7
Resultados

De acordo com as expectativas apresentadas no capítulo anterior, separamos a análise dos resultados de forma que cada modelo de concorrência pudesse ser analisado em função do modelo de *sandbox* e vice versa.

A seguir são mostrados gráficos agrupados por modelos. Para cada um dos 9 modelos estudados são apresentados 9 gráficos. Esses gráficos mostram como os sistemas se comportam em função de cada combinação de fatores. Olhando cada gráfico individualmente, o eixo x representa o número de requisições concorrentes que foram geradas para estimular o sistema. O eixo y representa o número de respostas por segundo que o sistema conseguiu gerar. Olhando para o conjunto de 9 graficos organizados em 3 linhas e 3 colunas, nos graficos que aparecem na mesma linha o sistema gerou saídas do mesmo tamanho enquanto nos gráficos da mesma coluna o sistema gerou o mesmo volume de processamento.

Optamos por ampliar cada gráfico o máximo possível, ao invés de manter uma escala única ou por grupos, já que as escalas variaram em até duas ordens de grandeza nos testes de um mesmo modelo.

Para cada conjunto de 9 gráficos de um determinado modelo segue uma análise comparativa entre os modelos representados nas linhas. Nessa análise é descrito o que era esperado e o que realmente ocorreu.

Iniciaremos a análise mostrando os resultados obtidos no sistema base (Linux-Celeron com janela TCP de 4K) e depois comparamos esses resultados com o comportamento das outras plataformas.

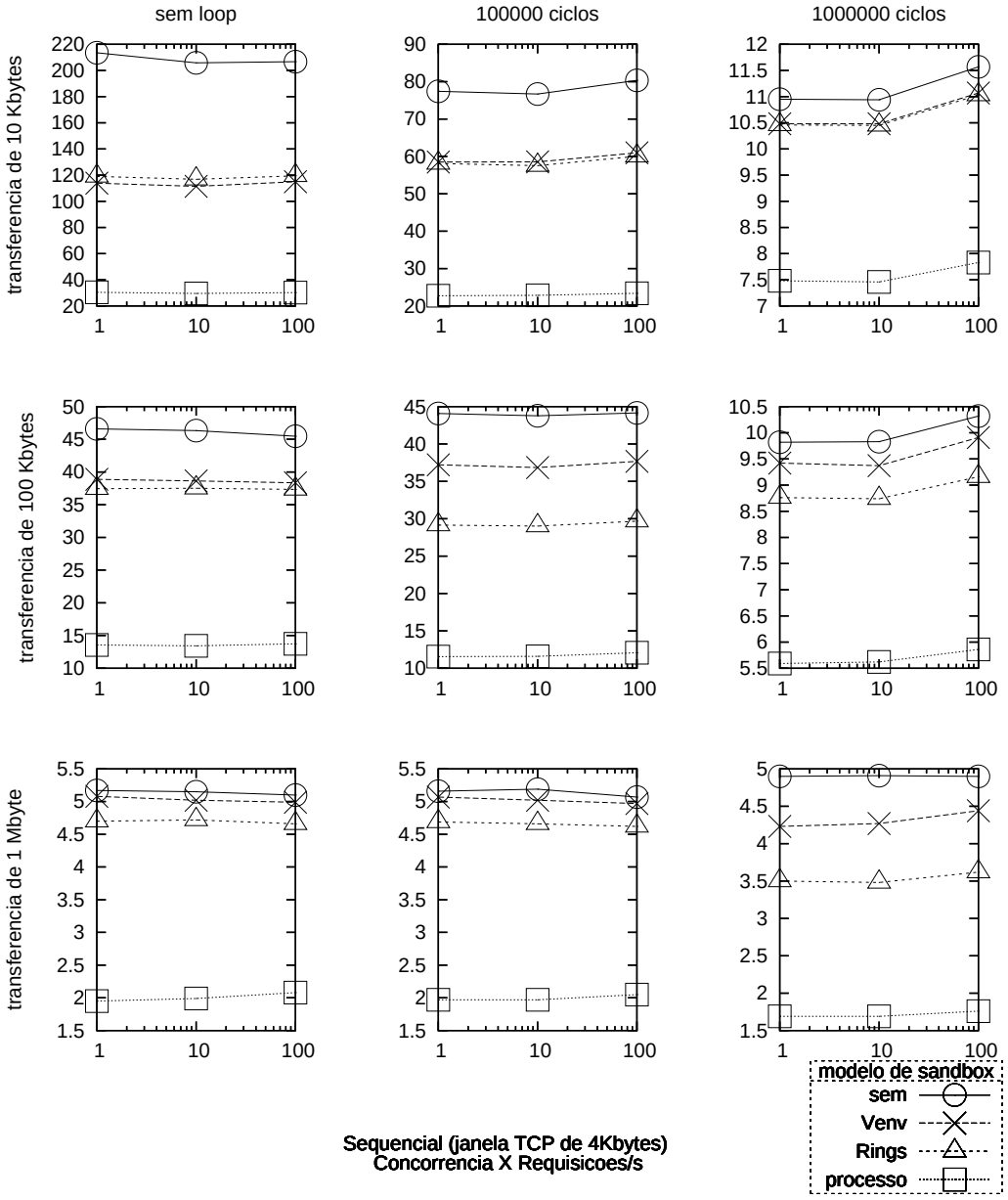
7.1
Resultados no sistema base

O sistema base consiste em uma máquina Linux-Celeron com a janela TCP fixada em 4Kbytes. Como se trata do sistema mais limitado do teste, consideramos esse sistema o patamar inicial para os testes.

7.1.1

Resultados fixando-se o modelo de concorrência e variando o modelo de sandbox

Cada grupo de gráficos representam os resultados de um modelo de concorrência. As linhas representam os modelos de *sandbox*.



Seqüencial

Esperado: Esperávamos uma resposta constante em relação ao número de requisições concorrentes e variação linear em relação ao volume de processamento e de transferência.

Obtido: Obtivemos a tendência esperada.

– Sem *sandbox*

Esperado: Esperávamos que sempre fornecesse resultados melhores que os outros modelos.

Obtido: O modelo sem *sandbox* conseguiu o melhor desempenho em todos os casos conforme o esperado.

– Venv

Esperado: Esperávamos uma resposta constante em relação ao número de requisições concorrentes, com resultados piores que o modelo sem *sandbox* e melhores que o molelo Rings.

Obtido: Os casos de baixa transferência de dados ($X = 10Kbytes$) surpreenderam, obtendo um resultado praticamente idêntico ao modelo Rings. Isso indica que o tempo de criação da *sandbox* ($t_{criasandbox}$)são comparáveis no contexto de um servidor web. Com o aumento da transferência de dados ($X = 100Kbytes, 1Mbyte$) notamos que o desempenho do modelo Rings se degrada em relação ao Venv, nos levando a conclusão de que os mecanismos de transferência entre *sandbox* e servidor são mais eficientes no Venv que no Rings. Também notamos que o aumento da transferência de dados faz com que o desempenho do modelo Venv se aproxime do desempenho do modelo sem sandbox. Isso indica que a sobrecarga do mecanismo de comunicação do Venv é bastante baixa.

– Rings

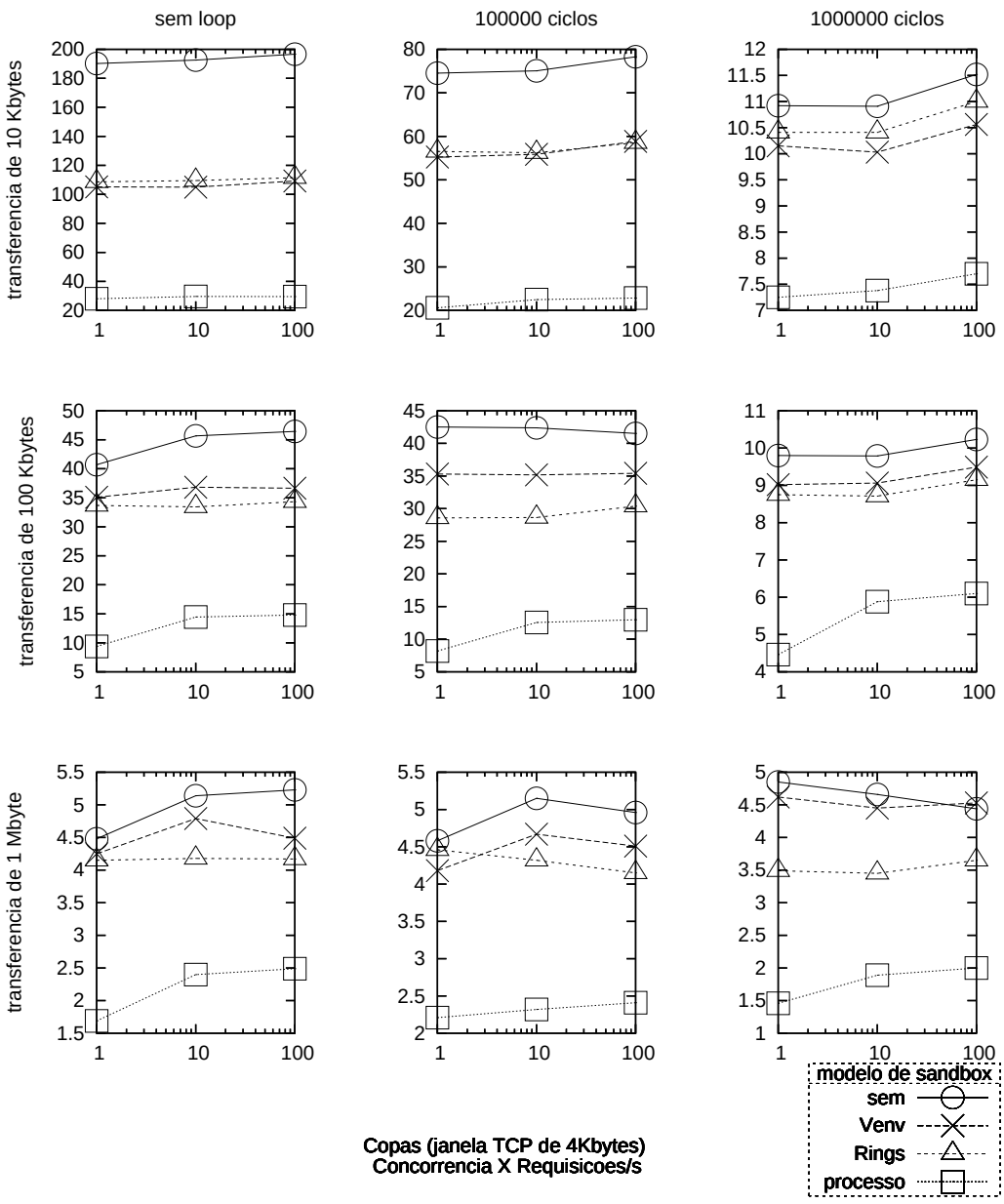
Esperado: Esperávamos uma resposta constante em relação ao número de requisições concorrentes, com resultados piores que o modelo Venv e melhores que o modelo de processos separados.

Obtido: Quando comparado com o resultado obtido pelo modelo de processos separados, obtivemos resultados até 3,9 vezes melhores.

– Processo separado

Esperado: Esperávamos uma resposta constante em relação ao número de requisições concorrentes, com um resultado pior que o do modelo Rings.

Obtido: Obtivemos o descrito no item anterior.



Co-rotinas

Esperado: Esperávamos que quanto maior o número de requisições concorrentes (C) e maior a transmissão, maior seria o *throughput* do servidor de dados (X).

Obtido: No caso de baixo processamento ($t_{loop} = 0$) é possível notar, em praticamente todos as combinações (10 das 12 linhas), um resultado melhor com o aumento da concorrência. Quando se aumenta C de 1 para 10 aparece uma melhora de até 54% , porém a partir daí o resultado tende a se manter constante para $C = 100$. Os resultados foram até 1,4 vezes mais lentos que os resultados do modelo seqüencial.

– Sem *sandbox*

Esperado: Esperávamos a linha dos resultados sem *sandbox* como uma referência de limite superior já que os outros modelos implicam em processamento adicional para a criação da *sandbox*.

Obtido: Obtivemos a tendência esperada.

– Venv

Esperado: Esperávamos uma degradação semelhante à esperada no modelo seqüencial, com resultados piores que o modelo sem *sandbox* e melhores que o molelo Rings.

Obtido: Mais uma vez o caso de baixa transferência de dados ($X = 10Kbytes$) surpreendeu gerando um resultado praticamente idêntico ao do modelo Rings.

– Rings

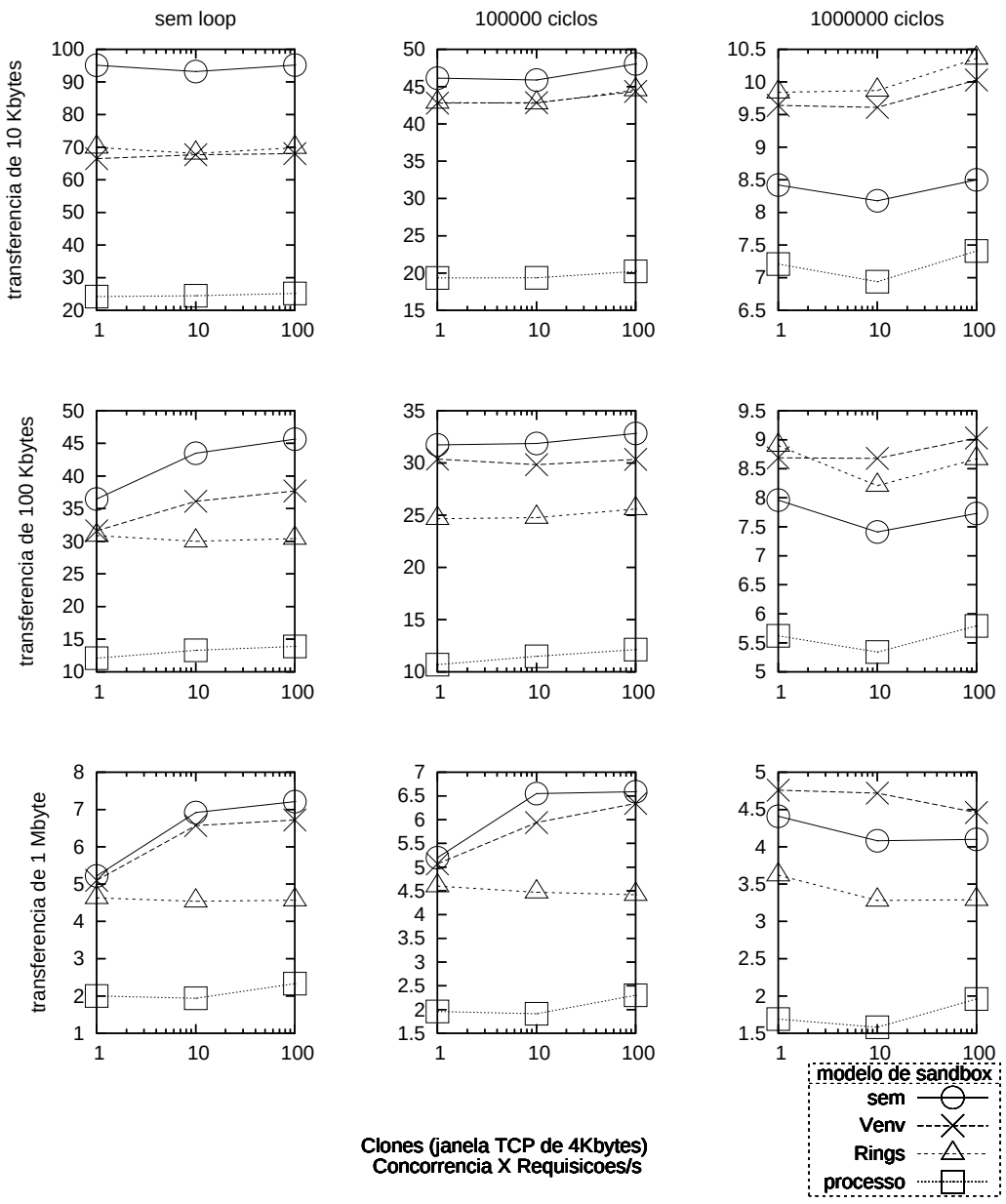
Esperado: Esperávamos uma degradação semelhante à esperada no modelo seqüencial, com resultados piores que o modelo Venv e melhores que o modelo de processos separados.

Obtido: Como já foi descrito no Venv, o resultado foi praticamente idêntico ao do modelo Venv.

– Processo separado

Esperado: Esperávamos uma degradação semelhante à esperada no modelo seqüencial,com um resultado pior que o do modelo Rings.

Obtido: Nesse modelo a melhora com o aumento da concorrência pode ser observada de forma mais consistente e acentuada.



Clones

Esperado: Esperávamos comportamentos semelhantes aos do modelo de co-rotinas, Esperávamos também que o peso da criação de um clone seja maoior que o peso de criação de uma corrotina, fazendo que o *throughput* dos sistemas sejam menores que o do modelo de co-rotinas.

Obtido: O comportamento foi semelhante ao obtido com o modelo de co-rotinas, com exceção do modelo sem *sandbox*, quando o processamento é alto (P=1000000). Os resultados foram até 2,1 vezes mais lentos que os do modelo de co-rotinas.

– Sem *sandbox*

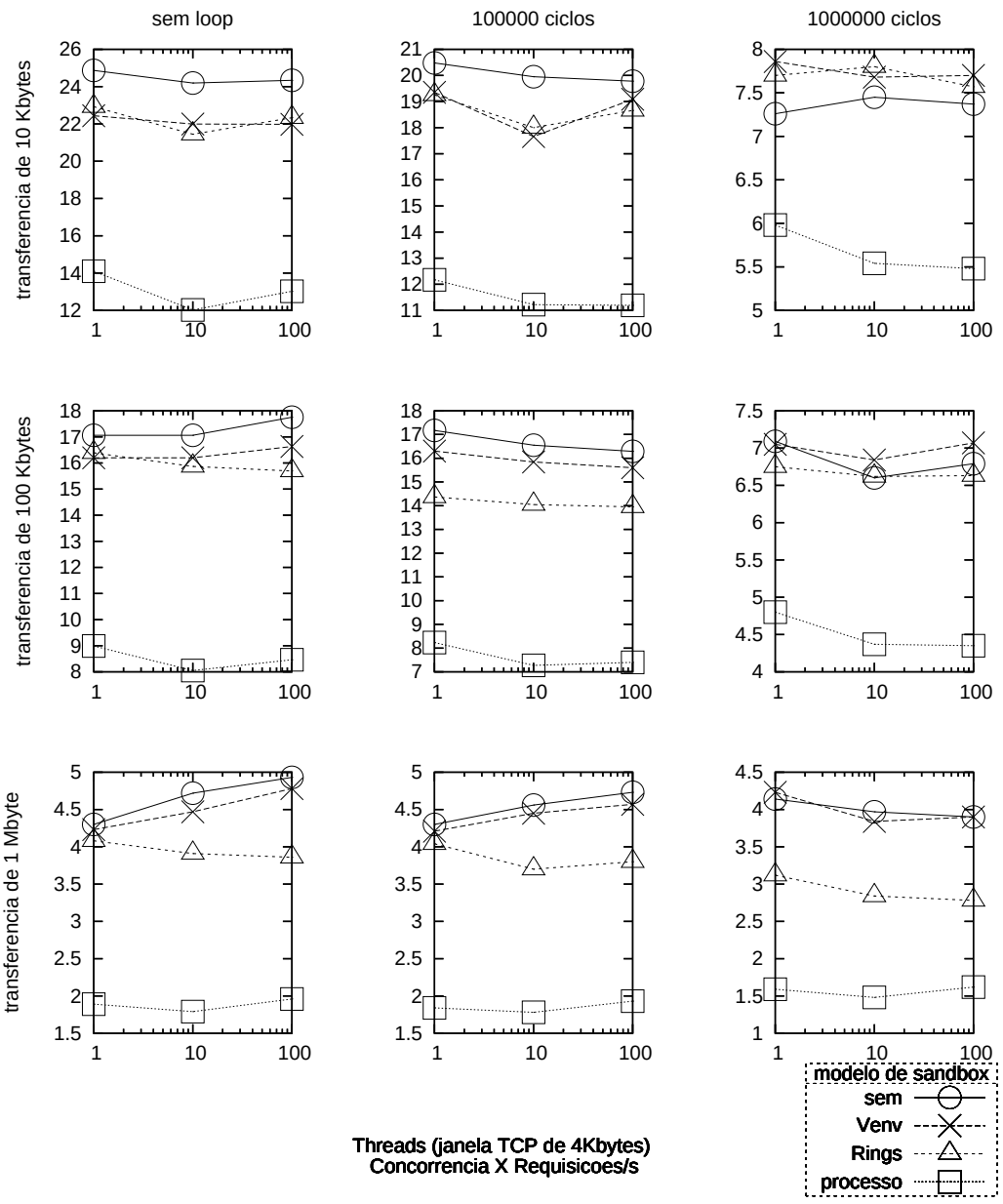
Esperado: Esperávamos que sempre fornecesse os melhores resultados.

Obtido: Aparece com nitidez uma anomalia. O modelo sem *sandbox* aparece como limite superior com exceção de quando o processamento é alto (P=1000000). Esse resultado inesperado, fora do modelo matemático levantado anteriormente, nos levou a realizar experimentos mais detalhados dessa combinação de fatores. Observamos que o tempo de resposta de um código Lua pode variar dependendo da posição onde as variáveis são alocadas na pilha da máquina virtual Lua. Testes realizados com uma máquina virtual compilada com 40 posições de pilha ao invés de 20 fizeram o desempenho desse modelo ultrapassar os outros modelos como era inicialmente esperado. Os resultados desses testes extras podem ser vistos no apêndice A.

– Venv, Rings e Processo separado

Esperado: Assim como no modelo seqüencial, esperavamos que a lista de modelos em ordem de desempenho fosse: sem *sandbox*, Venv, Rings e Processo separado.

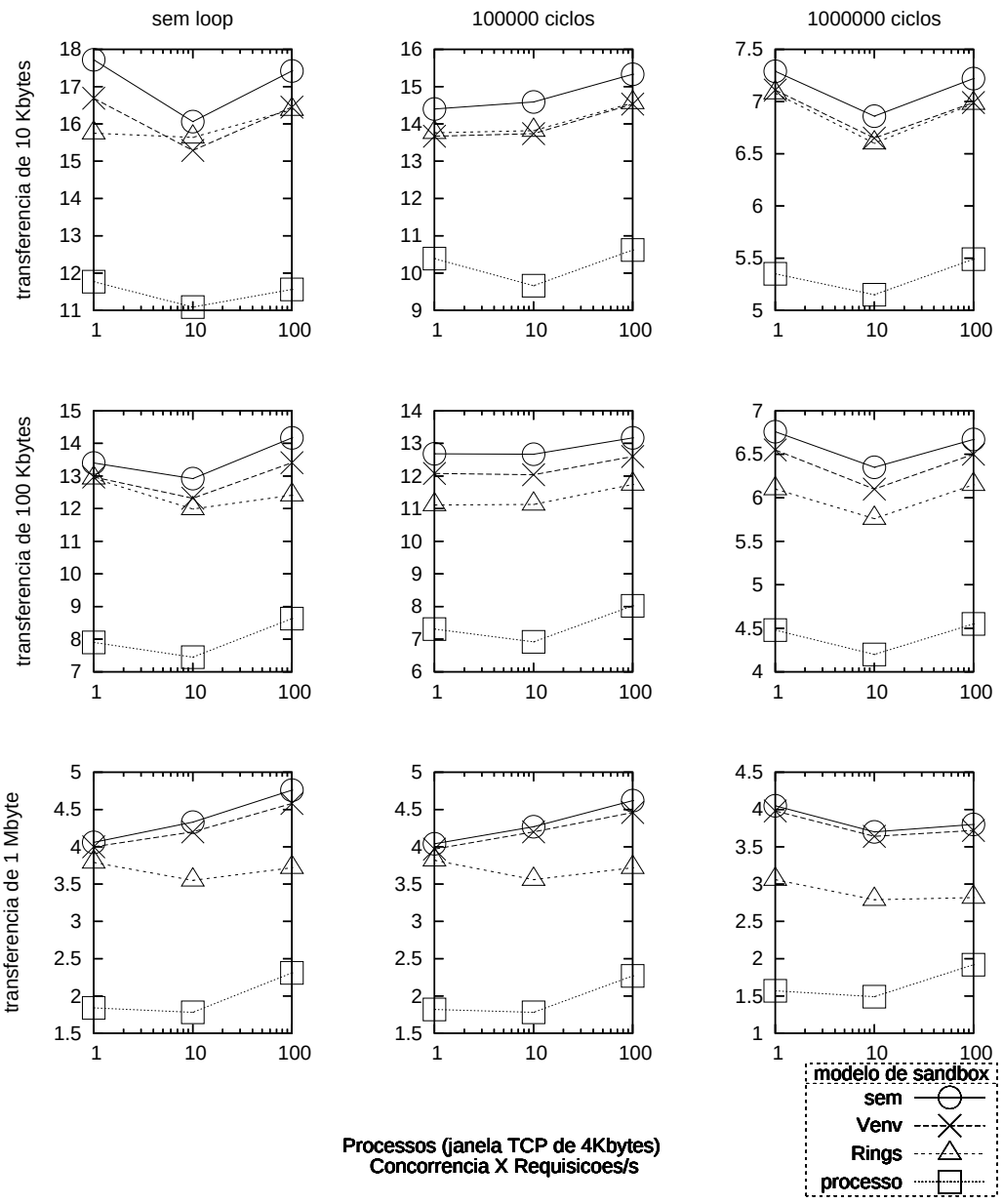
Obtido: Obtivemos resultados semelhantes aos modelos de concorrência anteriores, com o Rings se aproximando do Venv em baixos volumes de transferência.



Threads

Esperado: Esperávamos um comportamento dos modelos *de sandbox* semelhante ao apresentado nos gráficos dos modelo de co-rotinas e clones, porém, com menor desempenho.

Obtido: As tendências dos resultados do modelo de clones se repetiram, inclusive o comportamento do modelo sem *sandbox* com o processamento alto. Conforme esperado, os resultados foram até 3,9 vezes mais lentos que os do modelo de clones.



Processos

Esperado: Esperávamos um comportamento dos modelos *de sandbox* semelhante semelhante ao comportamento apresentado nos modelos de co-rotinas, clones e *threads* porém com desempenhos ainda inferiores aos obtidos no modelo de *threads*

Obtido: Obtivemos a tendência esperada. O caso sem sandbox apareceu como melhor resultado porém com pouca vantagem, não sendo possível dizer se houve influência da anomalia detectada nos modelos de clones e *threads*. Conforme esperado, os resultados foram até 1,5 vezes mais lentos que os do modelo de *threads*.

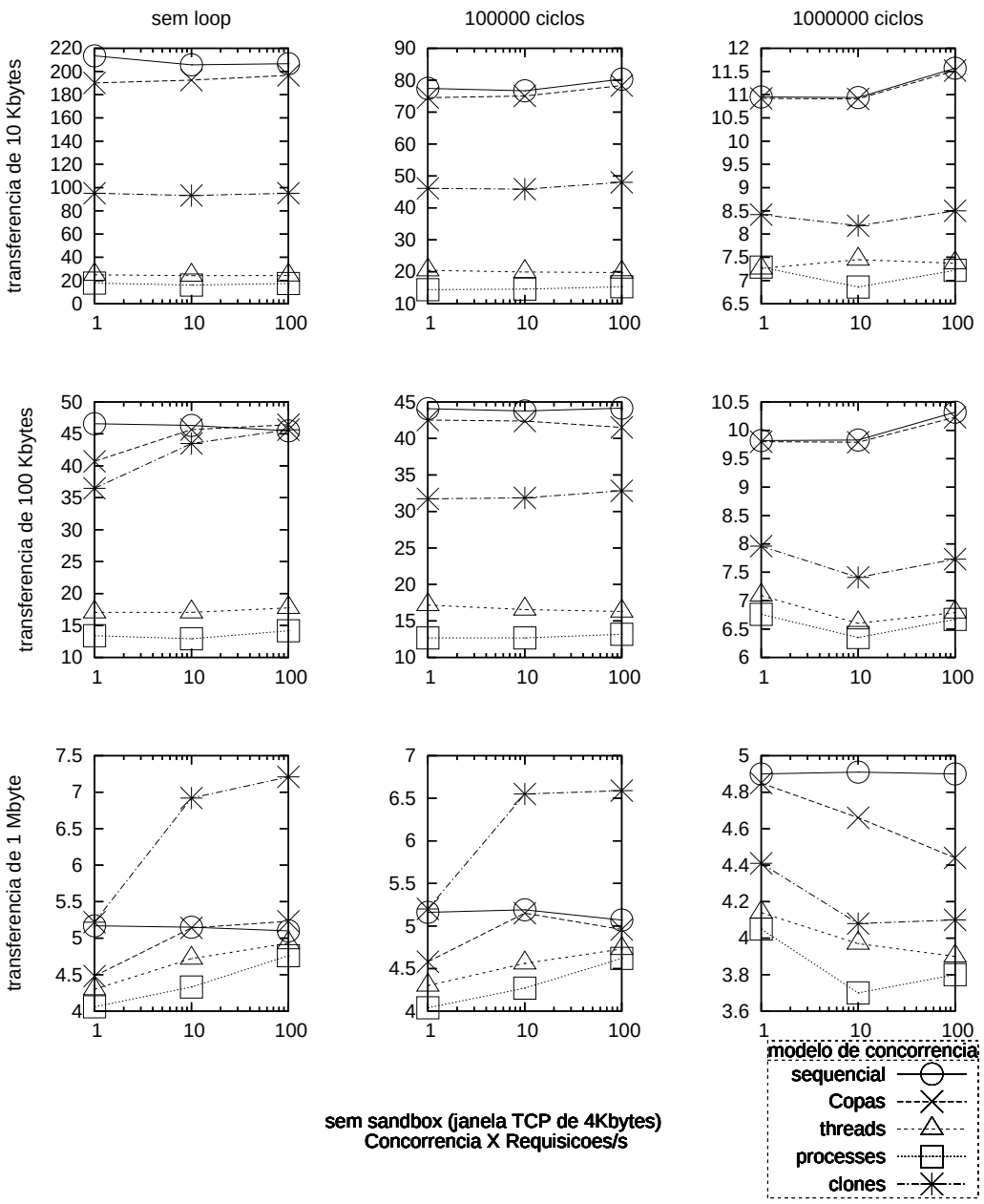
Conclusões parciais

Observou-se um novo parâmetro inerente ao sistema que afeta a carga de processamento das aplicações. Esse parâmetro é a posição onde as variáveis são alocadas na pilha da máquina virtual Lua que está relacionado com o número de registradores da máquina virtual Lua. Poderíamos eliminar o efeito da sua alocação dinâmica compilando a máquina virtual com um número inicial maior de registradores porém preferimos nesse trabalho é utilizar a máquina virtual Lua padrão. Existe então uma nova preocupação a ser adicionada a trabalhos desse tipo, se um mesmo trecho de código tem desempenho diferente dependendo de onde ele entra no cóigo fonte do sistema.

De forma geral os modelos de sandbox Venv e Rings obtiveram resultados muito semelhantes e em alguns casos se aproximaram do modelo sem *sandbox*. O modelo de processo como *sandbox* foi mais lento que os outros modelos em todas as combinações de fatores.

7.1.2 Resultados fixando-se o modelo de sandbox variando o modelo de concorrência

Cada grupo de gráficos representam os resultados de um modelo de *sandbox*. As linhas representam os modelos de concorrência.



Sem sandbox

Esperado: Esperávamos que o aumento de concorrência melhorasse o desempenho de todos os modelos.com exceção do modelo seqüencial

Obtido: Obtivemos a tendência esperada. Chama a atenção o caso de transferência de dados grande com pouco processamento ($X = 1Mbyte$ e $L = 1$ ou $L = 100000$) o modelo de clones obteve o melhor desempenho enquanto os outros modelos mal conseguiram alcançar o modelo seqüencial. Optamos por levantar o desempenho nessas condições com um nível maior de concorrência ($C = 500$ e $L = 1$). O resultado segue na tabela 7.1.

Modelo	respostas/s
seqüencial	3,86
Copas	4,37
threads	5,09
processos	4,88
clones	7,37

Tabela 7.1: Transferência de 1Mbytes, com 500 conexões concorrentes, sem processamento extra.

– Seqüencial

Esperado: Esperávamos o melhor desempenho para o caso de requisições seqüenciais ($C = 1$), por ser o único modelo que não precisa criar uma nova tarefa para processar a requisição, e que se mantenha constante nos outros valores de concorrência. A medida que a concorrência aumentasse os outros modelos tenderiam a responder melhor em relação ao modelo seqüencial.

Obtido: Obtivemos a tendência esperada.

– Co-rotinas

Esperado: Esperávamos um desempenho melhor em relação ao desempenho do modelo seqüencial quanto maior for o número de requisições concorrentes (C) e o volume de E/S (X) possivelmente obtendo um resultado melhor que o do modelo seqüencial

Obtido: Observou-se a melhoria no desempenho com o aumento de C somente para baixos volumes de processamento, porém em momento algum esse modelo foi melhor que o seqüencial. Isso indica que a soma do tempo de processamento em escalonamento colaborativo mais o tempo que o escalonador fica esperando bloqueado em E/S é maior que o tempo bloqueado em E/S do modelo seqüencial ($t_{E/Ssequencial} < t_{colab} + t_{E/Scolaborativo} \forall C \rightarrow 1$). Para levantar se t_{colab} está fazendo o sistema atrasar mais mais que $t_{E/Ssequencial}$, recorremos a tabela 7.1. Nessa tabela o Copas apresenta um desempenho ligeiramente melhor que o seqüencial ($t_{E/Ssequencial} > t_{colab} + t_{E/Scolaborativo} \forall C \rightarrow 500$) logo $t_{E/Ssequencial} > t_{colab}$ isso significa que escalonamento colaborativo tem um desempe-

nho melhor que o seqüencial para a combinação de grandes volumes de dados e grandes níveis de concorrência.

– Clones

Esperado: Esperávamos um comportamento parecido com o obtido no modelo de co-rotinas, porém também esperávamos que o desempenho piorasse em relação ao modelo de co-rotinas quanto maior fosse o volume de processamento.

Obtido: Esse modelo surpreendeu. O maior volume de E/S em condições de baixo processamento levou esse modelo a apresentar a melhor resposta. O resultado do modelo de clones e dos outros modelos preemptivos para $C = 500$ conforme mostra a tabela 7.1 mostra que o escalonamento em E/S apresenta um desempenho melhor quando realizado pelo escalonador do Linux do que pelo Copas.

– *Threads*

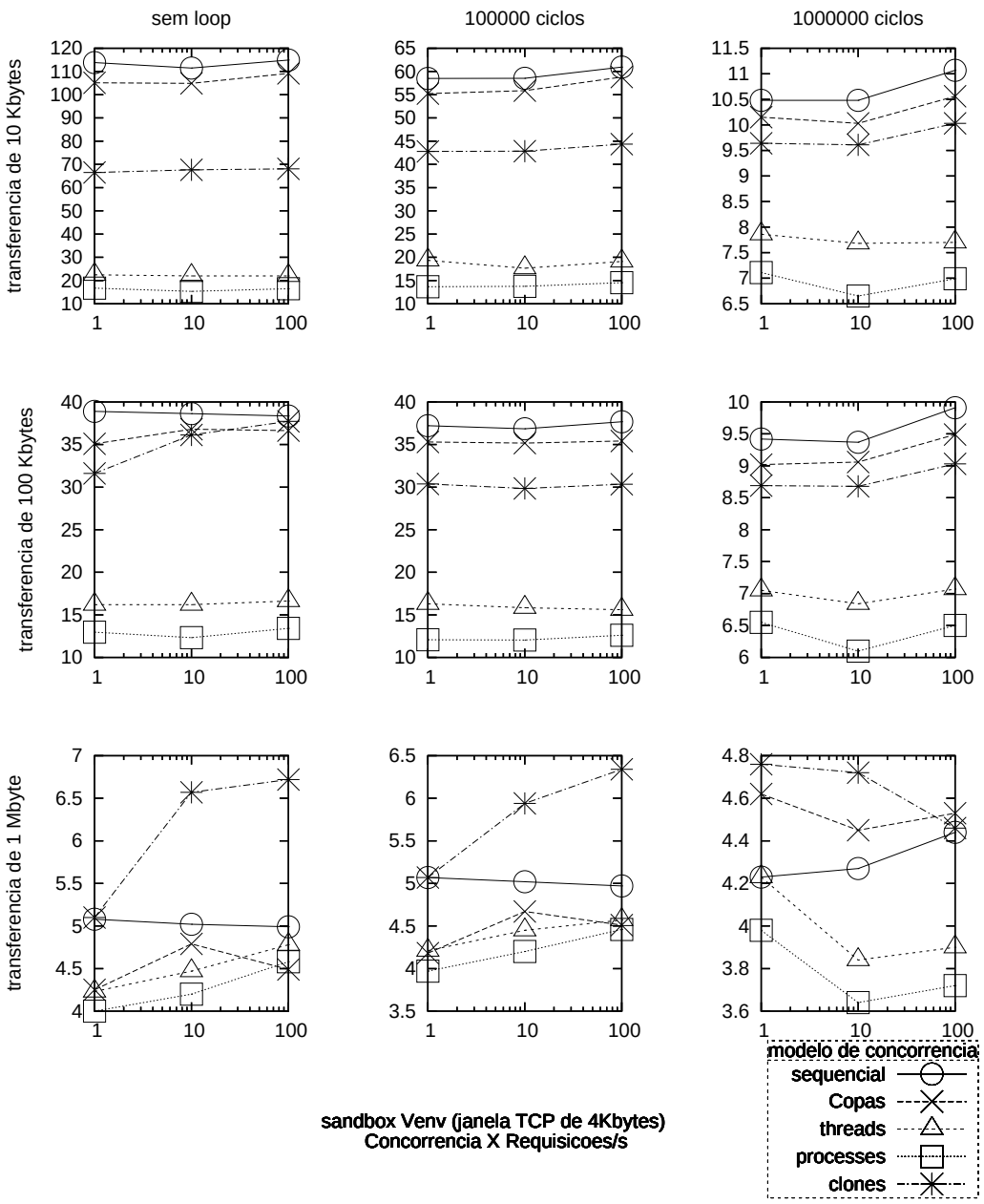
Esperado: Assim como para os clones, esperávamos que o desempenho em relação ao modelo de co-rotinas piorasse quanto maior fosse o volume de processamento. Esperávamos também, de acordo com a seção 6.2.1, que o peso inicial para a criação de um novo *thread* fosse maior que o de clonagem ou o de criação de co-rotinas.

Obtido: Obtivemos a tendência esperada, com exceção ao desempenho obtido no experimento complementar descrito no item anterior.

– Processos

Esperado: Esperavamos um resultado semelhante ao do modelo de *threads* porém que o peso inicial para a criação do processo fosse maior que o de clonagem ou o de criação de co-rotinas ou de criação de *threads*.

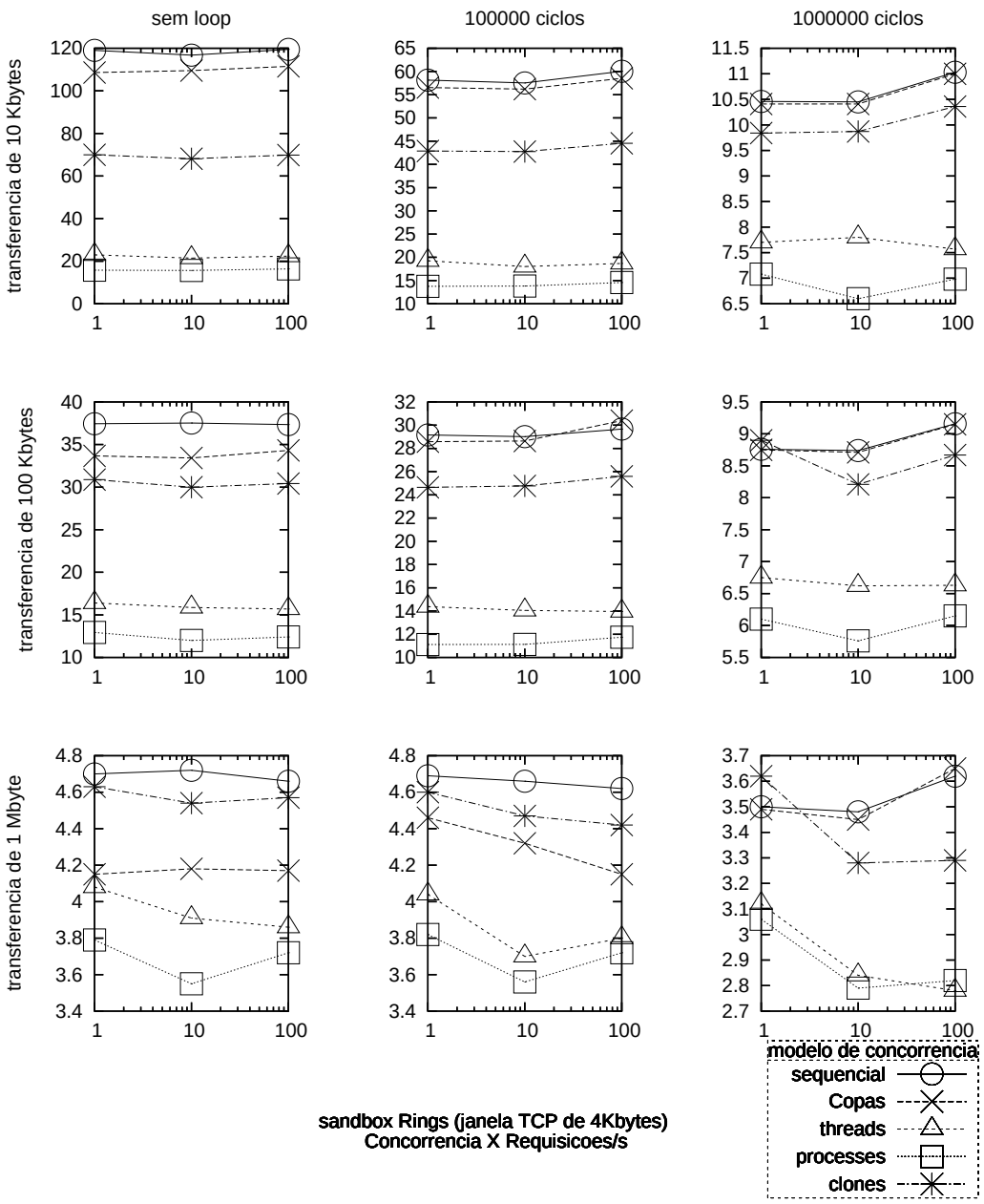
Obtido: Obtivemos a tendência esperada, com exceção ao desempenho obtido no experimento complementar descrito no item anterior.



Venv

Esperado: Esperávamos um comportamento semelhante ao modelo sem *sandbox*

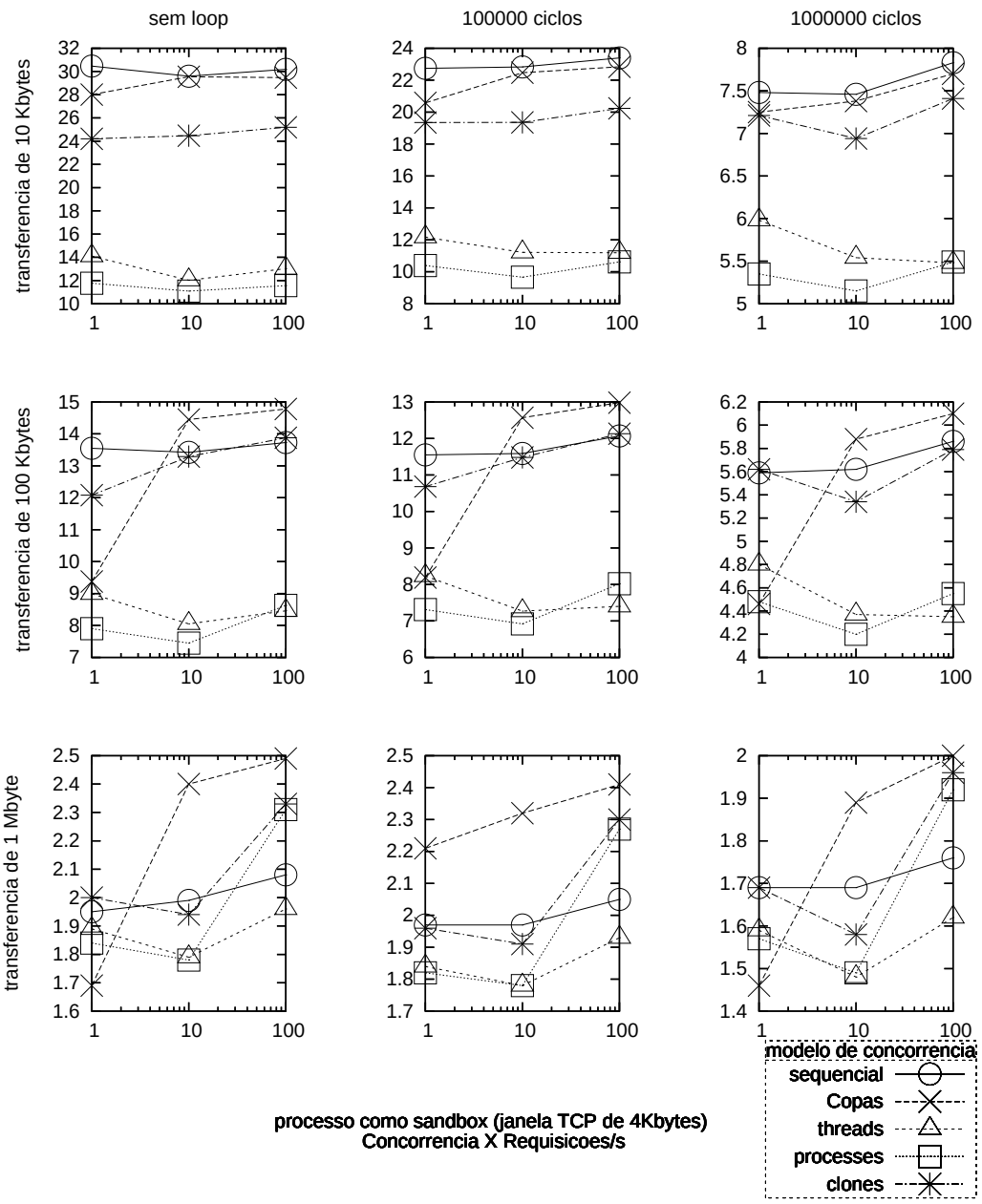
Obtido: Obtivemos a tendência esperada, com exceção do caso com alto processamento e alto volume de transferência que apresentou claramente a anomalia descrita anteriormente. Os resultados sem o efeito dessa anomalia podem ser vistos no apêndice A.



Rings

Esperado: Esperávamos um comportamento semelhante ao modelo sem *sandbox*

Obtido: Obtivemos a tendência esperada.



Processo separado

Esperado: Esperávamos um comportamento semelhante ao modelo sem *sandbox* com exceção do modelo de co-rotinas que deve sofrer uma melhoria quando comparado com o modelo sem *sandbox* pois teoricamente aumenta a E/S e o escalonamento nesse modelo.

Obtido: O modelo de co-rotinas apresentou uma melhoria acentuada com o aumento da concorrência. O modelo de co-rotinas até passou a responder melhor que todos com o aumento do volume de transferência de dados.

7.1.3 Conclusões parciais

O modelo seqüencial obtém o melhor resultado em quase todas as combinações de fatores. Isso indica que mesmo tendo diversas tarefas competindo por E/S o sistema tem capacidade de E/S ociosa. O modelo de co-rotinas é o que apresenta o melhor desempenho para processos como *sandboxes*, provavelmente porque aumenta a concorrência entre as tarefas e consegue usar melhor o tempo que não está conseguindo fazer E/S. O modelo de clones para alto volume de transferência e baixo processamento apresentou o melhor desempenho. Quando a rajada de processamento é baixa, o número de vezes que o processo sofre escalonamento preemptivo também será baixo ou nulo. Quase todo o escalonamento ocorre quando E/S bloqueia. O resultado superior dos clones para alto volume de transferência e baixo processamento indica que o escalonamento em E/S bloqueado é mais leve no modelo de clones que no modelo de co-rotinas.

7.2 Expectativas e resultados no sistema com a janela TCP de 128Kbytes

A expectativa era obter um desempenho melhor em todos os casos. Com a janela TCP de 128Kbytes foi possível atingir até o dobro do *throughput* do sistema base para as transferências de 1Mbyte com baixo processamento. Isso indica melhoria do desempenho tanto para transferências maiores como para volumes menores de processamento. Para a situação inversa não notamos nenhuma diferença expressiva, já que o peso da transferência de dados é menor nesses casos. A diferença de desempenho entre Venv e Rings da transferência de dados da sandbox para o servidor que apareceu no sistema base, nesse sistema aparece mais acentuada. Para altos volumes de processamento não se nota diferença para o sistema base.

Não aparece melhoria no desempenho dos modelos de concorrência com o aumento das requisições concorrentes (C).

O modelo de clones não aparece mais com o maior desempenho para volumes de transferências mais altos. O desempenho do Copas, diferentemente do resultado no sistema base, não se mostrou melhor que o dos outros modelos para processos como *sandbox*. Esses comportamentos indicam que houve menos escalonamento em E/S conforme o esperado.

7.3

Expectativas e resultados no sistema com o SO Windows

Como no Windows o tamanho da janela TCP ficou sendo controlada automaticamente pelo SO, era esperado que o desempenho fosse intermediário ao sistema base com janela de 4K e ao sistema com janela de 128K.

De maneira geral o sistema Windows atingiu um desempenho intermediário entre o sistema base e o com janela de 128K. As exceções foram os modelos de concorrência de threads e processos, onde o desempenho foi pior que o do sistema base.

A anomalia relativa à posição em que as variáveis são alocadas na pilha da máquina virtual Lua não apareceu como no sistema base.

7.4 Expectativas e resultados no sistema com 4 processadores

Esperávamos que os modelos seqüencial e multi-corrotina não apresentassem diferença de desempenhos relativos de um sistema com 1 processador para o de 4, dado que executam em apenas um desses. Esperávamos um desempenho muito melhor dos modelos *multi-threads* multi-processos e de clones que o do seqüencial.

Em todos os modelos, mesmo no modelo seqüencial o desempenho melhorou com o aumento da concorrência. Provavelmente houve melhoria porque o próprio SO concorre por processamento com o servidor web que está sendo testado.

No modelo Copas o aumento do processamento fez o modelo de processos como *sandbox* apresentar o melhor desempenho. Atribuímos isso ao aumento dos atendimentos concorrentes, já que 4 processos *sandbox* podiam ser executados simultaneamente.

O modelo de *threads* apresentou o pior desempenho em transferências de dados pequenos. Desconfiamos que estávamos usando algum tipo de *thread* a nível do usuário e não a nível do núcleo do SO. Já que esse tipo de *thread* implementa o escalonador dentro do próprio processo e não tem como fazer uso de múltiplos processadores. Verificamos a implementação de threads do SO e descobrimos que se trata da NPTL (*Native POSIX Thread Library for Linux*) (NPTL) que implementa os threads no núcleo do SO e suporta a arquitetura SMP (*Symmetric Multi-Processing*) (aas-understanding, linux26) que estávamos usando. Atribuímos então o desempenho pior do modelo de threads a algum algoritmo de afinidade de processador no escalonador do Linux 2.6 SMP. O escalonador deve procurar colocar múltiplos threads do mesmo processo na mesma CPU. Para dados pequenos os modelos seqüencial e Copas obtiveram um desempenho uma ordem de grandeza maior que o do

modelo de *threads*. Até o modelo de processos obteve um desempenho mais que 100% melhor que o modelo de *threads* nessa situação. Quando o volume de dados aumentou, *threads* apresentou um desempenho ligeiramente melhor que os modelos seqüencial e Copas.

O modelo de clones apresentou o melhor desempenho em todas as combinações de fatores com exceção das combinações onde a transferência de dados era pequena ($X = 10Kbytes$).

7.5 Expectativas e resultados no sistema com atraso de 50 ms

Como foi insserido um atraso de 50 mili-segundos em todos os pacotes do teste esperávamos que todos os resultados fossem mais lentos que o sistema base, principalmente nos casos de baixa transferência de dados. Esperávamos também um comportamento semelhante dos modelos de *sandbox* em relação ao sistema base

Observando os resultados do modelo seqüencial, nota-se que quando o número de requisições concorrentes aumenta de um para qualquer valor maior há um ganho, relativo aos pedidos das próximas conexões já estarem entregues antes que a anterior termine, aparecendo no gráfico como uma rampa seguida de uma linha constante. Continuando a análise do modelo sequencial, os diversos modelos de *sandbox* obtiveram paraticamente o mesmo desempenho, o que indica que o atraso na comunicação tornou a espera em E/S bloqueado tão grande que os tempos de criação e comunicação com a sandbox se tornaram desprezíveis.

Nesses resultados nota-se que os modelos de co-rotinas e clones melhoram o seu desempenho em uma taxa maior que o aumento da concorrência. Isto é: de 1 para 10 requisições concorrentes o desempenho aumenta nos modelos de atendimento concorrente, porém de 10 para 100, os modelos de co-rotinas e clones melhoram ainda mais o seu desempenho enquanto os outros modelos apresentam pouco ou nenhum ganho. Como clones e co-rotinas conseguem responder às requisições em taxas consideravelmente maiores, isso indica que os outros modelos não saturaram a capacidade da rede e sim algum outro ponto do sistema, como por exemplo CPU ou acesso a disco.

Co-rotinas obtiveram consistentemente um melhor desempenho e como os resultados sem a imposição de atraso na comunicação conseguiram desempenhos ainda maiores podemos concluir que o canal de comunicação não saturou em nenhum dos testes. A partir dai suponho que a taxa de resposta do modelo de co-rotinas e clones continuaria a melhorar com o aumento da concorrência até o momento que o canal de comunicação ficasse saturado.

Fica claro que observando os gráficos dos diversos modelos de concorrência que o peso da criação da *sandbox* Rings é praticamente igual a criação da Venv, porem a comunicação com a *sandbox* Venv é mais rápida que com a *sandbox* Rings.

7.6 Conclusões

O controle da janela TCP fez pouco diferença para emular canais de comunicação mais lentos (como numa rede geograficamente distribuída). A imposição de um atraso no envio e recebimento dos pacotes funcionou de forma mais adequada.

Do ponto de vista do desempenho, quando o atraso do canal de comunicação é baixo (como numa rede local), os diversos modelos de concorrência apresentam resultados na mesma ordem de grandeza. O modelo que mostrou o melhor desempenho na maioria dos casos foi o modelo seqüencial. Daí concluise que a espera por E/S na rede é baixa demais para valer algum ganho que realmente chame a atenção com escalonamento na maioria das condições apresentadas (não passa de 1,4 vezes e ocorre no clone transferindo 1Mbyte sem *loop* nem *sandbox*). Em algumas situações específicas, o desempenho de um dos modelos se sobressai.

Para o caso de uma rede geograficamente distribuída, o uso de qualquer modelo que permita atender requisições concorrentemente pode resultar em grandes ganhos quando comparado com o modelo seqüencial. O mesmo provavelmente ocorre caso os clientes demorem para enviar seus pacotes ao servidor. Como com o atraso de 50 ms o modelo de co-rotinas foi mais rápido em praticamente todos os casos, seguido de perto pelo modelo de clones, concluímos que obtemos algum ganho quando não realizamos escalonamento preemptivo e que a criação de uma co-rotina é mais leve que a criação dos outros mecanismos.

O modelo de concorrência ideal muitas vezes aparece em função do comportamento desejado entre as tarefas ou das facilidades que cada modelo oferece para o programadores.

O modelo seqüencial, por não oferecer nenhum tipo de processamento concorrente, é desaconselhável se é necessário interações entre tarefas na mesma máquina, já que se uma tarefa para, esperarando a resposta da outra, não vai deixar a outra executar porque o processamento está parado.

Todos os outros modelo testados oferecem algum tipo de processamento concorrente. Co-rotinas com escalonamento em E/S são ligeiramente mais leves e portáteis que threads e processos já que não demandam a carga de novos executáveis e facilitam a vida do programador por dispensar mecanismos de

sincronização para acessar dados compartilhados entre tarefas. Porém, têm a desvantagem de aumentar a interdependência entre as tarefas. Nesse modelo uma tarefa tem o poder de parar toda a aplicação até mesmo com um erro simples como erro em um teste para sair de um *loop* que nunca torne a condição do fim do *loop* verdadeira.

O modelo de *threads* espera que o SO forneça fatias de tempo para todos os *threads* processarem regularmente. Porém, o acesso a dados compartilhados demanda o uso de mecanismos de sincronização. O problema é que a falta ou mau uso dos mecanismos de sincronização, conforme descrito na seção 2.2.1, muitas vezes só apresenta erros depois de muitas execuções e por isso são difíceis de serem identificados e consertados.

O modelo de processos é o que oferece o maior isolamento entre as tarefas, indicado para quando se deseja que uma tarefa tenha o mínimo possível de influência sobre as outras, a exemplo de processos CGI em servidores web. Assim como *threads*, esse modelo espera o SO fornecer fatias de tempo para todos os processos executarem regularmente. Porém, quando comparados com os *threads* quanto a troca de dados, os mecanismos são mais complicados.

Concluímos pelo desempenho apresentado pelo modelo de *threads* na máquina com 4 processadores que a utilização de *threads* não garante uma utilização ótima dos processadores. Essa utilização depende do algoritmo de escalonamento do SO.

Obtivemos uma redução de desempenho significativa no uso da semântica de criação de processos com a semântica do Windows no Linux, principalmente quando levado em conta apenas o peso da criação do processo, porém, dependendo do peso da aplicação, vimos que essa diferença pode se tornar desprezível. Um fato interessante é que, mesmo no Windows, a criação de processos teve o desempenho pior que o Linux imitando a sua semântica, o que mostra que essa abordagem do problema tende a ser mais lenta em qualquer plataforma.

Esperávamos que o desempenho da criação de *sandboxes* com o módulo Venv fosse melhor que com o módulo Rings, porém os testes no nosso servidor web mostraram que o desempenho é praticamente igual. A princípio temos a impressão que o desempenho do modelo de submissão de *strings* de código para comunicação entre *sandboxes* teria um desempenho muito inferior aos outros modelos porém o resultado dos testes com transferência de 1Mbyte mostrou que a diferença de desempenho comparado com o modelo sem sandbox e com o módulo Venv é muito pequena. Por outro lado, em quase todas as situações o desempenho da comunicação do modelo Rings foi melhor que a comunicação por *sockets* do modelo de *sandbox* por processo.

Como cada modelo de concorrência e de *sandbox* apresenta algum ponto onde é mais forte, a escolha dos modelos deve ser feita de acordo com a aplicação que se deseja executar.

O modelo matemático do comportamento do sistema descreve aproximadamente os resultados obtidos, com exceção a modelagem do mecanismo de emulação de carga de processamento. O problema com a medição da carga de processamento ocorreu devido a decisão de embutir o código da aplicação dentro de cada manipulador de requisições para evitar uma nova leitura do disco.

Esperávamos um comportamento com tempo de processamento linearmente proporcional ao valor de entrada. Porém, observamos que o tempo de resposta de um código Lua pode variar dependendo da posição que as variáveis são alocadas na pilha da máquina virtual. Isso nos obrigou a fazer uma análise mais qualitativa que quantitativa a respeito do aumento do volume de processamento.

Em testes de desempenho futuros, onde a geração da carga seja feita em Lua, devemos nos preocupar com a consistência do *bytecode* gerado para os diversos casos de testes ou gerar essa carga, por exemplo, em uma função C ou até mesmo assembly, onde podemos ter maiores garantias de tempo de resposta.