Processando o Saber
eISSN 2179-5150
FATEC Praia Grande · fatecpg.edu.br/revista
submetido: Jan/2021 · **aceito**: Mar/2021 · **publicado**: Jun/2021

Comparativo de performance entre arquiteturas Ibm *z* e Ibm *power system*

Performance comparison between architectures Ibm z and Ibm power system

Cássio Félix Moura
Fatec Praia Grande
cassiosav@gmail.com

Hugo Leonardo Ariaz Amador
Fatec Praia Grande
hugoariaz@gmail.com

Simone Maria Viana Romano
Fatec Praia Grande
simone@fatecpg.com.br

Rodrigo Luiz Favorino Gaiotto
Fatec Praia Grande
rgaiotto@br.ibm.com

RESUMO

Neste artigo comparativo técnico são utilizados os recursos de duas importantes ferramentas da história da IBM, sendo eles: IBM Z e IBM *POWER SYSTEM*. Ambos os sistemas são dotados de tecnologia atual e utilizam como base de sistema operacional o Linux RedHat compilado de forma igual, dispostos de *hardware* equivalentes. Nessa abordagem, o objetivo principal é comparar a performance das arquiteturas utilizando a linguagem C baseando-se em um script elaborado para o teste, além de um teste com cargas de estresse comumente utilizado em ambientes *Linux*. Os resultados apresentam o tempo de resposta obtido após as execuções dentro das limitações escolhidas pelos autores e o recurso de *hardware* utilizado para obter os dados consolidados e a conclusão da melhor disposição de cada ambiente nos testes propostos nessa obra.

PALAVRAS-CHAVE: Performance. Linux. IBM *System*. Linguagem C. Processamento.

ABSTRACT

This comparative technical article, the resources of two important tools in the history of IBM are used, namely: IBM Z and IBM POWER SYSTEM. Both systems are equipped with current technology and use Linux RedHat as an operating system base, compiled in the same way, with equivalent hardware. In this approach, the main objective is to compare the performance of the architectures using the C language based on a script designed for the test, besides a test with loads of stress commonly used in Linux environments. The results show the response time obtained after executions within the limitations chosen by the authors and the hardware resource used to obtain the consolidated data and the conclusion of the best disposition of each environment in the tests proposed in this work.

KEYWORDS: *Performance. Linux. IBM System. Language C. Processing.*

INTRODUÇÃO

Neste artigo iremos explorar a arquitetura de dois importantes sistemas para processamento de dados da IBM que competem na busca de soluções para clientes envolvendo o que há de melhor em qualidade de *hardware* e *software*: o IBM Z e IBM *Power System*.

A escolha foi feita a partir de um estudo em relação a *hardware, software* e similaridade entre as arquiteturas propostas para que durante o teste de arquitetura não houvesse desigualdade na análise. É importante ressaltar que os testes executados não avaliam a infraestrutura como um todo e não englobam quesitos como virtualização, *network* e *storages*. O intuito deste estudo é trazer para a realidade as possibilidades de execução de testes em plataformas, por vezes mistificadas, impossíveis e complexas.

Nossa abordagem será em torno de dois testes de *benchmark*. O primeiro foi desenvlvido em linguagem C, utilizando lógica e analisando o tempo de processamento entre as duas arquiteturas. A IBM é conhecida mundialmente pelo *Deep Blue*, o sistema que derrotou o enxadrista Garry Kasparov em meados dos anos 90. Com base e inspiração nessa história, nosso script *Queens* fará um pequeno teste de matriz em um tabuleiro de xadrez.

O segundo teste é feito baseado em *STRESS-NG*, no qual um conjunto de instruções é executado em lotes pelo sistema. Ele permite estressar o sistema de forma que todo o conjunto de *hardware* disponível para o sistema operacional seja utilizado por completo.

Por fim, será apresentado o resultado dos testes e a conclusão sobre a comparação de performance de ambas arquiteturas que possibilitará ter uma visão sobre a estabilidade das infraestruturas.

1. *MAINFRAME*

O *Mainframe* por definição é: “um computador de alto desempenho e grande capacidade de processamento capaz de processar enormes quantidades de dados com extrema velocidade” (LAUDON, 2014). Seguindo a estrutura básica de Von Neuman, porém com um projeto voltado a maior desempenho no que tange às operações de leitura e escrita, acesso a banco de dados e entrada e saída, *Mainframe* pode ser definido como um servidor de alta capacidade, tendo como características adicionais segurança, confiabilidade e alta disponibilidade.

Levando em conta essas características, presume-se que, os primeiros *mainframes* da história eram máquinas estratégicas, destinadas, por exemplo, a calcular trajetórias de balística dos projéteis militares. Essa era uma das principais funções do ENIAC *(Electronic Numerical*

Integrator and Computer) e, ainda que não adotasse a estrutura de Von Neuman, em sua essência, era considerado um computador eletrônico.

Segundo uma descrição, a máquina tinha 24 metros de comprimento e 2,5 metros de largura e suas instruções eram programadas via cartão perfurado. Conforme Fonseca (2007, p.104) "ao todo possuía 18.000 válvulas. Executava desvios condicionais e era programável, o que o diferenciava das outras máquinas construídas até a data. Sua programação era feita manualmente, através de fios e chaves."

O ENIAC era conhecido como um computador de primeira geração, composto por relés e válvulas, consumia 140.000W de energia e pesava 30 toneladas. Essa máquina foi desenvolvida na Universidade da Pensilvânia (Filadelfia – EUA) e ficou pronta para uso em de 1946.

Segundo Tanenbaum (2013, p.13)

> A construção da máquina só foi concluída em 1946, quando era muito tarde para ser de alguma utilidade em relação a seu propósito original. Todavia, uma vez que a guerra tinha acabado Mauchley e Eckert receberam permissão para organizar um curso de verão para descrever seu trabalho para seus colegas cientistas, aquele curso de verão foi o início de uma explosão de interesse na construção de grandes computadores digitais.
> Após aquele curso de verão histórico, outros pesquisadores se dispuseram a construir computadores eletrônicos. O primeiro a entrar em operação foi o EDSAC (1949), construído na Universidade de Cambridge por Maurice Wilkes. Entre outros, figuravam o *JOHNIAC*, da *Rand Corporation*, o ILLIAC, da Universidade de Illinois, o MANIAC, do *Los Alamos Laboratory*; e o WEIZAC, do *Weizmann Institute*, em Israel.

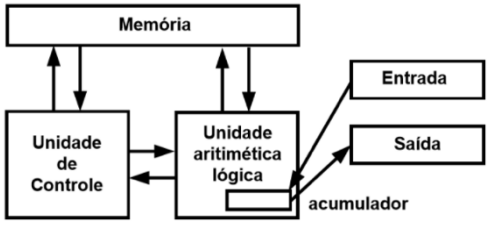
Os dois pesquisadores que encabeçaram o grupo trabalharam no ENIAC e em seguida iniciaram o projeto do EDVAC (*Electronic Discrete Variable Automatic Computer*). Em 1944, John Von Neumann começou a trabalhar nesta universidade como consultor e participou do projeto ENIAC. John Mauchley e J. Presper Eckert saíram da equipe da universidade para montar sua própria empresa, a Eckert- Mauchley, que após muitos anos, depois de sofrer seguidas fusões e aquisições, se tornou a UNISYS. Infelizmente esta saída atrapalhou o andamento do projeto inicial do EDVAC, mas John Von Neumann que havia ido para o *Institute of Advanced Studies de Princeton* iniciou sua própria versão do EDVAC, o IAS.

"Von Neumann era um gênio, da mesma estirpe de Leonardo Da Vinci. Falava muitos idiomas, era especialista em ciências físicas e matemática ... na época que se interessou por computadores já era o mais eminente matemático do mundo" TANENBAUM (2013, p. 14).

Esta era a descrição do homem que deu colaboração essencial na construção dos computadores. Talvez, a prova dessa genialidade seja que, depois de tantos anos e avanços, a

arquitetura básica de construção de muitos computadores continua sendo baseada nas ideias deste cientista.

Figura 1- Arquitetura de John Von Neumann



Fonte: Adaptado de TANENBAUM, 2013.

Na sequência de evoluções, o UNIVAC foi um dos principais computadores de segunda geração, cujas válvulas eletrônicas efetuavam operações permitindo aumento de velocidade. Em 1951, o conceito de memória para armazenar dados construído pela *Remington-Rand*, tornou-se o primeiro computador comercialmente disponível.

Em 1964 a IBM lançava o *SYSTEM /360*, um *mainframe* de terceira geração, o primeiro disponível comercialmente. Esse projeto teve um custo de US$ 5 bilhões, o dobro do faturamento anual da empresa. O artefato era baseado em circuitos integrados, semicondutores que continham vários transistores, com uma considerável capacidade computacional para a época.

Esse desafio tanto técnico como comercial gerou frutos para a empresa, que faturou US$ 3,6 bilhões em 1965 e subiu para US$ 8,3 bilhões em 1971. Na década de 70, mais de 70% dos *mainframes* vendidos no mundo eram IBM.

A empresa se tornava, então, referência neste tipo de computador, embora existissem outros fabricantes como General Electric, Unisys (na época Burroughs), Honeywell entre outros. A predominância da IBM permanece intacta até hoje.

Tanenbaum (2013, p.30) discorre sobre *mainframes* e sua estrutura e capacidade: “na maioria não são muito mais rápidas do que servidores de grande potência, mas sempre têm mais capacidade de E/S e costumam ser equipadas com vastas coleções de discos que contém normalmente milhares de gigabytes de dados”.

A partir deste ponto, é importante pontuar que qualquer assunto sobre *mainframes* deve obrigatoriamente incluir a IBM, não só como único material devido sua singularidade, mas também como o mais importante material de estudo sobre as tecnologias advindas da empresa.

Os computadores System / 360 executaram vários feitos, desde o sistema automatizado de emissão de bilhetes aéreos até o sistema de apoio computacional que possibilitou a missão

da APOLO 11 para explorar a lua. Esses supercomputadores eram projetados para atender tanto uso científico como comercial.

2. IBM Z

Para entender melhor a origem do IBM Z, é preciso remontar a história a partir do *System/360* de 1964, quando tiveram origem as primeiras estruturas de *mainframe* como são conhecidas hoje. Esses sistemas, já descritos anteriormente, formam a base de operação do *mainframe* da família IBM Z, cujos *hardwares* ainda são compatíveis com as operações de décadas atrás. Em 2005, a IBM introduziu no mercado o IBM *Enterprise* z9, utilizando a nomenclatura "SYSTEM Z" para designar os *mainframes*, tomando Z como proveniente de *zero downtime*, sendo essa uma das principais características do *mainframe*. Em 2017, a IBM renomeou toda a família de produtos do IBM System Z para apenas IBM Z, dando uma nova identidade ao *mainframe*.

Sua arquitetura é projetada para altas demandas de processamento de dados, sejam elas para operações bancárias, transações de dados e até a atual plataforma do *IBM Cloud.* A tolerância a falhas e sistemas compostos de hardwares redundantes garantem o total funcionamento da arquitetura todos os dias do ano.

Para que o funcionamento do *mainframe* seja satisfatório para seus clientes, percebe-se que há uma combinação robusta entre *hardware* e *software* e a cada versão lançada, os conjuntos são atualizados para atender o mercado complexo que consome esse produto. Em 2019 foi lançada a z15, produto da família IBM Z que possui o melhor *hardware* já apresentado pela empresa. Este supercomputador opera em seu interior com processadores, memória, placas de redes e placas criptográficas de última geração, visando a virtualização de ambientes e entrega de infraestrutura *Cloud Computing* em alta performance.

Como diferenciais desse produto para as demais arquiteturas de mercado, é possível pontuar a flexibilidade do IBM Z para *Clouds* públicas, privadas e híbridas, a resiliência cibernética devido ao isolamento e recuperação instantânea em casos de ataques, a encriptação de alta tecnologia e a combinação da arquitetura Z com as ferramentas da *Red Hat* para soluções em Linux.

Com relação aos sistemas operacionais, o IBM Z possui 5 importantes variações: z/OS, Linux, z/VSE, z/TPF e z/VM. Cada uma dessas variações é destinada a um tipo de necessidade de negócio e visa a utilização da arquitetura por completo, cada qual com sua especificação, atributos, requisitos, soluções e compatibilidade destinadas à operação. Vale ressaltar que o Linux para *mainframe* trabalha sobre a camada de virtualização do z/VM, e não naturalmente acima do *hardware* físico, sendo assim, ele opera dentro de um sistema complexo e robusto de virtualização de ambiente, originário do ano 2000. (LASCU, 2020).

Para entender basicamente como o *mainframe* funciona, devemos analisar a capacidade de virtualização dessa arquitetura, tendo em conta que esse é o *core* da linha IBM Z. O *mainframe* da linha Z é dividido por camadas, sendo a primeira camada chamada de PR/SM *(Processor Resource / System Manager)*, na qual são implantados blocos de unidades lógicas *(LPAR – Logical Partition)* para a instalação dos sistemas operacionais. Essa primeira camada permite a execução de um número limitado de LPARs, definido na instrução de cada *mainframe* Z. Na versão z15, por exemplo, pode ser configurado até 85 LPARs que incluem conjuntos de memória e processadores para cada, conforme descrito em seu guia técnico. (LASCU, 2020).

Após a camada PR/SM, existe a camada do sistema operacional que pode ser um sistema ativo para operação ou um *hypervisor* para virtualizar outros sistemas operacionais. Um exemplo de *hypervisor* é o z/VM, sistema operacional que gerencia os recursos de *hardware*, rede e armazenamento para sistemas operacionais acima de sua camada, provendo a estrutura necessária para executar várias instancias de Linux em uma única LPAR. Nesse ambiente é possível configurar inúmeros servidores Linux, compartilhando dados, redundantes e de alta disponibilidade devido ao modelo de arquitetura tanto do *hardware* como do *software* em questão. (CORDERO, 2013).

3. IBM *POWER SYSTEM*

A linha P da IBM, são *mainframes* com processadores e arquitetura *POWER* (*Performance Optimization With Enhanced Risc)*, em tradução livre: “Otimização de Performance com tecnologia RISC melhorada”, que por RISC *(Reduced Instruction Set Computer*) entende-se: “Computador com número reduzido de instruções”.

O principal objetivo desses processadores, advindos dos utilizados na década de 80, nos *mainframes* AS400 e RS6000, era permitir uma forte integração entre banco de dados e aplicativos rodando em um mesmo servidor. Graças ao design do processador, mesmo em

baixas velocidades de *clock*, o desempenho é mantido, pois a cada ciclo várias instruções são executadas.

Os servidores que utilizam estes processadores são mais eficientes em operações que trabalham com uma grande quantidade de dados. Devido à arquitetura dos processadores, com oito *threads* por núcleo, este equipamento se mostra adequado para aplicações de inteligência artificial.

Através da tecnologia de virtualização da IBM, chamada *PowerVM*, é possível particionar o servidor. O usuário pode adicionar e, também, remover recursos desta partição proporcionando flexibilidade para os usuários. (VETTER, 2020).

4. LINUX *RED HAT ENTERPRISE*

O *Red Hat Linux Enterprise* é uma distribuição *open source* Linux desenvolvida pela empresa *Red Hat* que teve sua primeira versão lançada em 2003. Desde então, a *Red Hat* procura aperfeiçoar seus sistemas operacionais para atender as demandas de mercado e as necessidades de seus clientes, cumprindo os requisitos de segurança, confiabilidade, performance, estabilidade e escalabilidade.

Sua estrutura permite a instalação e configuração em diversas arquiteturas distintas, como: x86-64, *Power*, ARM64, IBM Z e *desktop*. Após uma parceria de mais de 17 anos, a *Red Hat* foi adquirida pela IBM no ano de 2018, e assim o *Red Hat Linux Enterprise* se tornou o sistema operacional oficial da empresa para *mainframes*. (QUINTERO, 2020).

5. *BENCHMARK*

Apresentados os dois sistemas, a proposta é compará-los e verificar, sob determinadas condições, o desempenho de cada um, facilitando a escolha do usuário. Ambos os sistemas pertencem ao mesmo fabricante, porém possuem arquitetura e estrutura diferentes. Para fazermos uma comparação, além da apresentação de cada sistema, se faz necessário escolher um método comparativo.

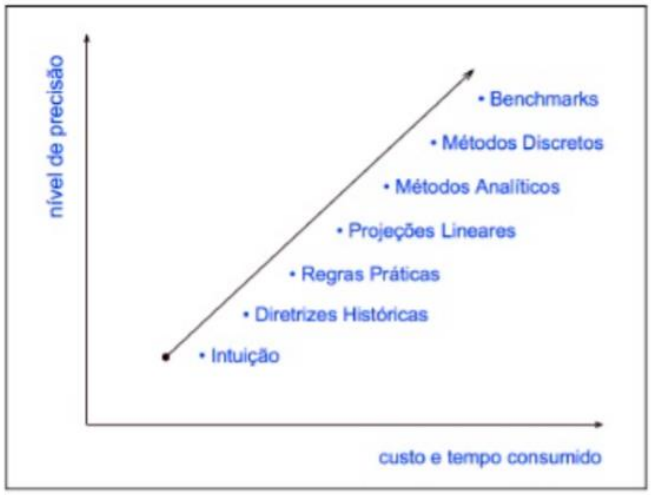
A definição de *Benchmark* em sistemas computacionais é justamente a medição através de programas de parâmetros que atestam o desempenho de determinado dispositivo e este

marcador, assim como um termômetro, permite uma referência que após medição em diferentes equipamentos nos dá um valor de comparação.

Essa comparação quantitativa começa com a definição de um *benchmark* ou carga de trabalho. O *benchmark* é executado em vários sistemas diferentes e o desempenho e o preço de cada sistema é medido e registrado. O desempenho é normalmente uma métrica de rendimento (trabalho / segundo) (GRAY,1992).

O *benchmark* é um método amplamente utilizado na área da tecnologia computacional. Para cada tipo de demanda existe uma personalização adequada, logo o desempenho de um sistema voltado a jogos com interface gráfica é medido de maneira diferente de um sistema voltado para o cálculo computacional. Existem vários métodos comparativos, cada um com seu custo, precisão e eficácia. A figura a seguir apresenta a ideia entre nível de precisão e o contraponto de custo e tempo consumido.

Figura 2 – Os diferentes métodos de comparação



Fonte: (Cesar & Dias, 2007)

5.1 TESTE DO TABULEIRO PARA VERIFICAÇÃO DE PERFORMANCE

Na história da programação além do famoso "*Hello World*", temos como base o ensino de matrizes para qualquer linguagem. Essas matrizes combinam os dados de forma concatenada e, como é possível a validade de sua integridade em testes de mesa, garantem quando combinadas a testes lógicos uma performance e precisão perfeita para avaliações e análise de dados.

Para executar a comparação de processamento entre as arquiteturas IBM, optou-se por desenvolver um *script* lógico para avaliar o tempo de execução em uma matriz de determinado tamanho. A premissa parte de um tabuleiro de xadrez com a matriz de 8 por 8 como nas medidas

oficiais, gerando 64 posições possíveis, onde 4 rainhas devem estar alinhadas sem ameaçar a si mesmas. Ressaltando que pode ser atribuído qualquer valor a matriz, porém aqui, será seguido a quantidade de casas corretas.

A linguagem C tem excelente aceitação nos sistemas Linux/Unix. Para executar o *script*, basta ter o pacote GCC instalado no sistema operacional e fazer a compilação/execução do *script* criado. Para avaliar o tempo de resposta, deve ser adicionado a *tag "time"* antes do nome do teste a ser executado, pois isso irá retornar o tempo de execução de toda a tarefa codificada no *script* em C.

5.2 TESTE DE PERFORMANCE UTILIZANDO CARGA DE ESTRESSE

Este teste é realizado por várias empresas para avaliar a performance de seus sistemas operacionais e *hardware* em pico de altas cargas de trabalho. Eles analisam o conjunto combinado do sistema e englobam processamento, memória e I/O por um determinado período e carga e são primordiais para averiguar como o sistema se portará em casos extremos.

Para o teste executado nos sistemas IBM Z e IBM *Power System*, foi eleito o *Pulse Loader* que é um *script* desenvolvido para gerar taxas de utilização e sobrecargas de estresse dos sistemas Linux e Unix. Seu funcionamento é baseado no conhecido *STRESS-NG*, portanto depende do pacote *stress-ng* para sua execução no ambiente cujo o teste consiste em 3 estágios aplicando-se cargas de 25% mais pesadas a cada 24 execuções.

Para execução do *script*, foram utilizados os seguintes parâmetros combinados ao comando abaixo:

```
/stress-ng --cpu X --io 1 --vm 1 --vm-bytes 100% --timeout 120s --
  metrics-brief
```

Traduzindo cada parâmetro do *stress-ng*, temos o seguinte:

- CPU X = fará uso de X processadores durante a execução
- IO 1= Parâmetro para gerar 1 lote de instruções de E / S
- vm 1 --vmbytes 100% = será gerado uma instância virtual que irá utilizar 100% da memória disponível.
- timeout 120s = Será executado durante 120 segundos.
- metrics-brief = Produzirá um relatório após a execução do script.

6. CÓDIGO EM C

```
/* C/C++ programa para resolver N Queen Problem usando
backtracking */
//Aqui podemos mudar e definir o tamanho do tabuleiro
#define N 16
#include <stdbool.h>
#include <stdio.h>
/* A função para imprimir */
void printSolution(int board[N][N])
{
    int i;
    for (i = 0; i < N; i++) {
          int j;
          for (j = 0; j < N; j++)
                printf(" %d ", board[i][j]);
          printf("\n");
    }
}
/* A função de utilidade para verificar se uma rainha pode ser colocada a
bordo [linha] [col]. Observe que esta função é chamada quando as rainhas
"col" já estão colocadas nas colunas de 0 a col -1. Portanto, precisamos
verificar apenas o lado esquerdo para rainhas de ataque */

bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    /* Verifique esta linha do lado esquerdo */
    for (i = 0; i < col; i++)
          if (board[row][i])
                return false;

    /* Verifique a diagonal superior do lado esquerdo */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
          if (board[i][j])
                return false;

    /* Verifique a diagonal inferior do lado esquerdo */
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
          if (board[i][j])
                return false;
    return true;
}

/* Uma função de utilidade recursiva para resolver o problema de N Queen
*/

bool solveNQUtil(int board[N][N], int col)
{
    /* caso básico: se todas as rainhas forem colocadas
então retorne verdadeiro */

    if (col >= N)
          return true;

    /* Considere esta coluna e tente colocar
    esta rainha em todas as filas uma a uma */
```

```
    int i;
    for (i = 0; i < N; i++) {
          /* Verifique se a rainha pode ser colocada a bordo [i][col] */

          if (isSafe(board, i, col)) {
                /* Coloque esta rainha no tabuleiro [i][col] */
                board[i][col] = 1;

                /* Recorra para colocar o resto das rainhas */
                if (solveNQUtil(board, col + 1))
                      return true;

                /* Se colocar a rainha no tabuleiro[i][col]
                não leva a uma solução, então
                remova a rainha[i][col] */
                board[i][col] = 0; // BACKTRACK
          }
    }

    /* Se a rainha não puder ser colocada em qualquer linha
    essa coluna col retorna false */
    return false;
 }
/* Esta função resolve o problema N Queen usando Backtracking. Ele usa
principalmente solveNQUtil () para resolva o problema. Retorna falso se as
rainhas não puderem ser movidas, caso contrário, retorna verdadeiro e
imprime a colocação de rainhas na forma de 1s. Observe que pode haver mais
de uma solução, esta função imprime uma das soluções viáveis. */

 bool solveNQ()
 {
    int board[N][N] = { { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 } };

    if (solveNQUtil(board, 0) == false) {
          printf("Solution does not exist");
          return false;
    }

    printSolution(board);
    return true;
 }

 // teste de funcao
 int main()
 {
    solveNQ();
    return 0;
 }
```

7. SOBRE OS AMBIENTES

Quando a comparação de arquiteturas é necessária, leva-se em consideração a similaridade de ambientes, sendo isto importante para que o teste seja imparcial e analise o

conjunto como um todo. Normalmente as arquiteturas são distintas em sua criação, codificação de baixo nível e demais componentes de *hardware*, então temos que buscar a proximidade em quantidades e nivelar pelo mais baixo nível de diferença. Seguimos as configurações dessa forma nos testes executados.

Sobre o *software*, foram utilizadas distribuições *Linux Red Hat Enterprise* em sua última versão disponível em mercado para cada arquitetura (GHORI, 2020). A diferença na compilação do *Kernel* é mínima e, por avaliação, não haveria uma intervenção significativa em valores. Ambas as máquinas foram testadas somente com o sistema operacional nativo e sem a instalação ou performance de outros aplicativos em paralelo.

Com relação ao processamento e memória, foram atribuídos em ambas as máquinas: 2 Processadores e 8GB de memória RAM, configuradas por padrão no sistema e sem intervenções. Cada qual possui sua particularidade em relação a *caches*, *threads* e frequências, conforme evidencia retirada de cada arquitetura e descrita nos itens deste artigo.

7.1 DETALHES DE *HARDWARE* IBM Z

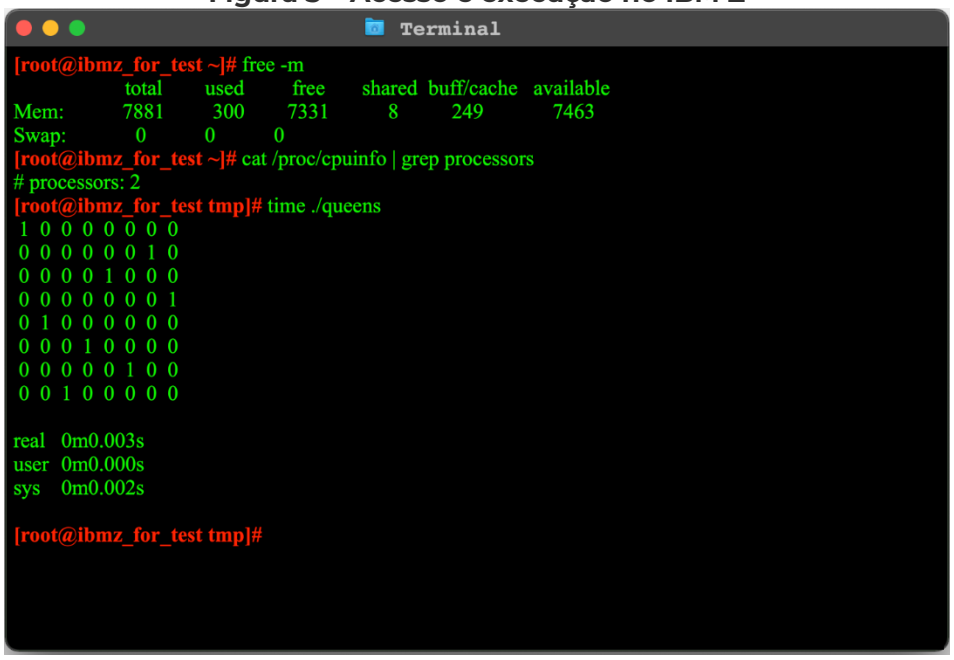
Hardware: Type 3906 - Model: 716 (z14)
Kernel: Linux ibmz_for_test.ibm.com 4.18.0-193.19.1.el8_2.s390x #1 SMP Wed Aug 26 15:15:48 EDT 2020 s390x s390x s390x GNU/Linux
Processor Type:
vendor_id : IBM/S390
processors : 1
bogomips per cpu: 21881.00
max thread id : 0
cache0 : level=1 type=Data scope=Private size=128K line_size=256 associativity=8
cache1 : level=1 type=Instruction scope=Private size=128K line_size=256 associativity=8
cache2 : level=2 type=Data scope=Private size=4096K line_size=256 associativity=8
cache3 : level=2 type=Instruction scope=Private size=2048K line_size=256 associativity=8
cache4 : level=3 type=Unified scope=Shared size=131072K line_size=256 associativity=32
cache5 : level=4 type=Unified scope=Shared size=688128K line_size=256 associativity=42
processor 0: version = FF, identification = 0133A7, machine = 3906
cpu number : 0
cpu MHz dynamic : 5208
cpu MHz static : 5208

7.2 EXECUÇÃO NO IBM Z

Passo a passo da execução do teste:

- Acesso via terminal por conexão SSH;
- Verificação de memória disponível:
- Verificação da quantidade de processadores;
- Acesso ao diretório / tmp onde está locado o teste;
- Execução do teste junto com a variável time;
- Obtenção do resultado.

Figura 3 – Acesso e execução no IBM Z



Fonte: Elaborado pelos autores para este trabalho.

7.3 DETALHES DE *HARDWARE* IBM *POWER SYSTEM* (*POWER* 9)

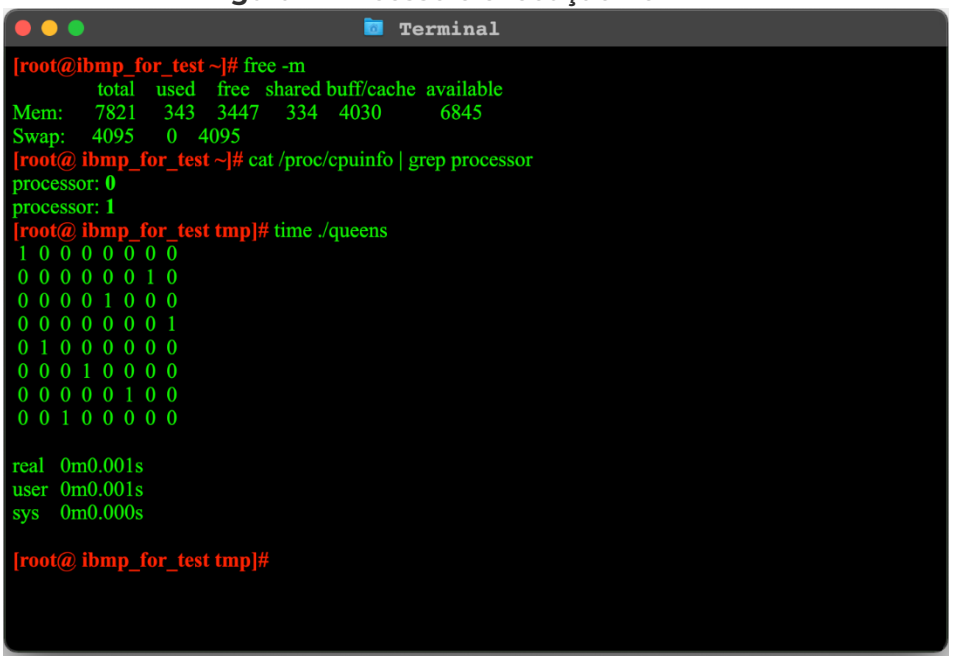
Hardware: Type 9080 - Model: M9S
Kernel: Linux ibmp_for_test.ibm.com 3.10.0-1127.10.1.el7.ppc64le #1 SMP Tue May 26 15:11:07 EDT 2020 ppc64le ppc64le ppc64le GNU/Linux
Processor Type:
description: POWER8 (architected), altivec supported
product: PowerPC, POWER9
physical id: 8
version: 1.3 (pvr 004e 2103)
size: 3150MHz
clock: 1600MHz
capabilities: performance-monitor
configuration: threads=8

7.4 EXECUÇÃO NO IBM *POWER SYSTEM* (*POWER* 9)

Passo a passo da execução do teste de tabuleiro:

- Acesso via terminal por conexão SSH;
- Verificação de memória disponível;
- Verificação da quantidade de processadores;
- Acesso ao diretório /tmp onde está locado o teste;
- Execução do teste junto com a variável time;
- Obtenção do resultado.

Figura 4 – Acesso e execução no IBM P

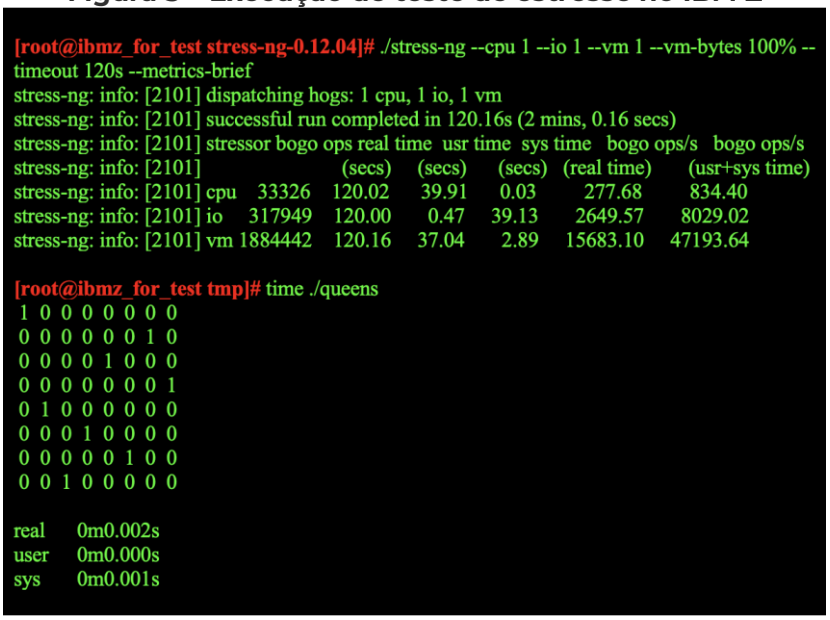


Fonte: Elaborado pelos autores para este trabalho.

8. CENÁRIO ADVERSO

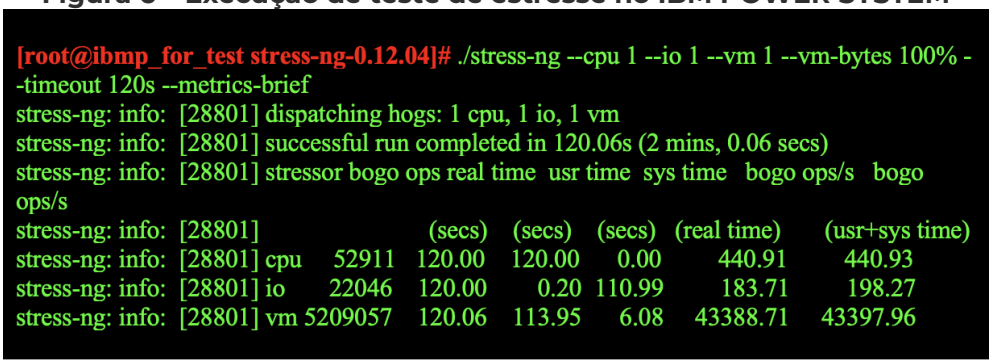
A execução do teste de estresse em ambas as arquiteturas foi utilizando apenas 1 processador e 8 gigabytes de memória. Como referenciado no item 5.2 deste artigo, foram utilizados os mesmos parâmetros do comando descrito anteriormente.

Figura 5 – Execução de teste de estresse no IBM Z



Fonte: Elaborado pelos autores para este trabalho.

Figura 6 – Execução de teste de estresse no IBM POWER SYSTEM



Fonte: Elaborado pelos autores para este trabalho.

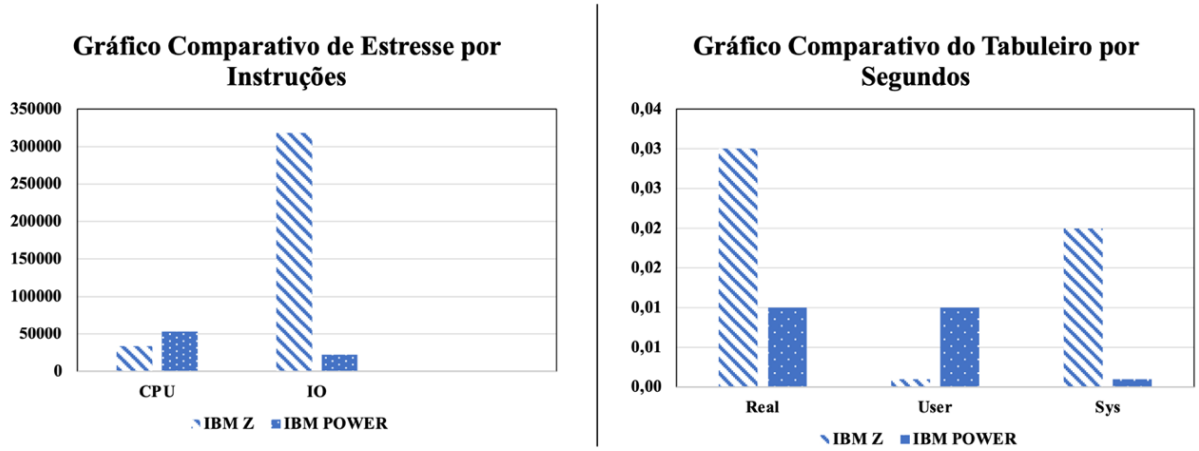
9. ANÁLISE DOS RESULTADOS OBTIDOS

De acordo como os resultados dos testes de tabuleiro, apresentados no gráfico a seguir, nota-se uma diferença no tempo real de processamento de 1 milissegundo entre as arquiteturas dando melhor colocação ao IBM *POWER SYSTEM*. As evidências nos itens 7.2 e 7.4, trazem também os itens *users* e *sys*, nos quais não há grande distinção nos resultados obtidos e garante que o IBM *POWER SYSTEM* possui uma melhor performance dentro das condições determinadas neste estudo.

Quando executado o teste de estresse em ambas as arquiteturas, concluiu-se que dentro do cenário adverso (item 8), a performance do IBM Z se consolida melhor por processar uma quantidade maior de I/O no mesmo tempo proposto. Esta diferença se deve ao número de instruções de baixo nível enviadas ao processador de ambas as plataformas para execução de

atividades. Denota-se, baseado neste estudo e cenário, que o processamento do *mainframe* se torna mais rápido quando se tem mais instruções.

Gráfico 1 - Comparativo de Performance em ambas as arquiteturas.



Fonte: Elaborado pelos autores após obtenção de resultados.

10. CONSIDERAÇÕES FINAIS

Novas oportunidades de hospedagem de aplicações distribuídas passaram a ser possíveis nestes equipamentos de alta performance desde a portabilidade do Linux para a arquitetura do *mainframe*. Tal evolução abre para o profissional de tecnologia da informação diferentes opções de ambientes e arquiteturas para processamento de cargas segundo as necessidades para o desenvolvimento de soluções utilizando código aberto.

Este artigo abordou duas destas opções que tratam de *hardwares* distintos, mas que oferecem funcionalidades semelhantes a nível de aplicação sob a camada do sistema operacional. Naturalmente, a escolha pelo ambiente adequado para o desenvolvimento de determinada solução torna-se complexa, então testes de performance podem ser utilizados como uma importante base de dados para apoio desta tomada de decisão.

Os testes de performance apresentados neste artigo, realizados através de estudos em ambientes controlados e determinados pelos autores, indicam que através de evidencias técnicas podemos concluir que: após executado o teste de tabuleiro, o *IBM POWER SYSTEM* possui melhor performance operando em baixas quantidades de instruções ao processador. Já o IBM Z processa 10 vezes mais dados de entradas e saídas do que seu sistema concorrente dentro dos parâmetros do ambiente controlado e proposto no teste de estresse. Isto significa que o IBM Z possui um diferencial quanto ao processamento em volume de dados, sendo mais bem aproveitado com os mesmos recursos equiparados de hardware do *IBM POWER SYSTEM*.

Dado o exposto, é importante considerar que este artigo está baseado somente nas limitações descritas nos ambientes propostos e não pode ser conclusivo quanto ao uso de determinada tecnologia para demais fins. Sendo assim, outros testes de *benchmark* podem definir se o IBM Z ou o *IBM POWER SYSTEM* se adequam a outras necessidades de negócio.

REFERÊNCIAS

CESAR, A., & DIAS, S. **A eficiência de mainframes medida através de conceitos de linha de produção e da Lei de Amdahl**. (p. 1–12), 2007.

CORDERO, Mel. Et al. ***IBM PowerVM Virtualization Introduction and Configuration***. Poughkeepsie, USA: IBM Redbooks, 2013.

FONSECA FILHO, C. **História da Computação:** O caminho do pensamento e da tecnologia. EDIPUCRS, 2007.

GHORI, Asghar. ***RHCSA Red Hat Enterprise Linux 8, Second Edition***. La Vergne, USA: Endeavor Technologies Inc., 2020.

GRAY, Jim. ***Benchmark Handbook:*** *For Database and Transaction Processing Systems.* San Francisco, USA: Morgan Kaufmann Publishers Inc., 1992.

LASCU, Octavian; Et al. ***IBM z15 (8561) Technical Guide***. Poughkeepsie, USA: IBM Redbooks, 2020.

LAUDON, Kenneth C., & LAUDON, Jane P. **Sistemas de Informações Gerenciais,** 11ª edição**.** São Paulo: Pearson, 2014.

QUINTERO, D; Et al. ***Front cover Red Hat OpenShift and IBM Cloud Paks on IBM Power Systems Volume 1. 1***. Poughkeepsie, USA: IBM Redbooks, 2020.

TANENBAUM, Andrew S; AUSTIN, Todd. **Organização Estruturada de Computadores, Sexta Edição**. São Paulo: Pearson, 2013.

VETTER, Scott; Et al. ***IBM Power Systems S922, S914, and S924 Technical Overview and Introduction***. Poughkeepsie, USA: IBM Redbooks, 2020.