Load average é bom? Conheça o stretch factor

Como medir desempenho?

O que significam aqueles valores de "load average" na saída de comandos como procinfo e uptime, e o que devemos fazer com eles?

por Neil Gunther

A maioria dos administradores de sistemas Linux estão familiarizados com aqueles três pequenos números que aparecem em comandos de *shell* como *procinfo*, *uptime*, *top* e *ruptime*. O uptime, por exemplo, informa:

```
9:40am up 9 days, load average:
➥ 0.02, 0.01, 0.00
```

A métrica da carga média (*load average*) está sempre incluída na saída de comandos como *uptime*. Enquanto os valores de load average são bem conhecidos pelos administradores, seu significado é pouco compreendido. A página de manual do uptime informa apenas que esses valores representam as médias de carga nos últimos 1, 5 e 15 minutos. Contudo, isso explica somente o motivo de haver três valores, mas não o significado de *carga*, o que nos impede de tentar imaginar o que há de errado com o servidor, ou o que fazer para corrigir seu desempenho. Este artigo analisa de perto a métrica usada pelo load average e como usá-la.

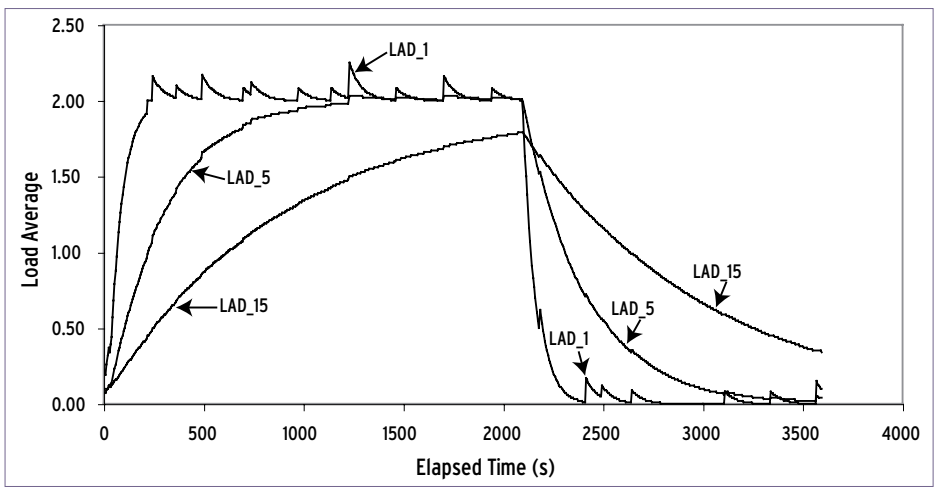

Figura 1 Dados de *load average (LAD)* coletados numa plataforma Linux controlada por um período de uma hora. *LAD 1*, *LAD 5* e *LAD 15* são as métricas de um, cinco e quinze minutos, respectivamente.

Experimentos controlados

Vamos começar com um pequeno experimento para demonstrar como os valores de load average respondem às mudanças na carga do sistema. Load averages experimentais foram amostradas ao longo de um período de uma hora (3600 segundos) numa máquina Linux monoprocessada ociosa. Esses testes consistiram em duas fases. Na primeira delas, dois *jobs* exigentes em CPU foram iniciados como processo em segundo plano, e rodaram durante

2100 segundos. Nesse momento, os dois processos foram parados simultaneamente, mas as medições de load average continuaram por mais 1500 segundos. O **exemplo 1** contém o script *Perl* que foi usado para amostrar os valores a cada cinco segundos usando o comando *uptime*.

Um programa em C chamado `burncpu.c` foi criado para desperdiçar ciclos de CPU. A saída do *top* mostra que as duas instâncias do *burncpu* aparecem nas duas primeiras colocações em relação ao consumo de CPU durante o período de execução do *getload* (**tabela 1**).

A **figura 1** mostra que o load average de um minuto atinge o valor de 2.0 após 300 segundos de teste; o load average de cinco minutos atinge 2.0 por volta dos 1200 segundos; o load average de 15 minutos chegaria a 2.0 em aproximadamente 4500 segundos, mas os processos foram mortos na marca de 2100 segundos.

Os leitores com conhecimentos em engenharia elétrica notarão imediatamente a semelhança da curva da **figura 1** e as de voltagem produzidas pelo carregamento e descarregamento de um circuito RC (resistor-capacitor). Note que a carga máxima durante o teste é equivalente ao número de processos exigentes de CPU rodando no momento das medições.

Os picos na curva do alto são resultado dos vários *daemons* "acordando" temporariamente e em pouco tempo voltando a "dormir".

O próximo objetivo é explicar por que os dados de load average desses experimentos exibem as características vistas na **figura 1**. Para isso, foi necessário explorar o código do kernel Linux 2.6.20.1[1] que calcula esses valores.

Código do kernel

Examinando o código-fonte do escalonador do kernel[2], encontramos a função `calc_load`. Essa é a rotina primária que calcula a métrica do load average. Essencialmente, a rotina verifica se o período de amostragem já expirou, reinicia o contador de amostragem e chama a subrotina `CALC_LOAD` para calcular cada uma das métricas de um, cinco e quinze minutos. O intervalo de amostragem usado para `LOAD_FREQ` é 5*HZ. Qual o comprimento desse intervalo?

Toda plataforma Linux possui um relógio implementado em hardware. Esse relógio em hardware tem taxa constante de batidas, à qual o sistema é sincronizado. Para tornar pública essa taxa de batidas, o relógio envia uma interrupção para o kernel a cada batida. O intervalo real entre as batidas é diferente. A maioria dos sistemas Linux possui um intervalo de batida de CPU configurado para 10 milissegundos.

A definição específica da taxa de batida está contida numa constante chamada de `HZ`, mantida por um arquivo de cabeçalho específico para cada sistema, chamado `param.h`. No código-fonte do Linux usado neste artigo, há o valor 100 para plataformas Intel em `lxr.linux.no/source/include/asm-i386/param.h`, e em plataformas *SPARC* o arquivo é `lxr.linux.no/source/include/asm-sparc/param.h`.

A função `calc_load` é chamada numa freqüência definida pela taxa de batidas – uma vez a cada cinco segundos (não cinco vezes por segundo, como muitos pensam). Esse período de amostragem de cinco segundos é independente dos intervalos de um, cinco e quinze minutos.

Revelação

A função `calc_load` do kernel Linux refere-se à macro C chamada `CALC_LOAD`, que faz o verdadeiro trabalho de calcular o load average. `CALC_LOAD` é definida em `http://lxr.linux.no/source/include/linux/sched.h`.

Nesse arquivo, pode-se ver que, matematicamente, `CALC_LOAD` depende do número de processos ativos e, obviamente, da carga, além da medição imediatamente anterior da carga.

No Linux, um processo pode estar em um de aproximadamente meia dúzia de processos, dos quais *em execução* (*running*), *executável* (*runnable*, *R* no comando *ps*) e *dormente* (*sleeping*, *S* no comando *ps*) são os três estados primários. Cada métrica de load average é baseada no número total de processos que estão:

- executáveis e aguardando na fila de execução do escalonador;
- atualmente em execução num processador.

Na terminologia usada no kernel, o total dos processos ativos é chamado de fila (*queue*). Ela literalmente significa não apenas os processos na fila de espera (a chamada fila de execução, ou *run queue*), como também aqueles atualmente sendo servidos (ou seja, em execução).

Stretch factor

Inevitavelmente, surge a questão: o que é um bom valor de load average? Se analisarmos atentamente o código-fonte do kernel, veremos que esse valor tem relação com a movimentação de processos na fila de execução, e então podemos converter essa pergunta para "qual deve ser o comprimento da minha fila?"

Filas longas correspondem a tempos de resposta dilatados, então é a métrica de tempo de resposta que realmente deve receber atenção. Uma conseqüência é que uma fila longa pode causar "tempos de resposta ruins", mas isso depende do que significa "ruim". Na maioria das ferramentas de gerenciamento de desempenho há uma desconexão entre o comprimento medido da fila de execução e os tempos de resposta percebidos pelo usuário. Outro problema é que o comprimento da fila

Tabela 1: Saída do comando top

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	%CPU	%MEM	TIME	CPU	COMMAND
20048	neil	25	0	256	256	212	R	30.6	0.0	0:32	0	burncpu
20046	neil	25	0	256	256	212	R	29.3	0.0	0:32	0	burncpu
15709	mir	24	0	9656	9656	4168	R	25.6	1.8	45:32	0	kscience.kss
1248	root	15	0	10M	10M	1024	S	9.5	2.1	368:25	0	X
20057	neil	16	0	1068	1068	808	R	2.3	0.2	0:01	0	top
1567	mir	15	0	38 M	38M	14260	S	1.3	7.6	40:10	0	mozilla-bin

Exemplo 1: Amostras de load average

```
01 #! /usr/bin/perl -w
02 $intervalo_amostragem = 5; #segundos
03
04 # Usar CPU ao maximo ...
05 system("./burncpu amp;");
06 system("./burncpu &");
07
08 # Monitorar para sempre o load average via
09 # uptime e separar os campos por tabs
10 # para uso em programas de planilhas.
11 while (1) {
12   @uptime = split (/ /, 'uptime');
13   foreach $up (@uptime) {
14     # coletar a hora
15     if ($up =~ m/(\d\d:\d\d:\d\ d)/) {
16       print "$1\t";
17     }
18     # coletar as tres metricas de carga
19     if ($up =~ m/(\d{1,}\.\d\d)/){
20       print "$1\t";
21     }
22   }
23   print "\n";
24   sleep ($intervalo_amostragem);
25 }
```

é uma medida absoluta, enquanto o que realmente precisamos é uma medida relativa de desempenho. Até as palavras "bom" e "ruim" são relativas. Essa medida relativa é chamada de *stretch factor* (fator de alongamento), e mede o comprimento médio da fila relativo ao número médio de requisições já em execução. Ele é expresso em múltiplos de unidades de serviço. Um stretch factor de 1, então, significa que nenhum processo precisa esperar.

O que torna o stretch factor realmente útil é que ele pode ser facilmente comparado com objetivos de nível de serviço. Esses objetivos geralmente são expressos em certos tipos de unidades de negócios (por exemplo, cotações por hora é uma unidade de trabalho comum na área de seguros). O nível de serviço esperado é chamado de *service level objective*, ou *SLO*, e é expressado como múltiplos da unidade de serviço relevante. Um SLO pode ser documentado como "O tempo médio de resposta para o usuário não deve exceder 15 unidades de serviço entre as horas de pico de 10:00 e 14:00".

Isso equivale a estabelecer que o SLO não deve exceder um stretch factor de 15.

Usando os símbolos definidos na **tabela 2**, o stretch factor f pode ser calculado como a razão:

$$f = \frac{Q}{mp}$$

Sabemos que, numa máquina monoprocessada, $m = 1$, e podemos supor $Q = 2$ para o load average de um minuto, e $p = 1$ porque a tarefa é totalmente dependente do processador. Substituindo esses valores na equação, obtem-se um stretch factor de $f = 2$. Esse resultado informa que o tempo esperado para qualquer processo ser finalizado é de dois períodos de serviço.

Note que não precisamos saber quanto vale o período de serviço. Em nosso caso, o stretch factor e o valor de load average são iguais, pois os processos estão sendo executados num único processador, e dependem primordialmente apenas da CPU. A seguir, é descrito um cenário de uso do stretch factor em situações reais.

Tabela 2: Definições do stretch factor

m	Número de processadores ou núcleos
Q	*Load average* medido
p	Utilização do processador medida

Anti-spam

Todos os principais serviços de email utilizam analisadores de spam. Uma configuração típica pode consistir em um conjunto de servidores especializados, cada um analisando diversas mensagens com uma ferramenta de filtragem como o *SpamAssassin*[3]. Um portal web bastante conhecido e com alto tráfego tem algo como 100 servidores, cada um contendo dois processado-

Tabela 3: Estatísticas diárias do servidor anti-spam

Número de CPUs	4
Spam detectado	33901
Não-spam aceito	23123
Emails processados	57024
Emails por hora	2376
Por CPU/hora	594
CPU usada(%)	99
Segundos por email	6
Load average	97.36

res de núcleo duplo, todos varrendo emails ininterruptamente. As estatísticas da filtragem diária de spam são mostradas na **tabela 3**.

Um balanceador de carga foi usado para distribuir o trabalho pelos servidores. A eficácia do balanceador foi monitorada usando load averages de um minuto. A amostra desses valores de metade dos servidores revela um desequilíbrio de carga no datacenter.

Um administrador de sistemas poderia perguntar:

- Por que há um desequilíbrio de carga?
- A maioria dos servidores está sobrecarregada devido ao desequilíbrio?
- O load average de Q = 97,36 emails é desejável?
- Qual deveria ser o desempenho real do servidor?
- Quantos servidores a mais precisam ser colocados no datacenter no próximo ano para manter o desempenho atual de varredura de emails sob uma carga maior?

Substituindo os valores na equação, chegamos ao stretch factor:

$$f = \frac{97,36}{4 \bullet 0,99} = 24.59$$

Exemplo 2: PDQ para o datacenter anti-spam

```
01 #!/usr/bin/env python import pdq
02 # Parametros de desempenho medidos
03 cpusPorServidor = 4
04 emailsProcessados = 2376 # emails por hora
05 tempoVarrendo = 6.0 # segundos por email
06 pdq.Init("Spam Farm Model")
07 # A unidade de tempo eh o SEGUNDO ...
08 nstreams = pdq.CreateOpen("Email", float(emailsProcessados)/3600)
09 nnodes = pdq.CreateNode("lataSpam", int(cpusPorServidor), pdq.MSQ)
10 pdq.SetDemand("lataSpam", "Email", tempoVarrendo)
11 pdq.Solve(pdq.CANON)
12 pdq.Report()
```

A **tabela 3** mostra que o tempo médio para varrer um email (*S*) é de seis segundos.

Então, um stretch factor de 25 períodos de serviço implica a duração de 25 x 6 = 150 segundos ou 2,5 minutos a partir da chegada do email no portal até sua entrega ao devido destinatário.

Um valor absoluto de Q = 97,36 para o load average nos diz muito pouco. O stretch factor relativo, no entanto, informa quantos períodos de serviço a tarefa de filtragem de spam está custando.

Quanto à questão sobre um load average de 97,36 ser desejável, isso depende dos objetivos de negócio acordados. Pelo menos, agora essas questões podem ser resolvidas quantitativamente, em vez de especulativamente.

Também é possível usar os dados da **tabela 3** para modelar o desempenho usando uma ferramenta de predição de desempenho como o *PDQ* (veja o **quadro 1**).

O modelo de servidor anti-spam no *PyDQ* (PDQ em *Python*) é mostrado no **exemplo 2**. Executá-lo produz um relatório que contém a saída mostrada no **exemplo 3**.

O stretch factor predito pelo PDQ é um pouco maior do que aquele calculado pela equação que mostramos acima. Para entender o motivo disso, é necessário examinar a seção do relatório do PDQ que apresenta as informações de desempenho do servidor (**exemplo 4**).

Dada a taxa a que a carga chega (2376 emails por hora), cada CPU deve estar 99% ocupada. Essa utilização é maior do que o verificado nos servidores reais, devido ao desequilíbrio da carga. O PDQ espera um balanceamento de carga ideal por todos os servidores, então há mais trabalho sendo exercido. O load average predito (*Queue length*, no **exemplo 4**), está mais próximo de 100 emails; portanto, o stretch factor de 25,45 predito é um pouco maior que o valor calculado de 24,59.

Ambos os valores de stretch factor foram considerados aceitáveis sob condições de pico de carga.

Exemplo 3: Saída do PDQ para desempenho do sistema

```
01 ****** SYSTEM Performance *******
02 Metric Value Unit
03 -- --- --
04 Workload:           "Email"
05 Number in system    100.7726 Trans
06 Mean throughput     0.6600 Trans/Sec
07 Response time       152.6858 Sec
08 Stretch factor      25.4476
```

Como todos os servidores estão quase saturados, um recurso é atualizá-los com CPUs mais rápidas ou, mais provavelmente, adquirir novos servidores quadriprocessados para lidar com a carga adicional esperada. O PDQ ajuda a dimensionar o número de novos servidores, com base nos stretch factors atual e esperado. Claramente, o stretch factor oferece um indicador mais apurado de gerenciamento de desempenho do que os valores absolutos de load average.

Triturando

Pode-se usar um modelo semelhante ao PyDQ para ver o que significa não ter uma fila de espera com todas as CPUs ocupadas. Nesse caso, cada processo do Linux leva dez horas para terminar, pois está transformando dados de exploração de petróleo para posterior análise por geofísicos. O modelo correspondente do PyDQ é mostrado no **exemplo 5**.

Exemplo 4: Saída do PDQ para desempenho dos recursos

```
01 ****** RESOURCE Performance *******
02 Metric Resource Work Value Unit
03 -- ---- -- --- --
04 Throughput spamCan Email        0.0660 Trans/Sec
05 Utilization spamCan Email       99.0000 Percent
06 Queue length spamCan Email      100.7726 Trans
07 Waiting line spamCan Email      96.8126 Trans
08 Waiting time spamCan Email      146.6858 Sec
09 Residence time spamCan Email    152.6858 Sec
```

Ao rodar o **exemplo 5**, a fila de espera tem comprimento essencialmente zero, e todas as quatro CPUs estão ocupadas, embora apenas 25% utilizadas. Se verificássemos as estatísticas de CPU enquanto o sistema rodava, veríamos que, na realidade, cada CPU estava 100% ocupada. Para entender o que o PDQ nos diz, é necessário examinar a seção *System Performance* do relatório do PDQ (**exemplo 9**).

O stretch factor é 1 (período de serviço) porque não há fila de espera. Cada trabalho leva dez horas para terminar, então a resposta é de aproximadamente dez horas.

O motivo disso parecer um pouco estranho é que o PDQ realiza suas estimativas com base num comportamento estável (ou seja, como o sistema se comporta a longo prazo). Com um tempo de serviço de dez horas, realmente é preciso observar o sistema por muito mais tempo que isso para verificar seu comportamento a longo prazo. Muito mais tempo, nesse caso, significa uma ordem de 100 horas ou mais. Não é realmente necessário fazer isso, mas o PDQ nos informa como seria o panorama se o fizéssemos.

Como o período médio de serviço é relativamente grande, a taxa de requisições é correspondentemente pequena, para que nenhuma fila de espera se forme. Isso significa que a utilização do processador em 25% também é baixa – a longo prazo.

Observar o sistema apenas alguns minutos enquanto ele tritura dez horas de trabalho com dados de exploração de petróleo corresponde a uma fotografia instantânea do sistema, não a uma visão de longo prazo.

Esses dois exemplos de stretch factor envolvem cargas de trabalho limitadas pela CPU. Cargas limitadas por I/O (seja de disco ou de rede) tenderão a exibir load averages menores que tarefas limitadas pela CPU, caso esses processos sejam suspensos ou adormeçam à espera de dados. Nesse estado, eles não estão nem executáveis nem em execução, e portanto não contribuem para os cálculos de load average. Da mesma forma, quando o driver de I/O do Linux está trabalhando, ele roda em modo de kernel numa CPU, e também não contribui para o cálculo do load average.

O valor de Q mede o número total de requisições; tanto em espera quanto em execução. Ele não tem muito significado porque se trata de um valor absoluto. Todavia, combinando-o ao número de processadores (m) e sua carga média aferida (p), o stretch factor f consegue oferecer uma métrica melhor para o gerenciamento de desempenho de servidores *SMP* (multiprocessados ou com múltiplos núcleos), pois é um indicador relativode desempenho

Quadro 1: PDQ em Python

PDQ (do inglês *Pretty Damn Quick*, "rápido pra caramba") é uma ferramenta de modelagem para analisar as características do desempenho de recursos computacionais, como processadores, discos e um conjunto de processos que fazem requisições desses recursos, por exemplo. Um modelo PDQ é analisado com algoritmos baseados na teoria de enfileiramento (*queueing*). A versão atual facilita a criação e análise de modelos de desempenho em *C, Perl, Python, Java* e *PHP*.

As funções de PDQ do Python usadas nessa seção são:

- `pdq.Init()` inicializa as variáveis PDQ internas;
- `pdq.CreateOpen()` cria uma carga de rede;
- `pdq.CreateNode()` cria um servidor;
- `pdq.SetDemand()` estabelece o tempo de serviço de carga de rede no recurso do servidor;
- `pdq.Solve()` calcula a métrica de desempenho;
- `pdq.Report()` gera um relatório genérico de desempenho.

Mais informações sobre o PDQ estão disponíveis em sua biblioteca online[4].

Exemplo 5: Modelo do PyDQ

```
01 #!/usr/bin/env python import pdq
02 processadores = 4 # Igual ao exemplo do anti-spam
03 taxaChegada = 0.099 # Tarefas por hora (poucas mensagens)
04 tempoTrituracao = 10.0 # Horas (tempo de servico bem longo)
05
06 pdq.Init("ORCA LA Model")
07 s = pdq.CreateOpen("Crunch", taxaChegada)
08 n = pdq.CreateNode("HPCnode", int(processadores), pdq.MSQ)
09 pdq.SetDemand("HPCnode", "Crunch", tempoTrituracao)
10 pdq.SetWUnit("Jobs")
11 pdq.SetTUnit("Hour")
12 pdq.Solve(pdq.CANON)
13 pdq.Report()
```

que pode ser diretamente comparado com SLOs estabelecidos.

Conclusão

A carga média fornece informações sobre a tendência de crescimento da fila de execuçaõ, que é o motivo de haver três métricas. Cada métrica captura informações de tendência da fila de execução conforme ela se mostrava há um, cinco e 15 minutos. Comparada às capacidades de visualização gráfica dos dados, essa técnica de representação de dados parece antiquada. Na verdade, o load average é uma das formas mais antigas de instrumentação de sistemas operacionais, datando de 1965.

Este artigo apresentou o stretch factor como uma forma mais adequada para usar os dados de load average para gerenciar o objetivo de nível de serviço dos aplicativos em servidores com múltiplos núcleos. ■

Mais informações

[1] Código-fonte do Linux 2.6.20.1: http://lxr.linux.no/source/

[2] Código do escalonador do Linux: http://lxr.linux.no/source/kernel/timer.c

[3] SpamAssassin: http://spanassassin.apache.org/

[4] Biblioteca PDQ: http://www.perfdynamics.com/Tools

Sobre o autor

Neil Gunther é consultor com renome internacional, e fundou a empresa Performance Dynamics após trabalhar na NASA, no PARC da Xerox e na Pyramid/Siemens Technology. Neil também é membro das instituições AMS, APS, ACM, CMG, IEEE e INFORMS.