VIII Simpósio de Iniciação Científica 2025

"Do conhecimento acadêmico à transformação sustentável: inovação com validação científica"

REAPROVEITAMENTO DE TV BOXES EM CLUSTERS COMPUTACIONAIS

Ana Laura Gonçalves Braga[1] (IC), Bruno Tardiole Kuehne (PQ)[1]
[1]*Universidade Federal de Itajubá.*

Palavras-chave: Computação Distribuída, Cluster de Baixo Custo, Balanceamento de Carga

Introdução

A presente pesquisa aborda a construção de um *cluster* de baixo custo utilizando um conjunto de *TV Boxes* reaproveitadas como pequenos computadores, aplicando conceitos de sistemas distribuídos, como balanceamento de carga e monitoramento em tempo real, para fins de análise de desempenho.

O objetivo é demonstrar a viabilidade da computação distribuída em cenários com recursos limitados, avaliando como o balanceamento de conexões impacta no uso da *CPU* (*Central Processing Unit*) e da memória *RAM* (*Random Access Memory*).

A proposta justifica-se por mostrar que soluções acessíveis e sustentáveis podem oferecer infraestrutura computacional eficiente em ambientes educacionais, projetos independentes ou laboratórios de testes.

A arquitetura do sistema foi organizada a partir de uma *TV Box* que atua como nó principal, responsável por hospedar o servidor *web (World Wide Web - WWW)* e gerenciar o fluxo de acessos. Esse nó concentra o serviço de *reverse proxy* e o *load balancing* por meio do *Nginx*, enquanto também disponibiliza o *Front-end* que serve como ponto de entrada para os usuários. A partir desse *Front-end*, as conexões são distribuídas entre as demais *TV Boxes*, seguindo a lógica do balanceamento configurado.

Além disso, tanto a *TV Box* principal quanto as secundárias executam serviços de *Back-end* desenvolvidos em *Node.js*, responsáveis por expor uma *API (Application Programming Interface) REST (Representational State Transfer)* e conexões *WebSocket*. Esses serviços permitem não apenas a comunicação contínua com o *Front-end*, mas também a coleta periódica de métricas de *hardware*, como uso de *CPU*, memória e tempo de atividade. Dessa forma, cada unidade do *cluster* contribui para o monitoramento em tempo real, possibilitando a análise do desempenho global do sistema.

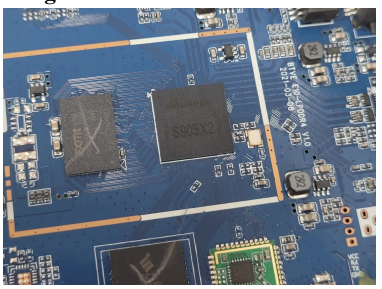
Metodologia

1. Instalação do Linux nas TV Boxes

Inicialmente, foi identificada a arquitetura do processador das *TV Boxes*, modelo *Amlogic S905X2*, para garantir compatibilidade com a distribuição *Linux*. Optou-se pelo *Armbian Bookworm Minimal*, que oferece leveza e estabilidade. A imagem do sistema foi gravada em um cartão *SD (Secure Digital)*, e para possibilitar o *boot* correto foi necessário ajustar o arquivo de configuração /boot/extlinux/extlinux.conf, adicionando a referência ao *DTB (Device Tree Blob)* apropriado para o modelo *(meson-g12a)*.

Outro passo fundamental foi a configuração do *u-boot*, responsável pelo processo de inicialização. Entre os arquivos disponíveis na pasta /boot, selecionou-se o u-boot-s905x2-s922, que foi renomeado para u-boot.ext. Em seguida, o dispositivo foi inicializado a partir do cartão *SD*, utilizando o botão de *reset* para forçar o *boot* externo.

Figura 1 – Processador da *TV Box*



Fonte – Autoral

Com o sistema em funcionamento, foram executadas atualizações básicas para assegurar a integridade dos pacotes. Para replicar o ambiente nas demais unidades, a configuração finalizada foi clonada para outros cartões *SD* por meio do comando *dd (Data Duplicator)*, permitindo que todas as *TV Boxes* inicializem já com o mesmo ambiente preparado.

2. Configuração de Dispositivos na Rede

Para a interconexão das unidades, foram confeccionados 20 cabos *Ethernet Cat5* crimpados manualmente, cada um com o comprimento adequado para alcançar o *switch* de 32 portas.

A alimentação elétrica foi centralizada por meio de uma fonte de 5V/40A, conectada em paralelo de modo que cada *TV Box* recebesse aproximadamente 2A, valor suficiente para seu funcionamento estável. Os conectores originais das fontes individuais foram reaproveitados, sendo soldados e adaptados para a fonte maior, o que eliminou a necessidade de múltiplos adaptadores externos.

No lado da rede, o *switch* foi integrado ao roteador principal, onde foram atribuídos endereços *IP (Internet Protocol)* estáticos para cada dispositivo. Essa configuração foi essencial para que cada *TV Box* pudesse ser identificada individualmente dentro do *cluster*, permitindo a gestão organizada de nós e facilitando a implementação do *load balancing.*

Figura 2 – Organização das 20 *TV Boxes*

Fonte – Autoral

3. Desenvolvimento e Aplicação

O sistema desenvolvido foi dividido em duas camadas principais: *Back-End (API + WebSocket)* e *Front-End* (interface de monitoramento em *React*). Essa arquitetura possibilita tanto o coletor de métricas locais em cada *TV Box* quanto a visualização centralizada do desempenho em tempo real.

3.1. Back-End

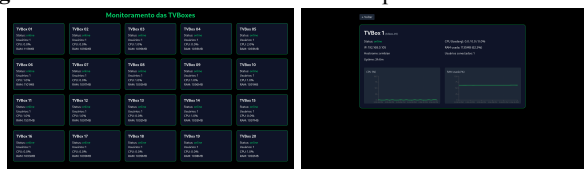
Cada *TV Box* executa uma *API* desenvolvida em *Node.js* com *Express*, responsável por disponibilizar informações sobre o *hardware*, como uso de memória, tempo de atividade, *IP* e carga da *CPU*. Essas informações são expostas por meio da rota *REST* /api/status, acessível a partir da rede local.

Além disso, foi implementado um servidor *WebSocket* utilizando a biblioteca *socket.io*. Esse módulo envia periodicamente atualizações em tempo real sobre os recursos monitorados, como *CPU* instantânea, memória utilizada e número de usuários conectados. Essa abordagem reduz a latência na coleta de métricas, já que não depende de múltiplas requisições *HTTP (Hypertext Transfer Protocol)*.

3.2. Front-End

O *Front-End* foi construído em *React*, onde um painel central exibe todas as *TV Boxes* conectadas. Cada dispositivo é representado por um *card*, que mostra o *status (online/offline), IP, uso de CPU e RAM*, e número de usuários conectados. Para maior clareza visual, foram aplicados estilos com *Tailwind CSS*, além de gráficos de séries temporais para *CPU* e memória, atualizados em tempo real.

Figura 3 – *Dashboard* com dados em tempo real das 20 *TV Boxes*



Fonte – Autoral

A *TV Box* principal abriga tanto o Front-End quanto um servidor *Nginx*. O *Front-End* é a interface pela qual o usuário acessa o sistema, funcionando como o ponto de entrada das conexões. A partir desse acesso, o *Nginx* atua como *reverse proxy* e executa o *load balancing*, distribuindo as conexões entre as demais *TV Boxes* do *cluster* por meio da estratégia *round robin.*

3.3. Integração

A integração do sistema ocorre em dois níveis distintos: administrativo e de usuários finais.

No nível administrativo, o *Front-End* mantém conexões *WebSocket* abertas com todos os serviços de *Back-End* das *TV Boxes*. Essas conexões são estabelecidas com a função de monitoramento e identificadas como de *"admin"*, não sendo contabilizadas como acessos de usuários. Através delas, o *dashboard* central recebe continuamente as métricas de cada unidade (uso de *CPU, RAM, uptime* e quantidade de conexões), permitindo uma atualização em tempo real da interface sem necessidade de múltiplas requisições *REST*.

No nível de usuários finais, a conexão é iniciada apenas quando alguém acessa o *Front-End* principal. Nesse momento, o *Nginx*, configurado como *reverse proxy* com *load balancing* round robin, direciona a sessão do usuário para uma *TV Box* específica. A partir desse redirecionamento, o usuário passa a manter uma sessão exclusiva com a unidade designada, através de um canal *WebSocket* dedicado, que é contabilizado no total de usuários conectados daquela *TV Box*.

Essa arquitetura garante que a camada de monitoramento *(admin)* opere de forma independente da camada de uso efetivo (usuários finais). Dessa forma, assegura-se a confiabilidade dos dados de desempenho exibidos no *dashboard*, ao mesmo tempo em que o *load balancing* distribui as conexões de usuários de maneira equitativa entre as *TV Boxes* do *cluster*.

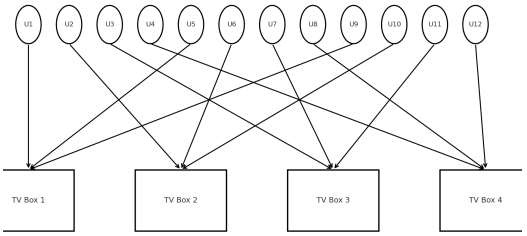
4. Configuração do Nginx

Para que o *cluster* pudesse operar de forma integrada, foi configurado o servidor *Nginx* na *TV Box* principal. Ele exerce duas funções centrais: *reverse proxy e load balancing*.

Como *reverse proxy*, o *Nginx* centraliza todas as conexões de usuários no endereço da *TV Box* principal, ocultando os *IPs* reais das demais unidades. Isso simplifica o acesso, já que os usuários interagem apenas com um ponto de entrada, enquanto o *Nginx* se encarrega de redirecionar o tráfego internamente. Esse processo também adiciona uma camada de segurança e isolamento, protegendo a rede do *cluster* contra acessos diretos.

Para o *load balancing*, foi configurado o método *round robin*, no qual cada nova conexão é encaminhada sequencialmente para uma *TV Box* diferente. Assim, se o primeiro usuário for direcionado à *TV Box* 1, o próximo será atendido pela *TV Box* 2, e assim por diante, retornando ao início da lista ao completar o ciclo. Esse mecanismo distribui as requisições de maneira uniforme, garantindo que nenhuma unidade seja sobrecarregada e que o processamento seja dividido entre todos os nós disponíveis.

Figura 4 – Diagrama ilustrativo do método *round-robin*

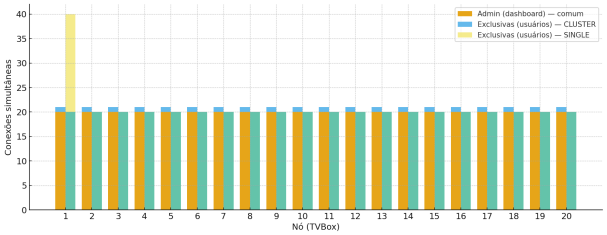


Fonte – Autoral

Resultados e discussão

Para evidenciar o impacto do balanceamento, os resultados serão apresentados em dois cenários: (i) com balanceamento (*Nginx*, *round-robin*) e (ii) sem balanceamento (tráfego fixado na *TV Box* 1). Em ambos, cada aba do dashboard mantém um *WebSocket* com todas as *TV Boxes* mais uma conexão exclusiva com a *TV Box* que atende o usuário, portanto, com N usuários e c nós, cada *TV Box* tem N conexões *admin* nos dois casos, e a diferença entre os cenários aparece apenas nas conexões exclusivas.

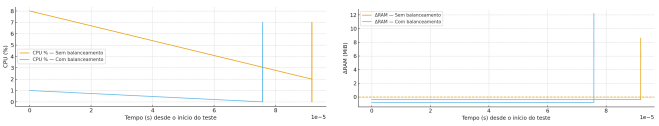
Figura 5 – Conexões por nó (N=20, c=20)



Fonte – Autoral

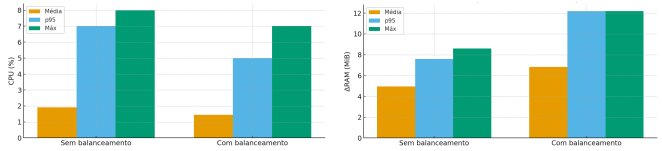
A fim de simular as conexões exclusivas de usuários reais, foram executados *bursts* de requisições *HTTP* independentes contra http://192.168.0.105/ (TVBox 1) sem *keep-alive* (conexões curtas e distintas) nos cenários i e ii para comparação.

Figura 6 – *CPU* e *RAM* ao longo do tempo (cenários i e ii)



Fonte – Autoral

Figura 7 – Resumo de desempenho de *CPU* e *RAM* (cenários i e ii)



Fonte – Autoral

Pelos gráficos de séries e de resumo, observa-se que o balanceamento reduz a carga sustentada de *CPU* na *TV Box* 1, pois a *CPU* fica perto de 0–1% quase todo o tempo e no resumo obtém ≈1,5% (média), 5% (p95) e 7% (máx), enquanto sem balanceamento esta se mantém mais alta (≈1,9% média) e com cauda pior (7% p95, 8% máx), sinal de *hotspot*. Na *RAM* normalizada (*ΔRAM*), ambos os cenários permanecem estáveis perto de zero na maior parte do ensaio; os picos são transitórios e, nesta execução, o p95/máx foi maior no balanceado (≈12,2 MiB) do que no sem balanceamento (≈7,6/8,6 MiB), indicando um pulso pontual sem tendência de acúmulo. Em suma, a diferença prática evidenciada é a *CPU* sustentada menor e a melhor cauda de *CPU* com o balanceamento, preservando *headroom* na *TV Box* 1.

Logo os resultados atingem o objetivo da pesquisa: demonstram a viabilidade da computação distribuída em ambiente de recursos limitados (*TV Boxes S905X2, 2GB)* e mostram que o balanceamento de conexões melhora o uso de *CPU* e *RAM*. Com o *cluster*, a carga fica uniformemente distribuída (sem *hotspots*), a *CPU* permanece baixa com *headroom* e a *RAM* apresenta *ΔRAM* reduzida por nó, pois sem balanceamento as conexões se concentram na *TV Box* 1, elevando a pressão de memória local. Assim, mesmo com aplicação leve, o balanceamento preserva *headroom*, evita saturação localizada e escala a capacidade com o número de nós (c), comprovando a eficácia da abordagem distribuída.

Conclusões

O trabalho demonstrou que é tecnicamente viável reaproveitar *TV Boxes* apreendidas como um *cluster* de baixo custo para aplicações de computação distribuída. A implementação de um sistema composto por *API REST + WebSocket* em *Node.js*, um *Front-End* em *React* e um servidor *Nginx* configurado como *reverse proxy* e *load balancing* permitiu validar, na prática, conceitos como distribuição de conexões *(round robin)*, monitoramento em tempo real e análise de uso de recursos de *hardware*.

Os resultados indicaram que, embora a *CPU* apresente baixa utilização devido à simplicidade da aplicação, a *RAM* se mostrou constantemente ocupada, refletindo a manutenção dos serviços em todas as unidades. Além disso, foi observado que o modelo atual de monitoramento gera conexões administrativas em todas as *TV Boxes* simultaneamente, o que impacta a eficiência do cluster. Esse aspecto abre caminho para melhorias futuras, como a adoção de balanceamento também no nível do *Front-end* de monitoramento, de modo a tornar o sistema mais escalável.

Conclui-se que o projeto atinge seu objetivo principal de comprovar a viabilidade técnica e econômica do uso de dispositivos reaproveitados como infraestrutura de computação distribuída. Além de oferecer uma solução sustentável, o estudo reforça o potencial dessas arquiteturas em ambientes educacionais e de pesquisa, servindo como alternativa prática e de baixo custo para experimentação em clusters e sistemas distribuídos.

Agradecimentos

Agradeço à Universidade Federal de Itajubá (UNIFEI) pela oportunidade de desenvolver este trabalho no âmbito da Iniciação Científica voluntária. Também agradeço ao meu orientador e colegas do curso de Engenharia de Computação pelo apoio técnico, pelas discussões e pelas contribuições que possibilitaram o avanço deste projeto.

Referências

STEEMAN. **Installation Instructions for TV Boxes with Amlogic CPUs**. Armbian Community Forums, 25 fev. 2023. Disponível em: Armbian TV Boxes Forum. Acesso em: ago. 2025

TANENBAUM, A. S.; VAN STEEN, M. **Distributed Systems: Principles and Paradigms**. 2. ed. Pearson, 2007.

NGINX. **NGINX Documentation**. Disponível em: https://nginx.org/en/docs/. Acesso em: ago. 2025.

ARMBIAN. **Armbian Documentation**. Disponível em: https://docs.armbian.com/. Acesso em: ago. 2025.

NODE.JS FOUNDATION. **Node.js Documentation**. Disponível em: https://nodejs.org/en/docs/. Acesso em: ago. 2025.

REACT. **React Documentation**. Meta Platforms, Inc. Disponível em: https://react.dev/. Acesso em: ago. 2025.

SOCKET.IO. **Socket.IO Documentation**. Disponível em: https://socket.io/docs/. Acesso em: ago. 2025.

TAILWINDCSS. Tailwind CSS Documentation. Disponível em: https://tailwindcss.com/docs. Acesso em: 20 ago. 2025.