSD** = Solid State Drive: dysk półprzewodnikowy

Z życia wzięte

Z ŻYCIA WZIĘTE

Piątek, 15:30, tuż przed zamknięciem miesiąca. Pani Halina skarży się, że Optima na sql01 odpowiada wolniej niż zwykle. Tomek uruchamia iostat i widzi, że dysk nie jest przeciążony ilością danych, tylko czasem oczekiwania na pojedyncze operacje, co wskazuje na problem z samą macierzą dyskową, nie z bazą.

Po co to istnieje

Pojęcie: średnia od startu systemu kontra próbka z interwału. `vmstat` i `iostat` liczą swoje liczniki jako skumulowane wartości od momentu uruchomienia jądra. Pierwsza linia wyniku to zawsze te skumulowane liczniki podzielone przez cały czas działania systemu —

czyli uśrednienie po tygodniach czy miesiącach pracy, kompletnie bezużyteczne do oceny „co się dzieje teraz". Każda kolejna linia to już różnica między dwoma kolejnymi odczytami liczników, podzielona przez rzeczywisty interwał między nimi — czyli faktyczna, aktualna prędkość. To jest najczęstszy błąd interpretacji tych narzędzi w całej administracji Linuksem, opisany wprost w ich dokumentacji, a mimo to regularnie pomijany.

Analogia

Wyobraź sobie **licznik przebiegu w samochodzie i prędkościomierz** — to dwa różne pomiary tej samej podróży.

Licznik przebiegu (odometr) pokazuje sumę wszystkich przejechanych kilometrów od dnia produkcji auta. Jeśli podzielisz tę liczbę przez wiek samochodu w godzinach, dostaniesz jakąś „średnią prędkość", ale kompletnie bezużyteczną do pytania „jak szybko jadę teraz" — auto stało w garażu większość czasu, ta średnia będzie śmiesznie niska niezależnie od tego, czy w tej chwili pędzisz autostradą, czy stoisz na czerwonym świetle.

Prędkościomierz pokazuje co innego: różnicę przebytej drogi między dwoma momentami, podzieloną przez czas między nimi. To jest dokładnie to, co robi druga i każda kolejna linia `vmstat` — różnica liczników między próbkami, nie suma od początku. Pierwsza linia `vmstat` to odczyt z licznika przebiegu podzielony przez wiek auta; reszta to prędkościomierz.

Jak to działa

Rozwiń: jak to działa pod spodem

Pojęcie: przepustowość kontra opóźnienie. Przepustowość (throughput) to ilość danych przesłanych w jednostce czasu — `rkB/s`, `wkB/s`, `bi`, `bo`. Opóźnienie (latency) to czas, jaki zajmuje pojedyncza operacja — `await`, `r_await`, `w_await`. To są dwie niezależne osie: dysk może mieć wysoką przepustowość i wysokie opóźnienie naraz (dużo danych, ale każda operacja czeka długo w kolejce), albo niską przepustowość i niskie opóźnienie (mało ruchu, każde żądanie obsłużone od razu). Aplikacja bazodanowa zwykle bardziej cierpi na wysokie opóźnienie pojedynczych zapisów niż na ograniczoną przepustowość — stąd `await` jest zwykle ważniejszą kolumną w `iostat -x` niż same `kB/s`.



Żądanie odczytu lub zapisu przechodzi przez kolejkę urządzenia, zanim dysk zwróci odpowiedź. (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: nasycenie urządzenia.** `%util` w `iostat` mierzy procent czasu, w którym urządzenie blokowe miało *choć jedno* żądanie w toku — nie to, jak *mocno* było obciążone. Dla klasycznego dysku talerzowego (HDD), który fizycznie obsługuje jedną operację naraz (jedna głowica, jeden talerz w danym momencie), 100% `%util` rzeczywiście oznaczało pełne nasycenie — nic więcej się nie zmieści. Dysk SSD i napęd NVMe potrafią obsługiwać dziesiątki albo setki operacji równolegle (dzięki kolejkom sprzętowym i wielu kanałom pamięci flash), więc `%util` na poziomie 100% może oznaczać tylko, że w kolejce zawsze było przynajmniej jedno żądanie — sam dysk może mieć jeszcze mnóstwo zapasu. Do oceny prawdziwego nasycenia nowoczesnego nośnika lepiej służy `aqu-sz` (średnia długość kolejki) w zestawieniu z `await` — rosnąca kolejka przy rosnącym opóźnieniu to prawdziwy sygnał, że urządzenie nie nadąża, niezależnie od tego, co pokazuje `%util`.

Nazwa pakietu, z którego pochodzą `iostat`, `sar`, `pidstat` i `mpstat`, to **`sysstat`** — identyczna na RHEL/Rocky, Debianie/Ubuntu i SLES. Sam `sar` bez uruchomionej i włączonej usługi `sysstat` (którą uruchamia się `systemctl enable --now sysstat`) nie zbiera żadnych danych do plików `/var/log/sa/saDD` — zapytanie o wczorajszą godzinę 3 w nocy zwróci wtedy błąd albo pustkę, nie dlatego że coś jest zepsute, tylko dlatego że nikt nie kazał systemowi tych danych w ogóle rejestrować.

Krok po kroku

1. Odpal `vmstat` z liczbą powtórzeń i zignoruj pierwszy wiersz.

```
vmstat 2 6
```

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
```

```
 2  1  524288 1980000 184000 5180000    0    0    45    12  120   90 12  4 60 24  0
 1  0  524288 1975000 184000 5182000    0    0     4     1   52   30  5  2 92  1  0
 1  0  524288 1971000 184000 5183000    0    0     6     0   50   28  4  1 94  1  0
```

Pierwszy wiersz (`wa` 24%) wygląda niepokojąco — ale to jest średnia od startu systemu, mieszająca się z bieżącym stanem. Kolejne wiersze (`wa` 1%) pokazują rzeczywistą, aktualną sytuację: system jest spokojny.

2. Jeśli `wa` w kolejnych próbkach faktycznie jest wysokie, sprawdź dyski.

```
iostat -xz 2 5
```

```
Device     r/s    w/s   rkB/s   wkB/s  r_await  w_await  aqu-sz  %util
sda       12.00  420.00  256.0  51200.0    3.20    180.50    8.40   98.00
```

`w_await` 180 ms to bardzo dużo (dobry dysk SSD to zwykle pojedyncze milisekundy), a `aqu-sz` 8.4 pokazuje realnie zapchaną kolejkę — to jest prawdziwe nasycenie, nie tylko wysoki `%util`.

3. Sprawdź, kto konkretnie pisze tak dużo.

```
pidstat -d 2 3
```

```
10:30:01    UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
10:30:03    999      1203      0.00  48200.00      0.00      18  postgres
```

Proces `postgres` (PID 1203) generuje niemal cały zapis — to zawęża dalsze dochodzenie do konkretnej bazy danych i konkretnego zapytania/operacji, zamiast zgadywania po całym systemie.

4. Jeśli potrzebujesz porównania z wczoraj, sięgnij po dane historyczne.

```
sar -d -f /var/log/sa/sa16 -s 03:00:00 -e 04:00:00
```

`-s`/`-e` zawężają wynik do konkretnego przedziału godzin z danego dnia — bez tego `sar -f` wypisze wszystkie próbki z całego dnia.

Częste błędy

Ocena stanu systemu na podstawie pierwszej linii `vmstat` albo `iostat`.

```
$ vmstat 2 3
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
```

```
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0 524288 2150000 184000 5210000    2    1    18    22   55   34  8  3 40 49  0
 0  0 524288 2148000 184000 5211000    0    0     4     1   50   30  5  2 92  1  0
```

`wa` 49% w pierwszej linii wygląda jak poważny problem z dyskiem — administrator budzi zespół storage w środku nocy. Druga linia (`wa` 1%) pokazuje prawdziwy, spokojny stan. Naprawa: zawsze ignoruj pierwszy wiersz `vmstat` i `iostat` — to średnia od startu systemu, nie bieżący stan.

Uruchomienie `sar` bez wcześniej włączonej usługi i oczekiwanie na dane historyczne.

```
$ sar -r -f /var/log/sa/sa15
Cannot open /var/log/sa/sa15: No such file or directory
```

Plik nie istnieje, bo usługa `sysstat` nigdy nie była włączona na tym serwerze — `sar` bez własnego demona zbierającego dane nie ma nic do pokazania z przeszłości, mimo że polecenie `sar -u 2 5` (na żywo) działałoby bez zarzutu. Naprawa: `systemctl enable --now sysstat`, poczekać na pierwsze zebrane próbki (domyślnie co 10 minut) i dopiero wtedy oczekiwać danych historycznych.

Alarmowanie przy `%util` 100% na dysku NVMe.

```
Device     r/s    w/s   rkB/s   wkB/s  r_await  w_await  aqu-sz  %util
nvme0n1  850.00 620.00 108000.0 79200.0    0.18     0.22    0.95   99.80
```

`%util` prawie 100% wygląda alarmująco, ale `r_await`/`w_await` poniżej 1 ms i `aqu-sz` blisko 1 pokazują, że urządzenie obsługuje operacje niemal natychmiastowo — nie jest realnie przeciążone, po prostu w każdej chwili pomiaru miało jakieś żądanie w locie, co dla NVMe jest normalne przy stałym, umiarkowanym ruchu. Naprawa: na SSD/NVMe oceniaj nasycenie przez `await` i `aqu-sz`, nie przez `%util`.

Brak pakietu `sysstat` i próba użycia `iostat`/`sar`/`pidstat` wprost z instalacji minimalnej.

```
$ iostat -x 2
bash: iostat: command not found
```

Na instalacji minimalnej (`minimal install` przy RHEL/Rocky, obraz `slim` kontenerów) tych narzędzi po prostu nie ma domyślnie. Naprawa: `dnf install sysstat` (RHEL/Rocky), `apt install sysstat` (Debian/Ubuntu) albo `zypper install sysstat` (SLES) — nazwa pakietu jest identyczna na wszystkich trzech.

Porównywanie `pidstat -d` z `iostat` i oczekiwanie identycznych liczb.

`iostat` liczy ruch na poziomie urządzenia blokowego, po uwzględnieniu m.in. scalania żądań (`merge`) przez planistę I/O jądra. `pidstat -d` liczy ruch zgłoszony przez konkretny proces, przed ewentualnym scaleniem. Suma `kB_wr/s` wszystkich procesów z `pidstat -d` bywa więc zbliżona, ale

nie identyczna co do bajta z `wkB/s` z `iostat` dla tego samego urządzenia w tym samym interwale — to nie jest błąd pomiaru, tylko dwa różne poziomy obserwacji tego samego ruchu.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Instalacja `sysstat` wymaga uprawnień administratora.

Ćwiczenie 1 — złapanie pierwszej próbki na gorącym uczynku

```
uptime
vmstat 2 3
```

Porównaj `wa` w pierwszym wierszu `vmstat` z tym, ile wynosi uptime systemu. Dlaczego na maszynie działającej od wielu dni pierwszy wiersz `vmstat` bywa bardziej „wygładzony" (bliższy zeru w skrajnych kolumnach) niż na maszynie uruchomionej pięć minut temu i już mocno obciążonej?

Odpowiedź

Bo pierwszy wiersz to suma zdarzeń od startu systemu podzielona przez cały czas działania (uptime). Im dłużej system działa, tym bardziej krótkotrwałe szczyty obciążenia rozmywają się w mianowniku — godzina intensywnego I/O podzielona przez trzy tygodnie uptime da znikomy wynik procentowy. Na świeżo uruchomionym systemie, który od razu trafił pod duże obciążenie, ten sam szczyt podzielony przez kilka minut uptime da znacznie wyższy, bardziej widoczny wynik w pierwszym wierszu.

Ćwiczenie 2 — instalacja i pierwsze dane historyczne z `sysstat`

```
sudo dnf install -y sysstat   # albo apt/zypper, zależnie od dystrybucji
sudo systemctl enable --now sysstat
sleep 660   # nieco ponad 10 minut, żeby zebrać pierwszą próbkę
sar -u -f /var/log/sa/sa$(date +%d)
```

Jeśli 660 sekund to za długo na ćwiczenie, sprawdź w `/etc/cron.d/sysstat` albo w jednostce systemd timer, jak często faktycznie działa zbieranie, i skróć oczekiwanie do najbliższego zaplanowanego uruchomienia.

Odpowiedź

Po zebraniu pierwszej próbki `sar -u -f /var/log/sa/saDD` (gdzie `DD` to bieżący dzień miesiąca) powinien pokazać przynajmniej jeden wiersz z danymi CPU, ze znacznikiem czasu odpowiadającym momentowi uruchomienia zbierania. Zanim usługa zdąży zebrać choć jedną próbkę, plik `saDD` z bieżącego dnia może w ogóle nie istnieć — to nie jest błąd konfiguracji, tylko po prostu brak jeszcze jakichkolwiek danych do zapisania.

Pytania kontrolne

1. `vmstat 2 5` pokazuje w pierwszym wierszu `wa` równe 35%, a w kolejnych czterech — 1-2%. Który wiersz opisuje bieżący stan systemu?

Odpowiedź

Kolejne cztery wiersze. Pierwszy to średnia od startu systemu, nie bieżąca próbka — należy go zawsze ignorować przy ocenie aktualnego stanu.

2. Dysk NVMe pokazuje `%util` 100%, ale `await` poniżej 1 ms. Czy dysk jest przeciążony?

Odpowiedź

Niekoniecznie. `%util` mierzy tylko udział czasu, w którym urządzenie miało choć jedno żądanie w toku — dla nośnika obsługującego wiele żądań równolegle (jak NVMe) to nie jest miara nasycenia. Niski `await` (czas oczekiwania na operację) i niska `aqu-sz` (kolejka) świadczą, że urządzenie odpowiada praktycznie natychmiast, więc realnie nie jest przeciążone.

3. Chcesz zobaczyć obciążenie CPU serwera o 3 w nocy sprzed tygodnia. Jakie dwa warunki muszą być spełnione, żeby `sar -f` w ogóle mógł na to odpowiedzieć?

Odpowiedź

Po pierwsze usługa `sysstat` musiała być włączona (`systemctl enable --now sysstat`) i działać nieprzerwanie od tamtego czasu. Po drugie plik z danymi z tamtego dnia (`/var/log/sa/saDD`) musi jeszcze istnieć — domyślna retencja `sysstat` bywa ograniczona (zwykle kilka tygodni),

więc bardzo stare dane mogą być już rotowane i usunięte.

4. `iostat` i `pidstat -d` liczone w tym samym interwale na tym samym serwerze dają zbliżone, ale nie identyczne liczby zapisu na sekundę. Czy to błąd?

Odpowiedź

Nie. `iostat` liczy ruch po stronie urządzenia blokowego, już po ewentualnym scaleniu żądań przez planistę I/O jądra (widoczne w kolumnach `rrqm/s`/`wrqm/s`). `pidstat -d` liczy ruch zgłoszony przez procesy, przed scaleniem. Niewielka rozbieżność między tymi dwoma poziomami obserwacji jest normalna i nie wskazuje na błąd pomiaru.

Powiązane: Pamięć naprawdę: free i /proc/meminfo · SWAP: czym jest, kiedy czyścić, kiedy nie ruszać · Obciążenie systemu: load average, uptime, dmesg · ps — zdjęcie procesów

Źródła: `man 8 vmstat` · `man 1 iostat` · `man 1 sar` · `man 1 pidstat` · `man 8 iotop` · pakiet `sysstat` (dokumentacja `/usr/share/doc/sysstat*`) · wewnętrzna notatka „Monitorowanie zasobów: free, vmstat, iostat, sar"

Revision #16
Created 2026-09-17 21:43:30 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:16 UTC by Tomasz Konieczny