6
Expectativas

Prever o comportamento de um sistema em função dos fatores de carga é uma tarefa bastante complicada. A maioria das previsões só são razoáveis para determinados intervalos e taxa de amostragem. Um bom exemplo é o envio de um pacote em uma rede ethernet. Enviar um pacote contendo apenas um byte ou 50 bytes provavelmente tem o mesmo peso para o sistema já que o tamanho mínimo do pacote para permitir a detecção de colisão é 64 bytes, logo o sistema irá completar o pacote até chegar ao tamanho mínimo. A sobrecarga das outras camadas para trafegar alguns bytes a mais provavelmente é imperceptível. Porém, se começarmos a comparar a transmissão de 10Kbytes que precisa de uma série de pacotes, com seus múltiplos, podemos começar a esperar uma relação aproximadamente linear.

Se considerarmos que esses pacotes estão encapsulados em uma conexão TCP, aí entra a influência de um protocolo com janelas de transmissão e confirmação de recebimento. Se a janela for muito grande e existirem muitos dados para serem transmitidos, uma única conexão consegue saturar a rede, introduzindo outra não linearidade.

Assim como o exemplo de rede, temos outras não linearidades, por exemplo o comportamento do escalonamento preemptivo, que só ocorre a partir de um certo volume de processamento, se o processamento for curto, o processo nem chega a ser interrompido. Outro exemplo é a taxa de acerto dos *caches* de memória. Porém, esperamos poder descrever matematicamente uma tendência de comportamento em uma faixa de operação normal.

Começamos identificando por variáveis os fatores de carga:

C número de requisições concorrentes,

X volume de dados transferido,

P volume de processamento.

Esses fatores influenciam as variáveis de resposta do sistema:

C_p número de atendimentos concorrentes,

t_{chave} tempo para realizar uma tarefa identificada pela chave,

T_{chave} tempo total que o atendimento de uma requisição usou ou esperou por tarefas do tipo t com a mesma chave.

O número de requisições concorrentes (C) implica na maioria dos sistemas analisados em atendimentos concorrentes (C_p), mas mesmo nos sistemas que não implementam atendimento concorrente, esperamos que C influencie o desempenho de tarefas desempenhadas pelo SO, que por sua vez implementa concorrência.

Analisando os parâmetros do modelo, esperamos que os resultados em máquinas mono-processador sigam aproximadamente o comportamento descrito a seguir.

6.1 Formulação das expectativas

Esperamos que exista um tempo de latência no *handshake* de uma conexão TCP (t_{hshake}) aproximadamente constante para todos os modelos de concorrência. Esse tempo corresponderia ao tempo que o pedido de conexão e sua resposta demorariam para serem trocados e a conexão começasse a transmitir dados. No caso do atendimento de múltiplos pedidos de conexão simultaneamente, o tempo total do *handshake* da conexão (T_{hshake}) seria amortizado pelo número de requisições concorrentes (C), pois um pedido pode estar trafegando na rede enquanto outro é processado no servidor.

$$T_{hshake} \approx \frac{t_{hshake}}{C}$$

Esperamos que exista um tempo de processamento inicial de uma tarefa (t_{inic}) que seja aproximadamente constante para todos os modelos. Esse tempo corresponderia ao processamento necessário para aceitar uma conexão, interpretar o cabeçalho do protocolo HTTP e chavear o controle para o manipulador de requisições que combina com a requisição.

Esperamos que o tempo total de processamento inicial (T_{inic}) aumente linearmente com o aumento dos atendimentos concorrentes (C_p).

$$T_{inic} \approx t_{inic} \times C_p$$

Esperamos que cada modelo de concorrência tenha um tempo diferente, mas aproximadamente constante por modelo, de criação de uma tarefa ($t_{criatarefa}$) responsável pela requisição. Esperamos que o tempo total de criação de tarefas ($T_{criatarefa}$) aumente linearmente com o aumento dos atendimentos concorrentes (C_p).

$$T_{criatarefa} \approx t_{criatarefa} \times C_p$$

Em $t_{criatarefa}$, a tarefa pode ser uma co-rotina, clone, *thread* ou processo. No caso de não ser criada uma nova tarefa, $t_{criatarefa} = 0$.

Esperamos que cada modelo de *sandbox* tenha um tempo diferente, mas aproximadamente constante por modelo, de criação da *sandbox* ($t_{criasandbox}$) responsável pela requisição. Esperamos que o tempo total de criação de *sandboxes* ($T_{criasandbox}$) aumente linearmente com o aumento dos atendimentos concorrentes (C_p).

$$T_{criasandbox} \approx t_{criasandbox} \times C_p$$

Em $t_{criasandbox}$, a *sandbox* pode ser Venv, Rings ou processo separado. No caso de não ser criada uma nova *sandbox*, $t_{criasandbox} = 0$.

Consideramos que o tempo de processamento de uma chamada de transmissão de um bloco de dados (t_{transm}) seja aproximadamente constante para todos os modelos. Então consideramos que o tempo total das chamadas de transmissão de dados (T_{transm}) aumenta linearmente com o aumento do volume de dados (X) e dos atendimentos concorrentes(C_p).

$$T_{transm} \approx t_{transm} \times C_p \times X$$

Esperamos que nos diversos modelos os sistemas regularmente fiquem bloqueados em E/S. Esperamos que o tempo de espera em uma série de E/S de uma tarefa ($t_{E/S}$) seja aproximadamente constante. Esperamos então que o tempo total de espera em E/S ($T_{E/S}$) sofra influência crescente com o volume de dados transferidos (X). Esperamos que o aumento dos atendimentos concorrentes (C_p) diminua o tempo de processamento bloqueado em espera pois teoricamente, ao invés de esperar em E/S, outra tarefa ganha o controle da CPU. Então:

$$T_{E/S} \approx \frac{t_{E/S}}{C_p} \times X$$

No modelo de co-rotinas temos escalonamento colaborativo apenas nos casos da E/S ficar bloqueada. Esse escalonamento gera um processamento adicional para a troca de contexto. Considerando que o tempo de troca de contexto colaborativo de uma tarefa (t_{colab}) seja aproximadamente constante, esperamos que o tempo total utilizado em escalonamento colaborativo (T_{colab}) aumente com o aumento do volume de dados transferidos (X) e com o aumento dos atendimentos concorrentes (C_p).

$$T_{colab} \approx t_{colab} \times X \times C_p$$

Esperamos que, nos modelos preemptivos, o sistema também escalone em operações de E/S que bloqueiem, então também podemos utilizar T_{colab} como um dos fatores do escalonamento preemptivo em E/S.

Para o caso de escalonamento preemptivo por limite de tempo de processamento, que chamaremos aqui simplesmente de preemptivo (não inclui o escalonamento em E/S bloqueante). Consideramos que o tempo de troca de contexto do escalonamento preemptivo (t_{preemp}) seja aproximadamente constante. Esperamos que o tempo total utilizado em escalonamento preemptivo (T_{preemp}) aumente com o aumento do volume de processamento (P) que é gerado pelo *loop* vazio do nosso teste e com o aumento dos atendimentos concorrentes (C_p).

$$T_{preemp} \approx t_{preemp} \times P \times C_p$$

Também esperemos que o volume de processamento (P) do *loop* vazio tenha forte influência no tempo de resposta. Consideramos o tempo de execução de um ciclo de *loop* (t_{loop}) constante. Então o tempo total utilizado em processamento cresce com o aumento do volume de processamento (P) e com o aumento dos atendimentos concorrentes (C_p).

$$T_{loop} \approx t_{loop} \times P \times C_p$$

Consideramos que cada modelo gasta um tempo aproximadamente constante para transferir uma quantidade fixa de dados entre a tarefa do *sandbox* e a tarefa do servidor web($t_{transfsandbox}$). O tempo total utilizado transferindo dados entre *sandbox* e a tarefa ($T_{transfsandbox}$) aumenta com o aumento do volume de dados transferidos (X) e com o aumento dos atendimentos concorrentes (C_p).

$$T_{transfsandbox} \approx t_{transfsandbox} \times X \times C_p$$

Esperamos então que o tempo total de resposta de uma requisição (T_{resp}) em função do número de requisições concorrentes (C), com X dados transmitidos e P processamento, varie seguindo as tendências da seguinte fórmula:

$$T_{resp} \approx T_{hshake} + T_{inic} + T_{criatarefa} + T_{criasandbox} + T_{transm} + T_{E/S} + T_{colab} + T_{preemp} + T_{loop} + T_{transfsandbox}$$

onde:

T_{hshake} diminui com o número de requisições concorrentes

$T_{E/S}$ diminui com os atendimentos concorrentes

$T_{inic}, T_{criatarefa}, T_{criasandbox}, T_{transm}, T_{colab}, T_{preemp}, T_{transf}, T_{loop}$ aumentam com os atendimentos concorrentes

$T_{E/S}, T_{colab}, T_{transfsandbox}, T_{transm}$ aumentam com o volume de transferência dados.

T_{preemp}, T_{loop} aumentam com o volume de processamento

Como o que estamos interessados é o *thoughput* em requisições por segundo (res).

$$res \approx \frac{C_p}{T_{resp}}$$

$$res \approx \frac{C_p}{\frac{t_{hshake}}{C} + \left(\frac{t_{E/S}}{C_p} \times X\right) + (t_{inic} + t_{criatarefa} + t_{criasandbox} + (t_{transm} + t_{colab} + t_{transfsandbox}) \times X + (t_{preemp} + t_{loop}) \times P) \times C_p}$$

Essa fórmula serve de base para prever tendências de mudança quando variamos os fatores de carga.

6.2 Expectativas no sistema base

Quanto ao modelo de concorrência, como numa rede ethernet local a latência é baixa, esperamos que o tempo para realizar o *handshake* de uma conexão seja muito pequeno comparado com o tempo total de resposta da aplicação. Esperamos então poder desprezá-lo ($\frac{t_{hshake}}{C} \approx 0$).

6.2.1 Expectativas fixando o modelo de sandbox e variando o modelo de concorrência

Gostariamos de saber o modelo de concorrência mais adequado para cada modelo de *sandbox*.

Seqüencial:

Espera-se que a resposta do modelo seqüencial seja praticamente constante em relação ao número de requisições concorrentes, ou seja, independente do número de requisições concorrentes, a mesma quantidade de requisições serão respondidas por intervalo de tempo ($C_p = 1 \forall C > 0$).

Espera-se que este seja o modelo que apresentará a resposta mais rápida sempre que as requisições sejam feitas de forma seqüencial ($C = 1$). Espera-se isso porque esse modelo não demanda processamento para criação de tarefas nem para a realização do escalonamento ($t_{criatarefa}, t_{colab}, t_{preemp} = 0$). Porém, espera-se que o aumento do número de requisições concorrentes ($C \to \infty$) não tenha maior influência no número de requisições atendidas por segundo,

já que mesmo que 100 conexões sejam solicitadas simultaneamente, não há atendimentos concorrentes, pois apenas uma será respondida por vez, seqüencialmente, até que a última seja atendida ($C_p = 1$), resultando numa taxa de respostas constante.

Espera-se que a medida que o volume de processamento e de transferência aumente ($X, P \rightarrow \infty$), o número de respostas por segundo se degrade linearmente. Espera-se que, quanto maior o processamento ($P \rightarrow \infty$), melhor o modelo seqüencial funcione em relação aos modelos com escalonamento. Espera-se que, quanto maior a E/S ($X \rightarrow \infty$), pior seja o desempenho do modelo seqüencial em comparação com os escalonados. Imagina-se que, a partir do momento que comecem a chegar mais requisições concorrentes ($C \rightarrow \infty$), os outros modelos que não bloqueiam todo o processamento na E/S de uma única tarefa, comecem a melhorar em relação ao seqüencial, eventualmente passando o modelo seqüencial.

Co-rotinas:

Espera-se um processamento adicional constante em relação ao modelo seqüencial para a criação de tarefas ($t_{criacorotina} > 0$). Além disso, espera-se que apesar de não haver escalonamento preemptivo ($t_{preemp} = 0$), quanto maior a transferência de dados maior o número de chamadas de E/S sem resposta imediata e conseqüentemente de escalonamento. O escalonamento também cria processamento adicional. Então, comparando o caso de requisições seqüenciais ($C = 1$) nos modelos de co-rotinas e sequencial, poderá ser levantado o peso da sobrecarga do escalonamento na chamada assíncrona em relação a chamada síncrona ($\frac{T_{rescorotina}}{T_{ressequencial}}$). A medida que o número de requisições e o volume de transmissão de dados aumentarem espera-se que o modelo de co-rotinas desempenhe melhor comparando com o modelo seqüencial. Assim será possível levantar o quanto se ganha aproveitando o tempo que a chamada de E/S estaria bloqueada.

Clones:

Espera-se um processamento adicional constante para a criação de tarefas em relação ao modelo seqüencial ($t_{criaclone} > 0$). Assim como no caso das co-rotinas, também se espera escalonamento quando requisições de E/S bloqueiem ($t_{colab} > 0$), além disso espera-se escalonamento preemptivo que também demanda processamento ($t_{preemp} > 0$). No modelo de clones ocorre escalonamento tanto quando um processo fica bloqueado em E/S quanto por preempção. Espera-se que o escalonamento preemptivo acrescente um peso a mais que o peso do modelo de co-rotinas que só tem escalonamento quando

fica bloqueado em E/S. Espera-se também que o tempo de criação de clones seja maior que o de criação de co-rotinas ($t_{criaclone} > t_{criacorotina}$).

Threads:

Espera-se um processamento adicional constante para a criação de tarefas ($t_{criathread} > t_{criacorotina}, t_{criaclone}$), ainda maior que para o modelo de co-rotinas e de clones, pois além de criar uma nova tarefa para o SO, nesse modelo é preciso compilar e executar um código inicial e depois carregar do disco o código que executará no *thread*. Também espera-se escalonamento como no caso anterior ($t_{colab}, t_{preemp} > 0$).

Processos:

Assim como no modelo de *threads*, espera-se um processamento adicional constante para a criação de tarefas ainda maior que para o modelo de co-rotinas e de clones ($t_{criaprocesso} > t_{criacorotina}, t_{criaclone}$). Pois além de criar uma nova tarefa para o SO, nesse modelo é preciso carregar do disco o código que executará no novo processo. Nos nossos casos de teste esse modelo não precisa compilar o código inicial, porém precisa carregá-lo do disco. Os SOs modernos otimizam a carga de executáveis binários, carregando somente as partes do binário a medida que ele é necessário e compartilhando áreas de código entre programas(rusling96linux, Oram02). Para baixa transferência de dados, baixo procesamento ($X, P \rightarrow 0$) e ausencia de concorência ($C = 1$) será possível comparar os pesos dos mecanismos de criação de *threads* e processos ($\frac{T_{resthread}}{T_{ressequencial}}e\frac{T_{resprocesso}}{T_{ressequencial}}$). Além disso poderemos medir a eficiência de transferir um trecho de memória contendo uma *string* e compilá-la com relação a carregar o código já compilado do disco. Também espera-se escalonamento como no caso dos *threads* ($t_{colab}, t_{preemp} > 0$).

6.2.2 Expectativas fixando o modelo de concorrência e variando o modelo de sandbox

Gostariamos de saber o modelo de *sandbox* mais adequado para cada modelo de concorrência.

Sem sandbox:

Espera-se que seja uma referência como limite superior nos gráficos dos modelos de concorrência ($t_{criasandbox}, t_{transfsandbox} = 0$), já que os outros modelos implicam em processamento adicional para a criação da *sandbox* e comunicação.

Venv:

Espera-se que adicione um processamento inicial para a criação da *sandbox* ($t_{criavenv} > 0$) e depois as chamadas de funções do ambiente do servidor para a transferência de dados ($t_{transfvenv} \rightarrow 0$) executem aproximadamente na mesma velocidade do modelo sem *sandbox*. Espera-se então que proporcionalmente ao modelo sem sandbox o desempenho se mantenha para todas as variações de dados transferidos($T_{resvenv} - T_{criavenv} \approx T_{ressemsandbox} \forall X$).

Rings:

Espera-se que adicione um processamento inicial ainda maior que o Venv ($t_{criarings} > t_{criatvenv}$) pois é preciso compilar um trecho de código sempre que se cria uma nova *sandbox*. Além disso, é preciso copiar trechos de memória para transferir dados entre ambientes($t_{transfrings} > 0$), então espera-se também um maior processamento para executar chamadas de funções do ambiente do servidor($T_{resrings} - T_{criarings} - T_{transfrings} \approx T_{ressemsandbox} \forall X$).

Então, para chamadas de pequena transferência de dados e pequeno processamento ($X, P \rightarrow 0$) e sem concorência ($C = 1$), poderemos comparar o peso da criação das *sandboxes* ($\frac{T_{resvenv}}{T_{ressemsandbox}}e\frac{T_{resrings}}{T_{ressemsandbox}}$). Nas chamadas de maior transferência de dados ($X \rightarrow \infty$), esperamos que essa carga se torne irrelevante ($T_{criarings} << T_{resrings}$). Poderemos assim verificar o peso da execução de chamadas entre máquinas virtuais Lua ($T_{transfrings} \approx T_{resrings} - T_{ressemsandbox}$).

Processo separado:

Espera-se que apresente o menor desempenho em todos os casos já que o processamento inicial demanda a criação de um novo processo e a carga de código do disco ($t_{criaprocessosep} > t_{criavenv}, t_{criarings}$). A partir daí a comunicação com o servidor é realizada através de *sockets*, o que implica na necessidade de cópias de memória e na espera por mecanismos de sinalização no núcleo do SO para realizar a E/S dos *sockets* (t_{transf} e $t_{E/S}$ maiores). Espera-se uma degradação semelhante à imaginada no modelo seqüencial. Como a execução de chamadas depende de um mapeamento realizado por um protocolo de comunicação, espera-se uma degradação ainda maior que o Rings, tanto na criação quanto na transferência de dados para o servidor.