Testowanie Jądra Linuxa

"Given enough eyeballs, all bugs are shallow"

Eric Raymond

”Tylko skąd wziąć te przyglądające się oczy?”

Michał Grabowski
Piotr Szczepański
Paweł Zuzelski

W temacie:

- Wprowadzenie do testowania
  - Podziały
  - Metodologie
- Testy Poprawnościowe
  - Pokrywanie kodu
- Testy Wydajnościowe

Czemu trzeba testować jądro?

- Gwarancja, że kolejne wersje jądra będą dobrze działać
- Wcześniejsze wykrycie błędu zaoszczędzi nam wiele czasu
- Wielkość jądra i fakt rozwijania go przez wielu użytkowników, może przyczyniać się do błędów
- Ewentualne błędy jądra mogą wywołać daleko idące konsekwencje
- Sumienie użytkownika Open Source

Skąd błędy?

Przejście z wersji jądra 2.6.27 do 2.6.28 potrzebowało:

- 70 dni prac
- 8995 commitów
- edycji 11076 plików
  - 975 468 lini dodano
  - 489 931 usunięto

Podstawowy podział testów:

- -poprawnościowe (czy system zachowuje się poprawnie dla różnych danych)
- -wydajnościowe (jaki jest czas działania systemu dla różnych danych)
- -długotrwałego działania (jak system zachowuje się w przypadku dłuższego działania)
- -obciążeniowe i przeciążeniowe (dla jakich obiążeń system działa poprawnie, jak zachowuje się dla większych)

Testy poprawnościowe

- Tworzenie scenariuszy testowych
- Analiza statyczna kodu
  - Przeglądanie i analizowanie kodu
- Weryfikacja formalna
- Testy dynamiczne

Testy wydajnościowe

- Przeprowadzanie duzej liczby testow, usrednianie wynikow
- Testy powinny byc przeprowadzane w niezmiennym srodowisku
- Wyniki musza byc uzyskiwane w sposob automatyczny, powtarzalny
- Ograniczenie czynników zewnętrznych
- Testy długotrwałe

Podział testów

- Szklanoskrzynkowe
  - Testowanie oprorgamowania z uwzględnieniem kodu
- Czarnoskrzynkowe
  - Traktowanie oprorgamowania jako zamkniętej skrzynki reagującej na dane wejściowe

Testy regresyjne

- Zazwyczaj wykonywanie testów regresyjnych związane jest z ponownym uruchomieniem zestawu testów, które wcześniej kończyły się poprawnie. Ma ono na celu ujawnienie potencjalnych problemów powstałych na skutek dokonanych zmian.

Projektowanie testów

- Pokrycie wszystkich gałęzi kodu
- Szczególne uwzględnienie przypadków brzegowych
- Sprawdzanie współbieżności
- Zachowanie przy błędach systemowych

Projektowanie testów

- Obciążanie zasobów systemowych (procesor, pamięć, sieć)
- Pokrycie kodu, weryfikuje ile procent naszego programu przebrnęło przez testy

  Oczywiście gdy 100% kodu zostanie wykonane,

  nie mamy żadnej gwarancji poprawności!

- Finalny, długotrwały test obciażeniowy

Automatyzacja testów

- Główne cele:
  - Równoległość prac
  - Jak najszybszy przechwyt błędów
  - Jak najwygodniejsze udostępnianie wyników
  - Zastąpienie czynnika ludzkiego maszynowym

- Platforma autotest :
  - # svn checkout svn://test.kernel.org/autotest/trunk autotest
- Dzielenie się wynikami
  - Test.kernel.org

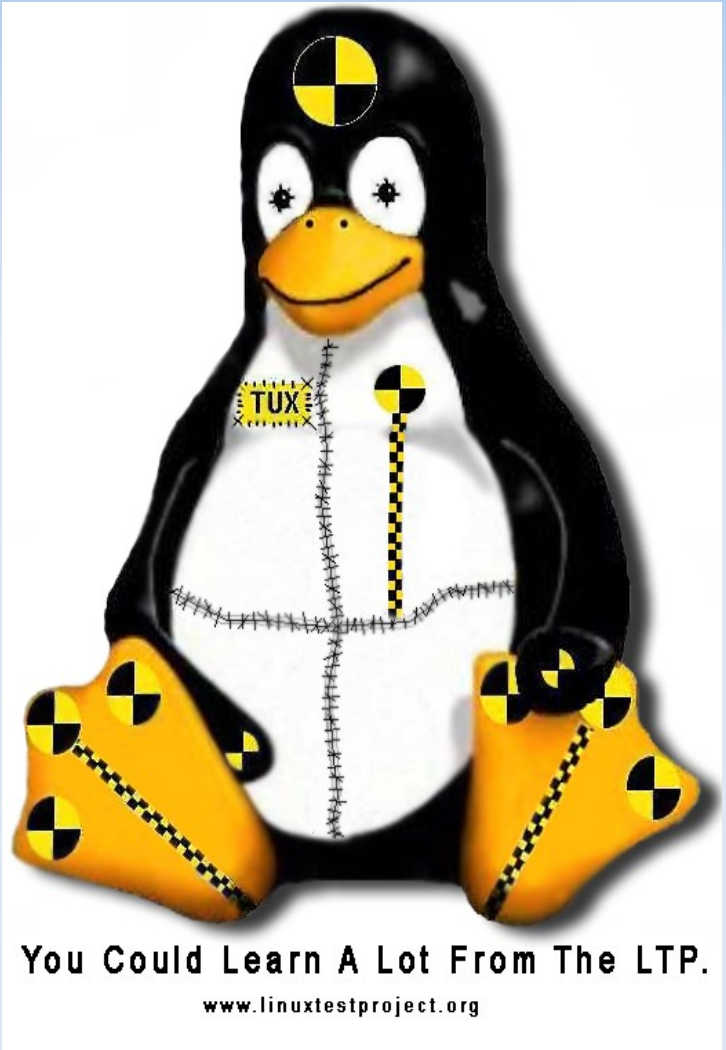
Automatyczne testy

- uruchamianie testów
- znajdowanie błędu
- klasyfikacja błędu
- przekazanie błędu developerowi
- śledzenie błędu
- próba naprawienia
- naprawa
- nowa notka do teamu testujacego

Automatyczne testy- zalety

- konsekwentność - gwarancja, że kolejne testy zostaną przeprowadzone tą samą drogą
- upowszechnianie wiedzy - wiedza jak przeprowadzać dane testy jest nie trzymana i jednej osoby ale w systemie
- udostępnianie - proste sposoby udostępniania testów i ich wyników
- odtwarzanie - tworzenie scenariuszy testowych, gotowych do odtworzenia i pokazania błędu

Linux Test Project



Linux Test Project

- Zestaw testów regresyjnych jądra Linuksa
- Projekt zapoczątkowany przez SGI, obecnie rozwijany przez IBM
- Nowe wydanie publikowane raz w miesiącu
- Autorzy publikują wyniki testów dla wybranych wresji kernela/glibc

Struktura LTP

- Wiele niezależnych małych programów do testowania poszczególnych elementów systemu, zgodnych z określonym szablonem. Aktualnie zawiera ponad 3000 testów.
- Programy automatyzujące wykonywanie testów.

Narzędzia LTP

- Makefile
  - Reguły do kompilacji I instalacji zestawu testów
  - Reguła do uruchomienia wszystkich testów
- runltp
  - Skrypt do uruchamiania grup testów wykorzystujący pan
- pan
  - Skrypt do uruchamiania poszczególnych testów

Testy w ramach LTP

- Grupy testów
- DOTS – Database Opensource Test Suite
- Ballista – testy wywołań systemowych
- Command – testy podstawowych komend
- Kdump – testy mechanizmu kdump
- Kernel – testy jądra
- Network – testy sieci

Testowanie zgodności ze standardami

- POSIX - Portable Operating System Interface
  - Zestaw standardów określonych przez IEEE definiujących interfejs uniksa
- SUSv3
  - Otwarty odpowiednik POSIX
- LSB
  - Linux Standard Base
  - LSB 2.0.1 zatwierdzony jako standard ISO 23360

Testowanie zgodności ze standardami

- Open POSIX Test Suite
  - zestaw testów badających zgodność dystrybucji ze standardami POSIX i SUSv3
  - Zapoczątkowany jako osobny projekt, później włączony do LTP
  - Projekt martwy od 2005 roku
- LSB test suite
  - Własny zestaw testów

Automatyczne testy

- Kerncomp – automatyczne testy budowania instalacji I bootowania kernela dla architektury IA64
- Automatyczne testu budowania Linuksa dla architektury ARM

  http://armlinux.simtec.co.uk/faq.html
- Testy cross-kompilacji Linuksa http://l4x.org/k/

GNU Gcov

- Narzędzie analizujące wykonywujący się kod twojego programu, daje informacje o:
  - Jak często wykonują się poszczególne linie
  - Które linia kodu się aktualnie wykonuje

Często narzędzie to urzywa się wraz z Gprof, które służy między innymi do wykrywania ”wąskich gardeł” naszego prorgamu.

• Ile czasu wykonania przypada na poszczególne sekcje

GNU Gcov

Działa wyłącznie z kompilatorem GNU CC.

Prosty przykład użycia:

- Kompilujemy, z odpowiednimi flagami
  gcc -fprofile-arcs -ftest-coverage test.c -o test
- Odpalamy nasz prorgam
- Analiza wyników
  gcov test.c
- Odczyt danych z pliku test.c.gcov

polecamy man gcov ;]

GNU Gprof

Użycie:

- Kompilacja:
  gcc -pg test.c -o test
- Uruchomienie
  gprof test

wybrane pola ”płaskiego profilu”:
%time – procent czasu spędzony w funkcji
%calls – ilość wywołań
%cumulative/self seconds - czas spędzony w funkcji wraz z wywołaniami/ bez wywołań

man gprof równie dobry

GNU Lcov

Frontend dla programu gcov tworzony w ramach projektu LTP . Może współpracować zarówno z kernelem (takim przypadku dane gcov są dostępne w katalogu /proc/) jak i z aplikacjami przestrzeni użytkownika.

Testy Wydajnościowe

- Testy Wydajnościowe (benchmarki, testy wzorcowe): ”testują różne charakterystyki sprzętu komputerowego i oprogramowania”
  - Benchmark Syntetyczny
  - Benchmark Aplikacyjny
- Testy Wydajnościowe: "przeprowadzane w celu oceny stopnia spełnienia wymagań wydajnościowych przez system"

”Inżynieryjność” programowania

- ”Inżynieryjne” podejście do tworzenia dużych systemów informatycznych
  - Godzimy się na błędy
  - Nie znamy optymalnych złożonościowo algorytmów
  - Znamy jedynie algorytmy aproksymujące rozwiązanie

Wydajność

- "Wydajność [...] wyraża ilość pracy wykonanej w określonym przedziale czasu"
- Wydajność w odniesieniu do jednostki czasu
- ze względu na skomplikowanie systemu tak naprawde mierzymy tylko konkretną wartość tj. wydajność w odniesieniu tylko i wyłącznie do zadania testowego.

Testy nie zawsze uwzględniają skoplikowane zależności pomiędzy elementami systemu.

Dlaczego mierzymy?

- Dla Sportu
  - Żeby sie pochwalić (speedtest)
  - Bo liczby są fajne
- W celu dokonania wyboru
  - Zakupu oprogramowania/sprzetu;
  - Lepszego algorytmu;
  - Oszacowania wydajności porządanej dla określonego zastosowania;
- Diagnostycznie
  - Znalezienie wąskich gardeł
  - Znajdywanie awarii sprzetu/oprogramowania
  - Znajdywania błędów

Benchmark syntetyczny

- Z reguły działające w petli wykonanie wielokrotnie okreslonej czynności.
- Przykładowe wskaźniki(w Lmbench)
  - Context switching in ms
  - Fork, exec process in ms
  - Communication latencies in ms(pipe, tcp, ...)
  - File delete in ms
  - Communication bandwidts in MB/s(pipe, tcp, ...)
  - Page fault in micros

Benchmark aplikacyjny

- Przykłady Benchmarków Aplikacyjnych
  - kodowania filmu do formatu XviD
  - uruchomienie systemu
  - kompilacja jdra linux
  - pakowanie plików(ZIP)
  - wczytywanie poziomu gry
  - renderowanie obiektu 3D
  - liczba klatek w grze na sekunde

Pomiar i jego wiarygodność

- Zakładamy, że pomiar jest powtarzalny i niezależny od otoczenia
- Chcemy ominąć "oszustwa" komputera w rodzaju cache'owania
- Uśrednienie wielu pomiarów nie musi być reprezentatywne
- Zakładamy, że nie zmieniają się warunki fizyczne(np. temperatura rdzenia)
- Używamy przy pomiarach tej samej wersji wersjach oprogramowania (np. kompilacja jądra przy użyciu gcc, renderowanie w 3Dstudio, benchmark 3DMark);
- Pomiar może być zależny od niektórych charakterystyk systemu jak np. fragmentacja systemu plików podczas kompilacji.
- Pilnujemy żeby nie "dociążając" komputera podczas testów(np. instalacja aktualizacji podczas testów czasu dostępu dysku)
- Testowanie na maszynie virtualnej: fatalny pomysł;

Interpretacja rezultatów

- Rezultat dotyczy "przykładowych danych"
- Pomiar jest obarczony błędem - choć często zminimalizowanym gdyż operacje na PC są silnie skwantowane
- Pomiar nie musi mieć typowego rozkładu statystycznego(np. rozkład Gaussa)
- Obecność anomalii
- Producenci sprzętu optymalizują go pod kątem benchmarków syntetycznych
- O szybkości działania(wyniku testu) nie musi decydować tylko badany element(wąskie gardła)
- Nie zawsze pomiar podstawowej wartości daje pełny obraz o danym sprzącie/programi - np. napędy optyczne... obciążają procesor
- Benchmarki aplikacyjne wydają się wiarygodniejsze od syntetycznych - lepiej symulują pracę systemu ale dają mniej szczegółowe informacje

Testy wydajnościowe zgodności ze specyfikacją wydajności

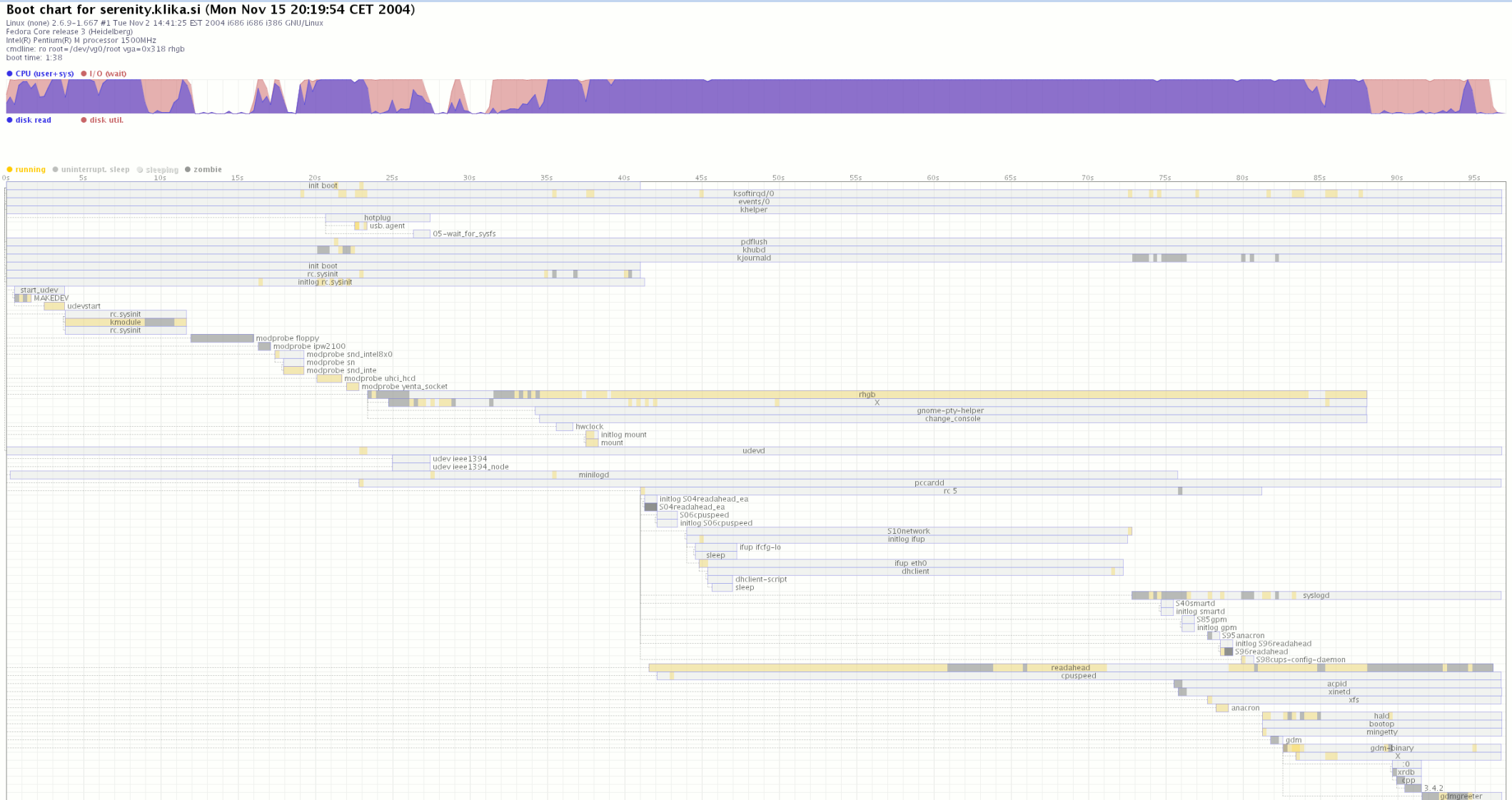
- Mamy wskazany w specyfikacji wskaźnik wydajności oraz wartość wymaganą
- Symulujemy realne użycie systemu np:
  - uruchomienie równolegle konsol via ssh
  - uruchomienie równolegle wielu procesów "coś" robiących
- Pomiar wskaźników:
  - "w stresie" - przy obciążeniu bliskim maksymalnemu
  - w przeciążeniu - przy obciązeniu wyższym od maksymalnego
  - mogą pojawić sie wówczas błędy w działaniu aplikacji lub nieprzewidywane zachowania

Bootchart

- Owen Taylor wrote:
- "Ideally, system boot would involve a 3-4 second sequential read of around 100 megabytes of data from the hard disk, CPU utilization would be parallelized with that, and all queries on external systems would be asynchronous ... startup continues and once the external system responds, the system state is updated. Plausibly the user could startwork under 10 seconds on this ideal system.
- The challenge is to create a single poster showing graphically what is going on during the boot, what is the utilization of resources, how the current boot differs from the ideal world of 100% disk and CPU utilization, and thus, where are the opportunities for optimization. "
- http://www.redhat.com/archives/fedora-devel-list/2004-November/msg00447.html

Bootchart

- **Bootchart** (Ziga Mahkovec, 2005)
  - Fedora Core 3



Bootchart

- Fedora Core 3
- dostrzeżone problemy to miedzy innymi:
  - rhgb używa za dużo CPU (znaleziono i poprawiono bug)
  - modprobe floppy na laptopach bez stacji dyskietek zabiera 4 sekundy
  - ...

Bootchart

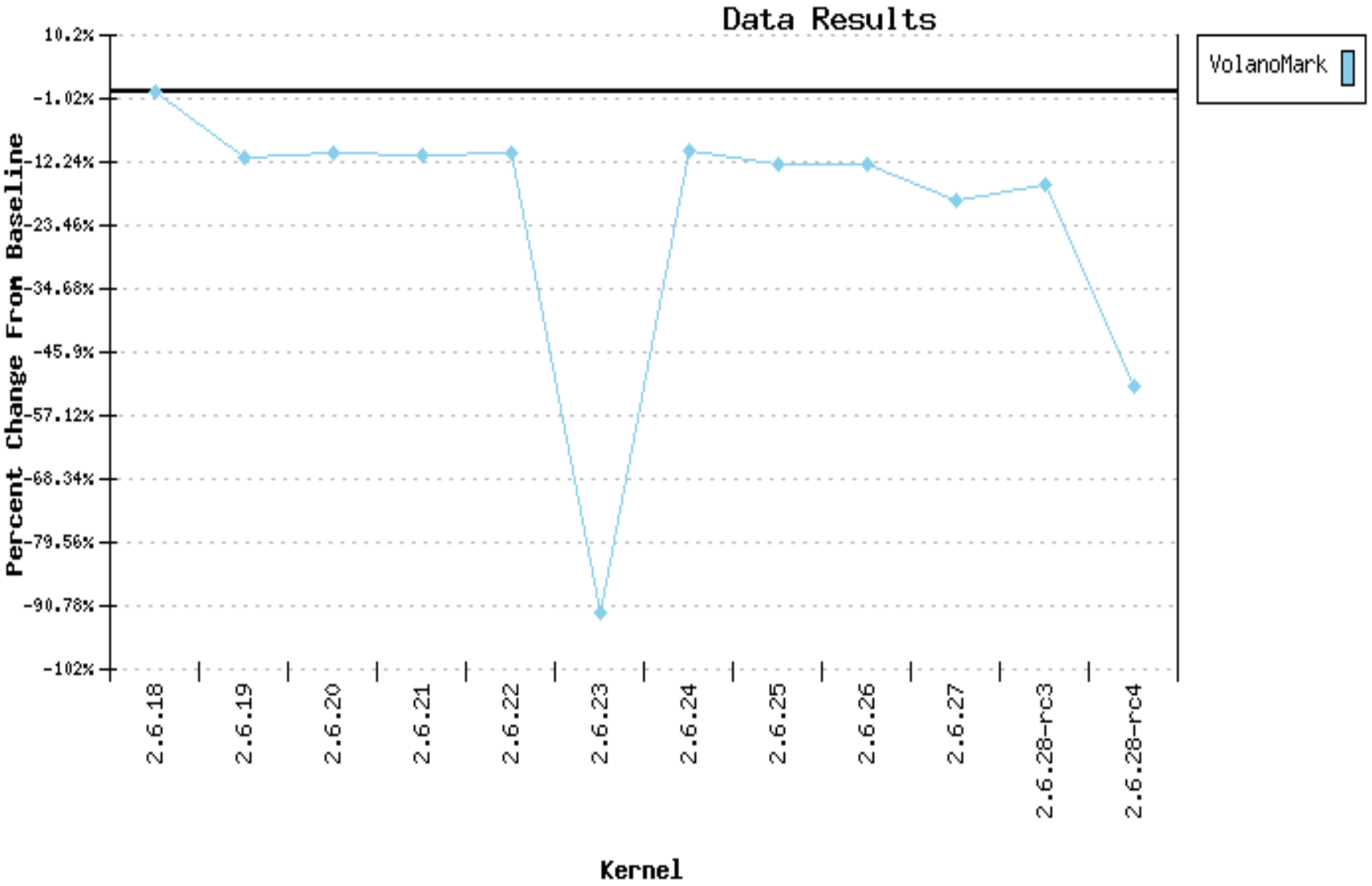
- Redukcja
- Z **1:38s**
- do **0:48s**

Linux Kernel Performance Project

- http://kernel-perf.sourceforge.net/
- Systematyczne benchmarki kolejnych dystrybucji jądra.
- Aplikacje z zestawu do testowania z przykładowymi wykresami przykladowe wykresy dla procesora:
- 2XP Xeon Core 2 Duo 2.66Ghz

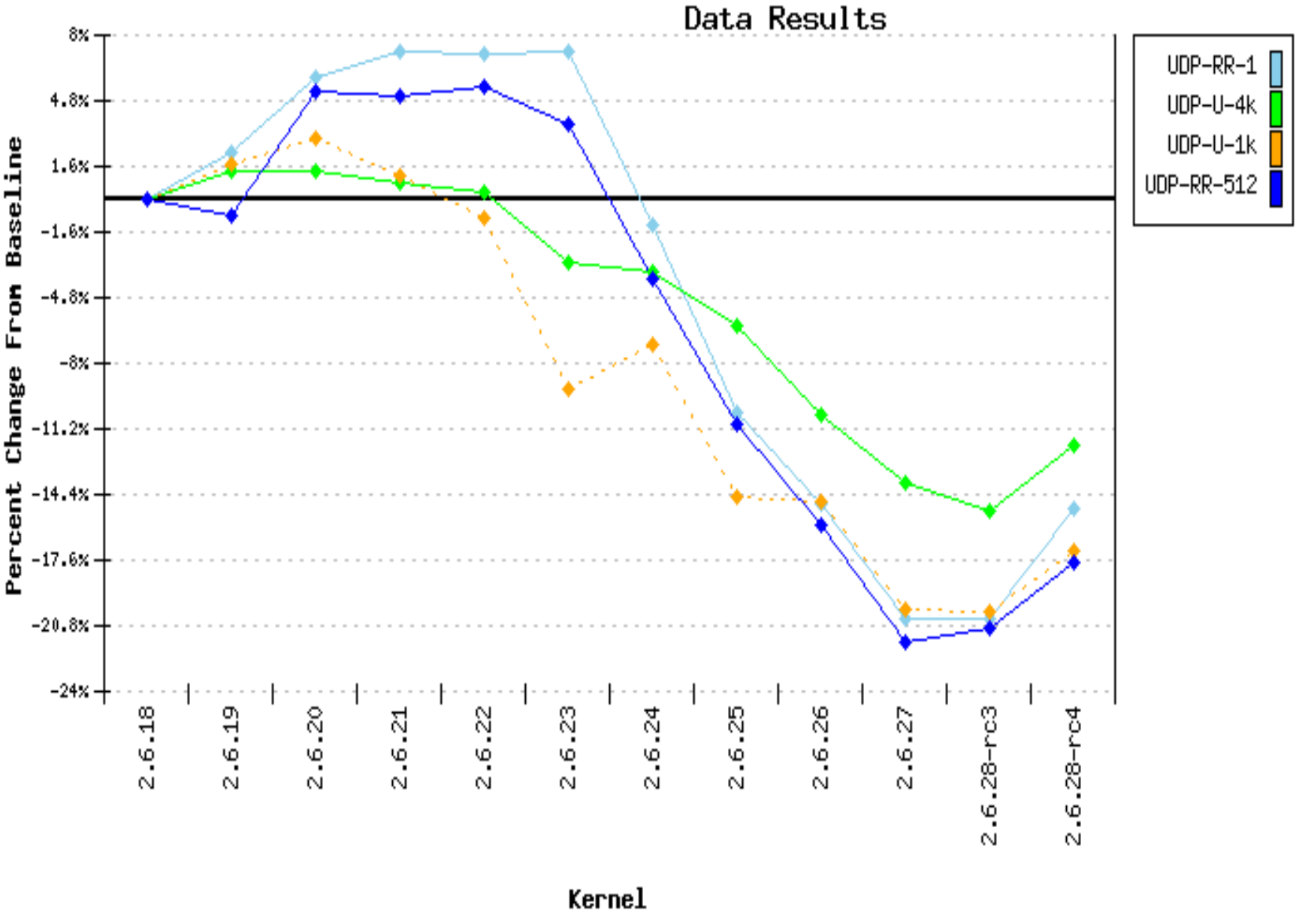
Linux Kernel Performance Project

- **Volanomark** (http://www.volano.com/benchmarks.html)
  - Wydajność komunikacj sieciowej i serwerów.



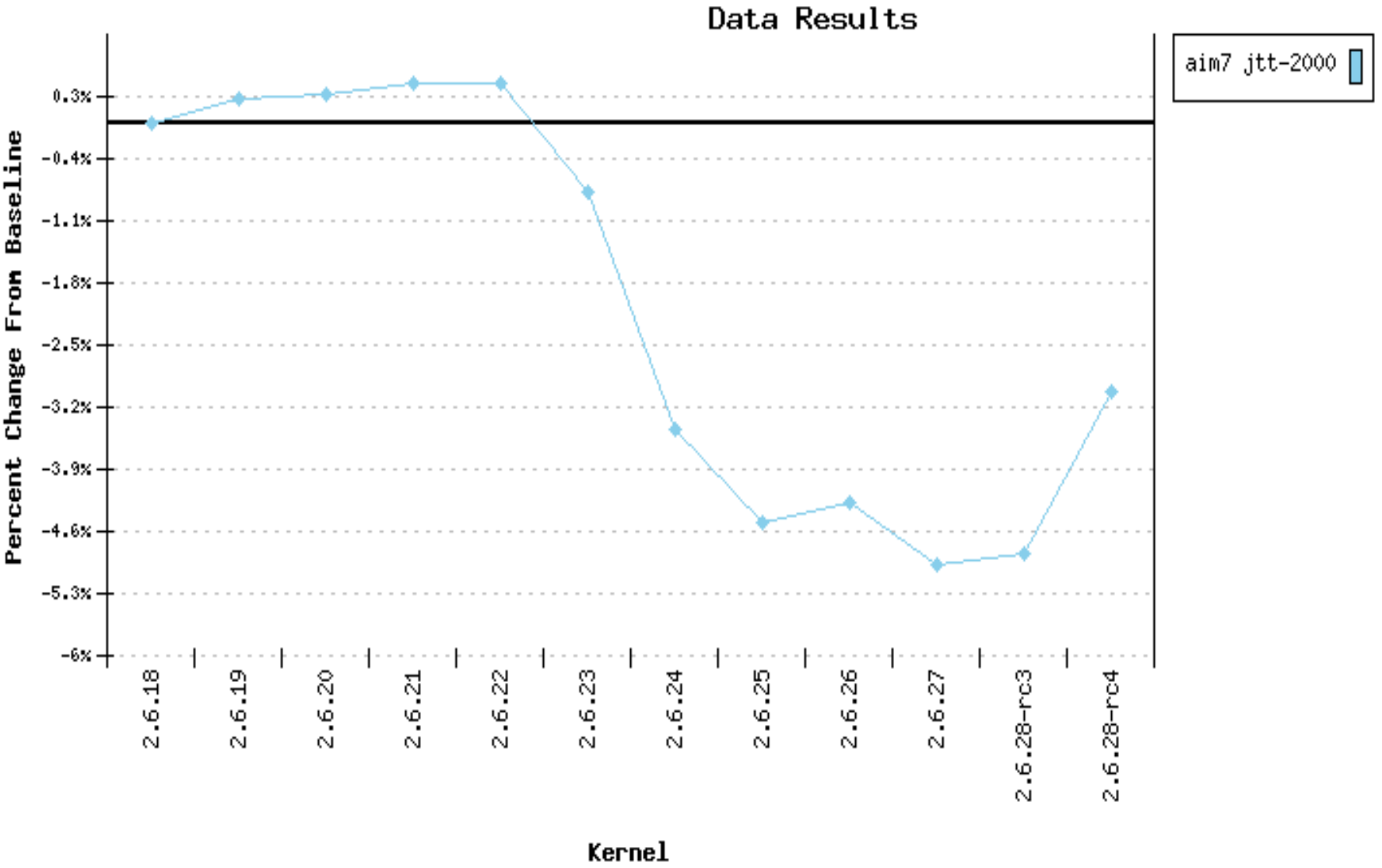
Linux Kernel Performance Project

- **Netperf** (http://www.netperf.org/netperf/NetperfPage.html)



Linux Kernel Performance Project

- **Aim** (http://sourceforge.net/projects/re-aim-7)
  - Wydajność sieci, pamięci i operacji dyskowych.

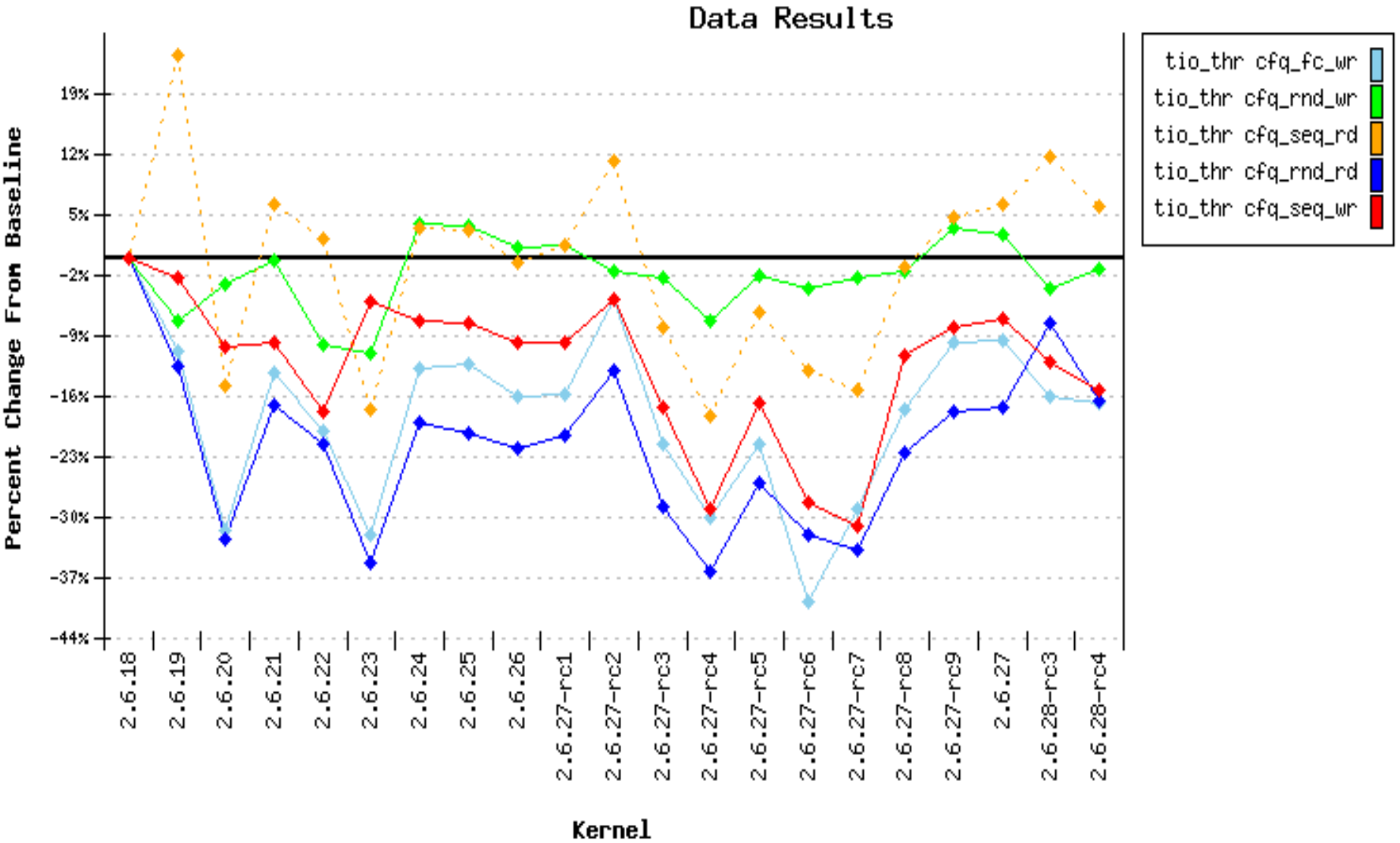


Linux Kernel Performance Project

- **lmbench** (http://www.bitmover.com/lmbench)
  - Niskopoziomowe testy wydajnościowe podstawowych zadań kernela
- **fileio** (http://sysbench.sourceforge.net)
  - Wydajość operacji wejścia-wyjścia
- **aiostress** (ftp://ftp.suse.com/pub/people/mason/utils/aio-stress.c)
  - Wydajność operacji wejścia-wyjścia, systemu plików
- **iozone** (http://www.iozone.org)
  - Wydajność systemu plików
- **dbbech/tbench** (ftp://samba.org/pub/tridge/dbench)
  - Emuluje pomiar wydajności serweru plików
- **vmregress** (http://www.skynet.ie/~mel/projects/vmregress)
  - Wydajność pamięci wirtualnej kernela

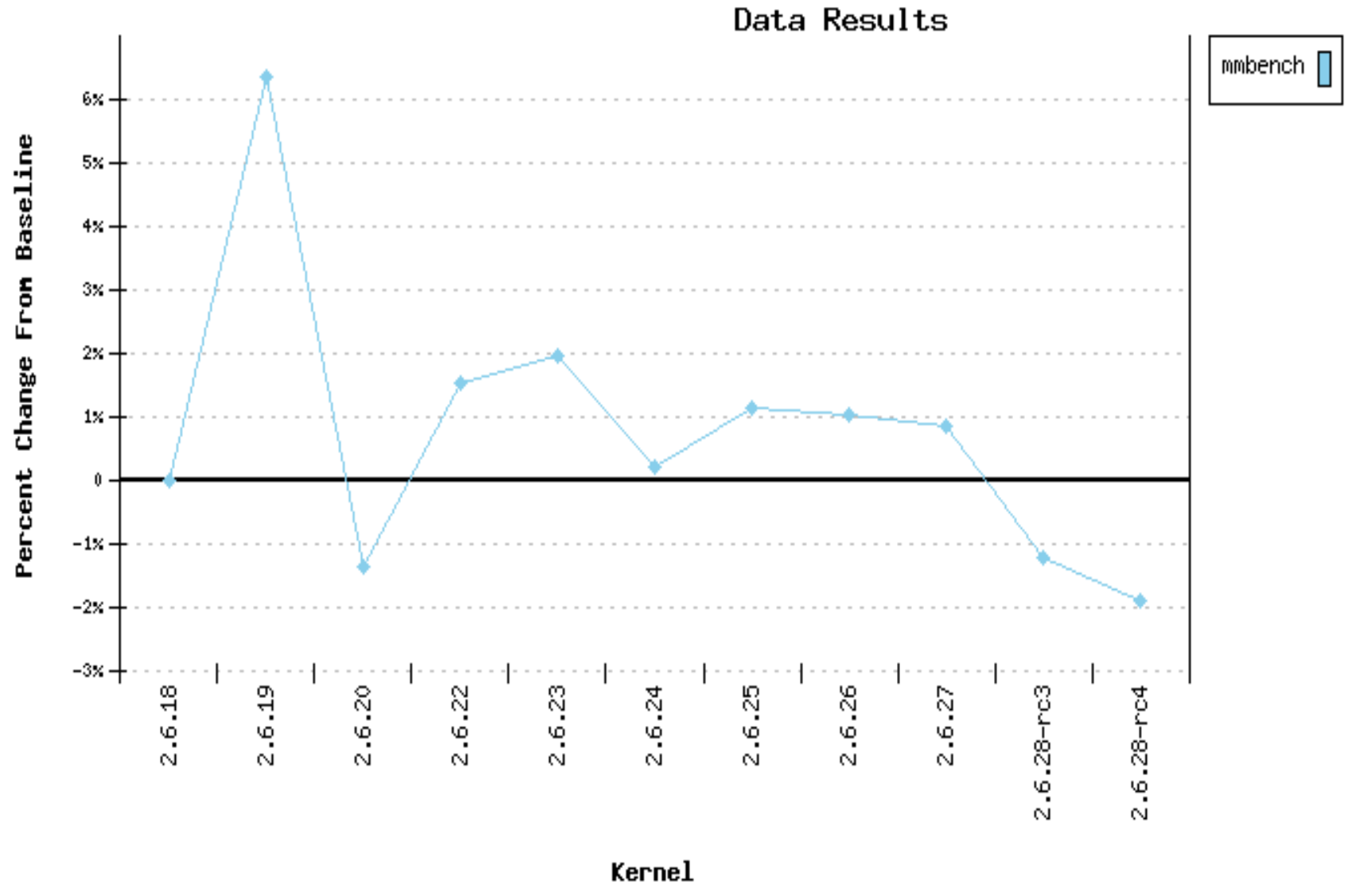
Linux Kernel Performance Project

- **Tiobench** (http://sourceforge.net/projects/tiobench/)
  - Wydajność operacji wejścia-wyjścia.



Linux Kernel Performance Project

- **Mmbench** (http://sourceforge.net/projects/tiobench/mmbench)
  - Wydajność zarządzania pamięcią.



Linux Kernel Performance Project

- oraz **JAVA***, **kernbuild**, **cpu-int**, **cpu-fp**