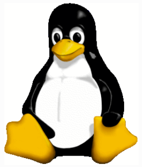
Linux Load Average



REVEALED

A Personal Journey
20,000 Lines Under the Shell

Neil Gunther
Performance Dynamics
Castro Valley, CA 94552
`njgunther@perfdynamics.com`

Motivation

load average: 0.02, 0.01, 0.00

Those 3 little numbers have always bugged me (still do)

I wanted to understand how they really work

I did some controlled experiments on Solaris and Linux

The data I produced needed interpretation

I didn't have access to UNIX [a] source code (it's all proprietary)

I used the hyperlinked Linux source on the web [1]

This is that story ...

[a] I won't assume you know any UNIX, even if you do

Depth Markers

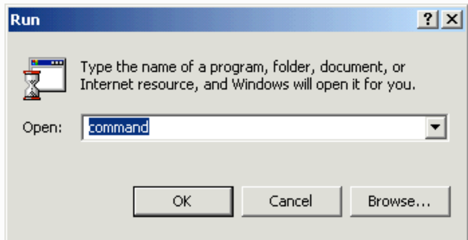
While diving 20,000 lines under the shell ...

1. What is Linux vs UNIX?
2. What is this so-called "load average"?
3. How does it work?
4. Is it any good for capacity planning?
5. Load average and the SCO law suit

What is Linux?

- A modularized operating system written in C
- Only the O/S nucleus or "kernel"
- Talks to the hardware: CPU, disks, RAM, NIC, etc.

But that ↑ is not even equivalent to this ↓



What is UNIX?

UNIX is an experiment that escaped from the (Bell) labs circa 1975

It has been mutating ever since!

AIX, HP-UX, MacOS X, Solaris, Unixware, ...

It's a descendent of the M.I.T. **Multics** project [2, Appendix B]

All UNIX cell lines are proprietary (just like Windows)

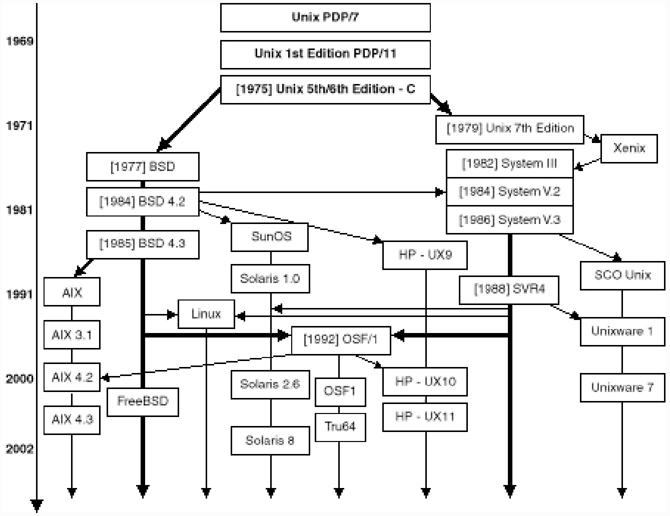
The key exceptions are:

- FreeBSD—licensed Berkeley UNIX derivative
- Linux—derived by Linus Torvalds from *Minix*, not UNIX

We'll return to the historical aspects on slide 36

In a word, there is no UNIX

UNIX Cell Lines



GNU is Neither Unix (nor Linux)

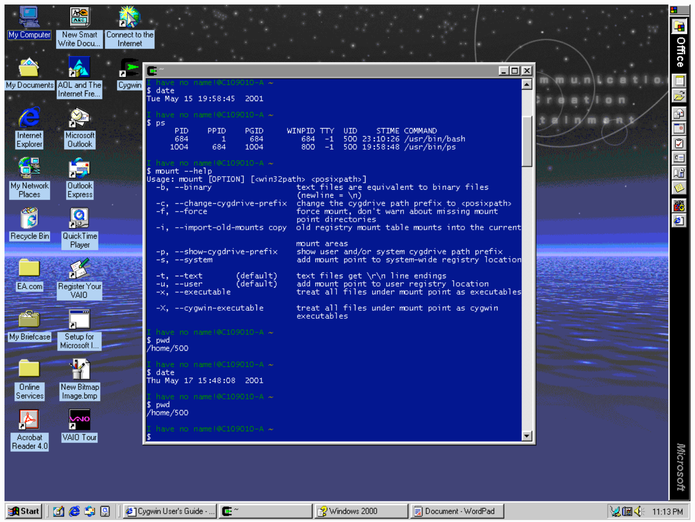
You need a *shell*; the thing outside the kernel which provides a command interface to it. Several standard UNIX shells:

- (Steve) Bourne shell—**sh**
- (David) Korn shell—**ksh**
- (Bill Joy's) C shell—**csh**
- Turbo C shell—**tcsh**
- (GNU [a]) Bash shell—**bsh**

Linux is usually packaged with the GNU *bash* (Bourne Again) shell [3] and other UNIX-like tools to make a complete UNIX-like operating system

[a] GNU is an example of a recursive acronym: *GNU is not Unix*

Windows Bashing (See [4])



What is a Load Average?

Embedded in ASCII output of certain UNIX commands:

```
[pax:~]% uptime
9:40am  up 9 days, 10:36,  4 users,  load average: 0.02, 0.01, 0.00
```

And Linux-specific commands:

```
[pax:~]% procinfo
Linux 2.0.36 (root@pax) (gcc 2.7.2.3) #1 Wed Jul 25 21:40:16 EST 2001 [pax]

Memory:      Total        Used        Free      Shared     Buffers      Cached
Mem:         95564       90252        5312       31412       33104       26412
Swap:        68508           0       68508

Bootup: Sun Jul 21 15:21:15 2002    Load average: 0.15 0.03 0.01 2/58 8557
...
```

Always three numbers. Why and what do they mean?

RTFM: Read the Free Manual

```
[pax:~]% man "load average"
No manual entry for load average
```

Oops! Let's try `man uptime`:

```
UPTIME(1)                    Linux User's Manual                    UPTIME(1)

NAME
       uptime - Tell how long the system has been running.

DESCRIPTION
       uptime gives a one line display of the following information.  The cur-
       rent time, how long the system has been running,  how  many  users  are
       currently  logged  on,  and the system load averages for the past 1, 5,
       and 15 minutes.
       ...
```

So, that's *why* there are three numbers, but what do they mean?

Let's Ask the Gurus

Tim O'Reilly and Crew [5, p.726]

> *The load average tries to measure the number of active processes at any time. As a measure of CPU utilization, the load average is simplistic, poorly defined, but far from useless.*

That's encouraging!

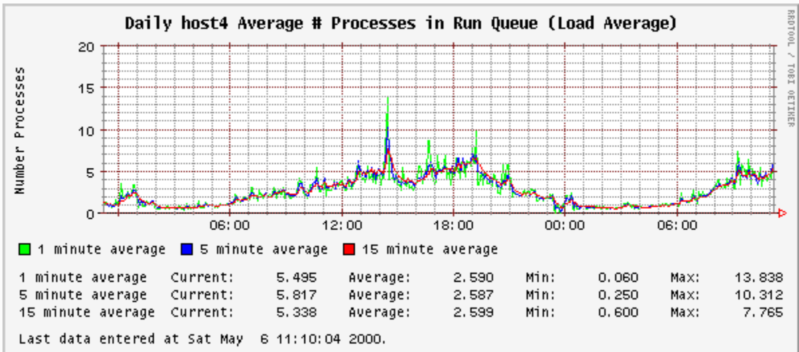
Adrian Cockcroft [6, p. 229]

> *The load average is the sum of the run queue length and the number of jobs currently running on the CPUs. In Solaris 2.0 and 2.2 the load average did not include the running jobs but this bug was fixed in Solaris 2.3.*

And it can be implemented incorrectly!! (cf. slide 36)

Graphical Load Average

ORCA [7] tool displays load averages as a **time series**



But this is just data, not information

So, What's an Average Load?

Back to our guru's ...

Tim O'Reilly et al.

> *What's high? ... Ideally, you'd like a load average under, say, 3, ... Ultimately, 'high' means high enough so that you don't need uptime to tell you that the system is overloaded. ... different systems will behave differently under the same load average. ... running a single cpu-bound background job can bring response to a crawl even though the load avg remains quite low.*

Eesh! ... This reads like it was written by a lawyer.

Nonetheless, their last sentence is 100% correct! (cf. slide 16)

Blair Zajac (author of ORCA tool [7])

> *If long term trends indicate increasing figures, more or faster CPUs will eventually be necessary unless load can be displaced. For ideal utilization of your CPU, the maximum value here should be equal to the number of CPUs in the box.*

This is probably the clearest statement of what the load average is intended to convey to a sys admin.

All three values of the load average should be kept in the neighborhood of the CPU configuration.

But is this really achievable?

The otherwise muddled statements arise because the load average is not your *average kind of average*. It's a **time-dependent** average or, as you'll soon see, it's a **damped** time-dependent average.

My Simple Experiment

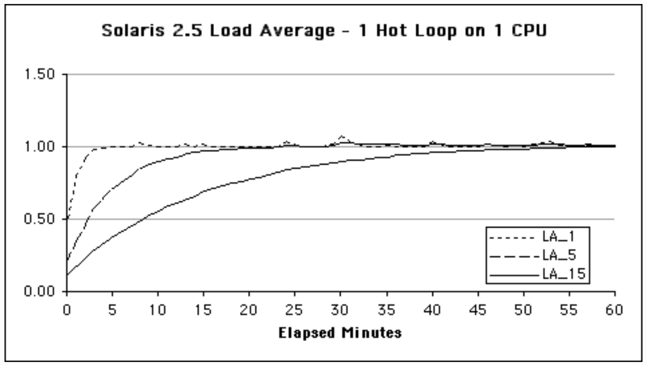
Two hot-loops initiated in background on single-CPU Linux box.

Test duration of 1 hour included two phases:

1. CPU pegged for 2100 seconds then processes killed.
2. CPU quiescent for the remaining 1500 seconds.

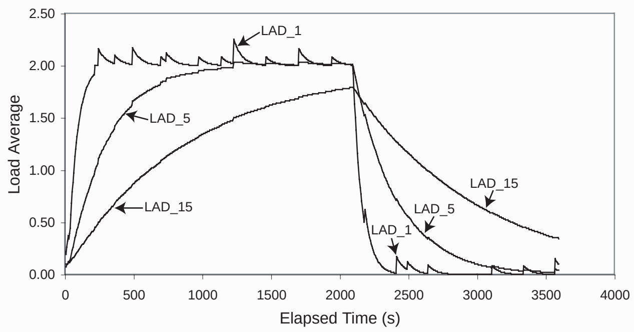
Perl script to sample all 3 load average metrics every 5 minutes from `uptime` output (see slide 9)

Earlier Experiment on Solaris

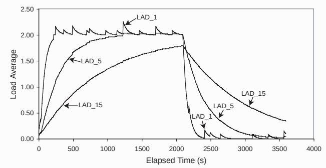


- Notice that the max load $\longrightarrow$ 1.0
- Some noise in 1-min LA_1 from system demons
- You're forgiven for thinking that load $\equiv$ CPU utilization

Experimental Data on Linux



Characteristic Profiles



1-minute load average (LAD_1) reaches value 2.0 at around 300 seconds

5-minute load average (LAD_5) reaches 2.0 around 1200 seconds

15-minute load average (LAD_15) would reach 2.0 at ∼ 4500 seconds, but I *killed* (a UNIX term) both processes at 2100 seconds

Resembles charging and discharging of an electrical capacitor

This what I mean by data is not the same thing as information

You would **never** see this kind of *data* in a million years of system data collection (cf. slide 12)

Diving Deeper

The question is, why does the experimental data resemble an RC circuit?

Why does it slowly rise and slowly decay?

To answer this question we need to examine the Linux kernel code

Depths of the Linux Kernel (See [1])

```
unsigned long avenrun[3];

static inline void calc_load(unsigned long ticks)
{
        unsigned long active_tasks; /* fixed-point */
        static int count = LOAD_FREQ;

        count -= ticks;
        if (count < 0) {
                count += LOAD_FREQ;
                active_tasks = count_active_tasks();
                CALC_LOAD(avenrun[0], EXP_1, active_tasks);
                CALC_LOAD(avenrun[1], EXP_5, active_tasks);
                CALC_LOAD(avenrun[2], EXP_15, active_tasks);
        }
}
```

The sampling interval of `LOAD_FREQ` is once every 5 HZ ≡ 5 sec

Don't confuse this with **reporting** periods: 1, 5, and 15 min

The Core Function

CALC_LOAD is a C **macro** defined in this code fragment:

```
extern unsigned long avenrun[ ];        /* Load averages */

#define FSHIFT          11              /* nr of bits of precision */
#define FIXED_1         (1<<FSHIFT)     /* 1.0 as fixed-point */
#define LOAD_FREQ       (5*HZ)          /* 5 sec intervals */
#define EXP_1           1884            /* 1/exp(5sec/1min) as fixed-pt */
#define EXP_5           2014            /* 1/exp(5sec/5min) */
#define EXP_15          2037            /* 1/exp(5sec/15min) */

#define CALC_LOAD(load,exp,n) \
          load *= exp; \
          load += n*(FIXED_1-exp); \
          load >>= FSHIFT;
```

Two questions:

1. What does the CALC_LOAD function actually do?
2. What are those *magic numbers* like EXP_5 $= 2014$?

What does CALC_LOAD do?

The CALC_LOAD function:

```
#define CALC_LOAD(load,exp,n) \
          load *= exp; \
          load += n*(FIXED_1-exp); \
```

is the fixed-point arithmetic version of this equation:

$$load(t) = load(t-1)\, e^{-\frac{5}{60m}} + n(t)\, (1 - e^{-\frac{5}{60m}}) \tag{1}$$

where

- $m = 1, 5, 15$ is the reporting period in minutes
- $load(t)$ means the load **now**
- $load(t-1)$ means the load **last time**
- $n(t)$ means the number of active processes **now**

To understand what eqn.(1) does, let's look at two extreme cases

Case 1: Empty Run Queue

Start at time $t = 0$ with no processes active $n(0) = 0$, so the 2^{nd} term vanishes.

Let the initial load be $load(0) = L > 0$.

Then eqn.(1) can be iterated up to time $t = T$ as:

$$\begin{aligned}
load(1) &= L\, e^{-\frac{5}{60m}} \\
load(2) &= load(1)\, e^{-\frac{5}{60m}} \\
&= \left(L\, e^{-\frac{5}{60m}}\right) e^{-\frac{5}{60m}} \\
&= L\, e^{-2\left(\frac{5}{60m}\right)} \\
load(3) &= L\, e^{-3\left(\frac{5}{60m}\right)} \\
&\ldots \\
load(T) &= L\, e^{-T\left(\frac{5}{60m}\right)}
\end{aligned}$$

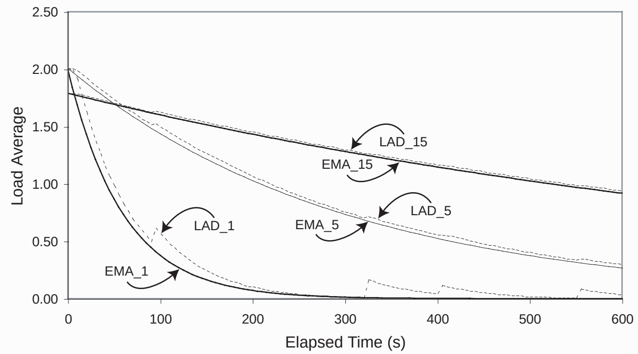
In general:

$$load(t) = L\, e^{-\left(\frac{5}{60m}\right)t} \tag{2}$$

which represents exponential **decay** (same as RC discharge)

The actual shape is determined by the value of $m = 1, 5, 15$

Exactly as seen in the experimental data after $t = 2100$ seconds



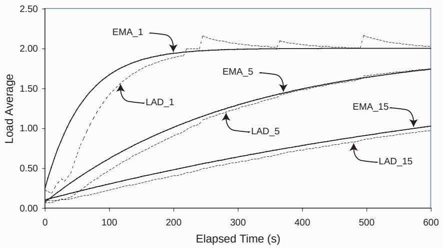
Case 2: Occupied Run-Queue

With $n(0) = 2$ processes the 2^{nd} term dominates eqn.(1)

Iteration produces:

$$load(t) = 2L(1 - e^{-\frac{5t}{60m}}) \quad (3)$$

Eqn.(3) is monotonically **rising** like our experiments. Applying a little Elec. Eng. theory ...



- Rise time is approx by $5\tau_{RC\_1}$
- Decay constant $\tau_{RC\_1} = 1$ min
- Rise time $\simeq$ 5 min or 300 sec

Exponential Smoothing

Exponential smoothing is a way to tame highly variable data. (Available in tools like EXCEL, R/S^+, Mathematica)

General form of smoothing equation is:

$$\underbrace{Y(t)}_{smoothed} = Y(t-1) + \underbrace{\alpha}_{damping} \left[\underbrace{X(t)}_{raw} - Y(t-1) \right] \quad (4)$$

By comparison the LA form is:

$$load(t) = load(t-1) + e^{-\frac{5}{60m}} [n(t) - load(t-1)] \quad (5)$$

Eqn.(5) is related to (4) via $\alpha = 1 - \exp(-\frac{5}{60m})$

Since they're used iteratively, it's called *exponential smoothing*

The Journey So Far

- Here, *load* refers to the length of the run **queue**
  - including processes in service
  - queue is sampled every 5 seconds
- `CALC_LOAD` is a type of **smoothing** function
- Magic numbers are the **weights** that control the amount of smoothing
  - Linux uses 10.11 fixed-point arithmetic
  - see my paper in these Proceedings for details
- Load **average** is an exponentially-damped moving average
  - moving average is the arithmetic sum of k samples
  - EMA is not your average average!
- `CALC_LOAD` puts more weight on **recent** load samples
- Same function is used for **financial** forecasting

About Those Magic Numbers

Remember this gobbledygook?

```
#define FSHIFT      11            /* nr of bits of precision */
#define FIXED_1     (1<<FSHIFT)   /* 1.0 as fixed-point */
#define LOAD_FREQ   (5*HZ)        /* 5 sec intervals */
#define EXP_1       1884          /* 1/exp(5sec/1min) as fixed-pt */
#define EXP_5       2014          /* 1/exp(5sec/5min) */
#define EXP_15      2037          /* 1/exp(5sec/15min) */
```

`1<<FSHIFT` is C code-speak for:

Move the binary-1 digit 11 places to the left

You know it as:

Multiply 2 by itself 11 times or 2^{11}

`CALC_LOAD` uses **fixed point** 10.11 format for "efficiency" in the kernel, which means 11 bits of precision are available for fractions [a]

[a] Because there are 10 + 11 = 21 bits altogether, that can become 21 + 21 = 42 bits when multiplied, and we only have 32-bits, ... blah, blah, blah

Dinking with Digits

Consider a decimal fraction like $n_{10} = 0.920044$

Suppose we want to express it correct to 3 decimal digits

The procedure is (See [8]):

1. $n_{10} \times 10^3 = 920.044$
2. Round down $\Rightarrow 920$
3. $n_{10} = 920 \times 10^{-3} = 0.920$

How many bits are needed to express 3 decimal places?

$10^3 \simeq 1024 = 2^{10}$ i.e., **10 bits** are needed

Magic Revealed

EMA damping factor is $e^{-\frac{5}{60m}}$

Consider $m = 1$ then $n_{10} = e^{-\frac{5}{60}} = 0.920044\ldots$

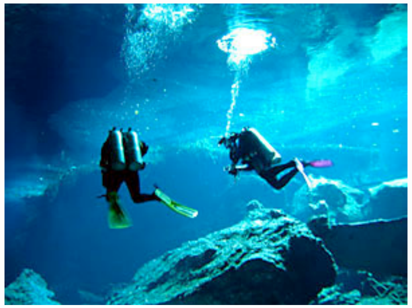
`CALC_LOAD` uses **11 bits** of precision: $2^{11} = 100000000000_2 = 2048_{10}$

That's equivalent to **4 decimal places** (actually 3.311)

Using the previous procedure:

1. $n_{10} \times 2048_{10} = 1884.25$
2. Round down $\Rightarrow 1884$
   - `#define EXP_1 1884` bingo!
3. $n_{10} = 1884 \times 2048_{10}^{-1} = 0.9200$

Coming Up for Air



Most sys admins tend to refer to and use the $m = 1$ minute load average

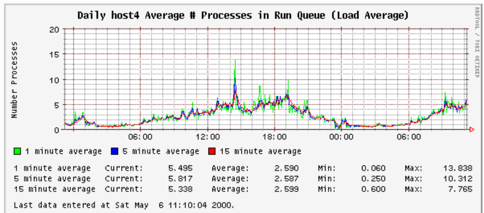
For queueing models we want the steady-state average so we can invoke useful relations like Little's law

But that suggests the $m = 15$ minute load average is more useful for capacity planning

- What is the relationship between the EMA and Little's law?
- How else can we use EMA data?

Time-Averaged Queue Length

Look at load over a long time (as $t \to \infty$) and break the time series



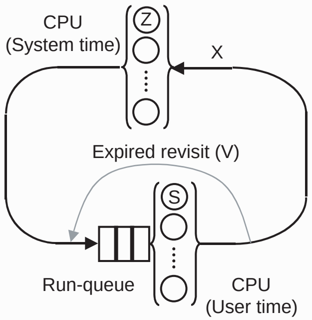
into set of columns:

- $\Delta t \ldots$ column width
- $Q(\Delta t) \times \Delta t \ldots$ sub-area
- $\sum Q(\Delta t) \times \Delta t \ldots$ total area

The time-averaged queue length: $\sum \frac{Q(\Delta t) \times \Delta t}{T} \to Q$

Steady-State Run Queue

- N: processes
- S: CPU service time (ticks)
- Z: suspended
- X: thoughput
- R: residence time
- $Q = XR$: Little's law

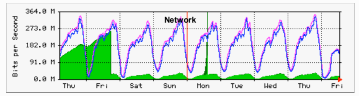


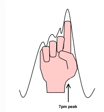
Extended this timeshare model to **fairshare** scheduler [2, 9]

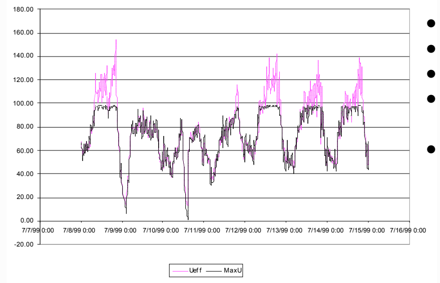
Load average is an **instantaneous** view or snapshot of the run-queue with the variance (spikes) damped down

Little's law is a time-averaged **steady-state** view of the run-queue

Forecasting Noisy Fingers

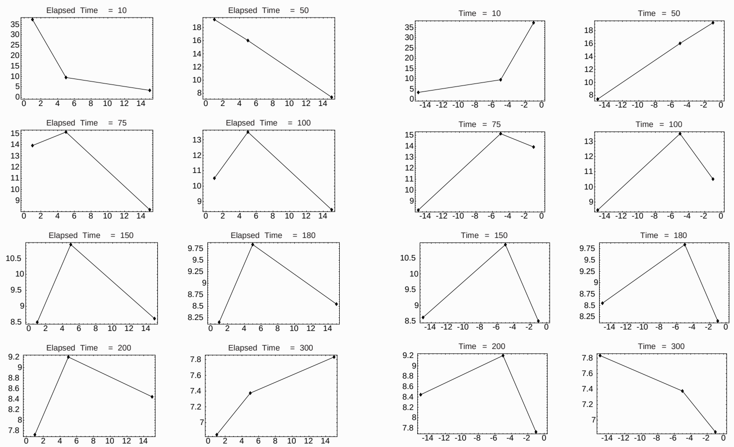




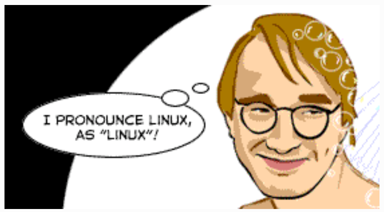


- *Numero Uno* signature in network data
- Truncated in CPU busy data
- Sample the 1-minute load average
- Treat as *instantaneous* queue length
- Apply *Multivariate Analysis* to estimate latent demand $\hat{\rho}$
- Metric variables: $X_1, X_2, \ldots, X_6$
- Coefficients: $\alpha_1, \ldots, \alpha_6, \beta$
- $\hat{\rho} = \alpha_1 X_1 + \ldots + \alpha_6 X_6 + \beta$
- *Noisy fingers* emerge as purple curve [10]
- Other applications of load average data to workload forecasting and GRIDs are discussed in my CMG paper [11]

Better Load Average Trending



Linux, Linus, Lex, and Load



Oh! This is the **other** Linus

The SCO Group law suit against infringement of its ownership of UNIX licensing rights is thought to include Linux

Linus Torvalds has publicly stated that he wrote all the Linux kernel code

Since it's a community effort, how can he be sure?

And who wrote the Linux load average code?

Performance history to the rescue!

- FreeBSD has no load average implementation, only hooks
- Solaris had the implementation wrong (in the past)
- The load average concept pre-dates UNIX
- CTSS and Multics [12] implemented it circa 1967
- Linus must have rolled his own *Q.E.D.*

References

[1] The Linux Cross-Reference project http://lxr.linux.no/source/kernel/

[2] N.J. Gunther, *Analyzing Computer System Performance Using Perl::PDQ*, Springer-Verlag, 2004

[3] Free Software Foundation http://www.gnu.org/

[4] http://cygwin.com/ Cygwin is a Linux-like environment (bash shell) for Microsoft Windows. N.B. It is *not* a way to run native Linux apps on the Windows operating system.

[5] J. Peek and T. O'Reilly and M. Loukides, *UNIX Power Tools*, O'Reilly & Assoc., 1997

[6] A. Cockcroft and R. Pettit, *Sun Performance and Tuning*, SunSoft Press, 1998

[7] ORCA tool www.orcaware.com/orca/docs/orcallator.html#processes_in_run_queue_system_load

[8] N.J. Gunther, "Sins of Precision—Damaging Digits in Capacity Calculations," Proc. CMG 2002, Reno, NV

[9] N.J. Gunther, "Capacity Planning for Solaris SRM. 'All I Ever Want is My Unfair Advantage!' (And Why You Can't Have It)," Proc. CMG 1999, Reno, NV

[10] *Guerrilla Capacity Planning* See the class schedule at www.perfdynamics.com/Classes/schedule.html

[11] N.J. Gunther, "Linux Load Average Revealed," Proc. CMG 2004, Las Vegas, NV

[12] J.H. Saltzer and J.Q. Gintell, "The Instrumentation of Multics," 1970 www.multicians.org/InstrumentationPaper.html,