APLICAÇÃO DE ARMAZENAMENTO EM NUVEM UTILIZANDO A PLATAFORMA NODE.JS

CAMPOS, Diego Passos Garcia; PEREIRA, Kaique Eduardo; CARVALHO, Samuel Alves de; MOREIRA, Thiago Libânio; PINHEIRO, Vinicius Santos[1]; MOREIRA JÚNIOR, Maurício[2].

[1] Acadêmicos do Curso de Ciência da Computação da Universidade José do Rosário Vellano. UNIFENAS, Campus Alfenas – MG, 2016.
[2] Professores do Curso de Ciência da Computação da Universidade José do Rosário Vellano. UNIFENAS, Campus Alfenas – MG, 2016.

Resumo: Cada vez mais, as pessoas necessitam ter acesso aos seus arquivos demasiadamente importantes, todavia sempre os esquecem em seus computadores, *pen drives*, em casa ou no trabalho, perdendo assim suas apresentações de trabalho por falta ou falha destes arquivos. Por esta razão, o conceito a que se refere como "nuvem", facilita o acesso, o armazenamento, a manipulação e a edição desses arquivos, não importando o lugar em que a pessoa esteja desde que haja alguma conexão com a internet. Sendo assim, este trabalho teve como finalidade a criação de uma aplicação em nuvem que facilite o acesso e manipulação de arquivos dos usuários, para que eles possam acessar de qualquer lugar, seja com uma conexão de internet banda larga ou até do uso das conexões 3G/4G de redes móveis. Um dos principais objetivos desse trabalho consiste em desenvolver uma nova tecnologia, que responda ao acesso de maneira rápida e simples. O uso do framework Node.js para o desenvolvimento desta aplicação foi escolhido por facilitar o desenvolvimento e tentar atender melhor às necessidades encontradas pelos usuários.

Palavras-chave: cloud, nuvem, node.js.

***Abstract*:** Increasingly, people need to have access to their files that are too important, but they always forget them on their computers, pen drives, at home or at work, thus losing their work presentations due to the lack or failure of these files. For this reason, the existence of the concept that we refer to as "cloud" facilitates the access, storage, manipulation and editing of these files, no matter where you are since there is some connection to the internet. Thus, this work aims to create a cloud application that facilitates access and manipulation of users' files, so they can access from anywhere, whether with a broadband internet connection or even the use of 3G connections / 4G of mobile networks. One of the main objectives of this work is to develop a new technology, which responds quickly and easily, the use of the Node.js framework for the development of this application was chosen for facilitating development and trying to better meet the needs encountered by Users.

Keywords: cloud, node.js.

1 INTRODUÇÃO

Armazenar arquivos importantes com o avanço da computação tem se tornado usual. Manipular e armazenar documentos, comprovantes e diversos outros arquivos, de forma segura vem se tornado cada vez mais difícil, pois estes ficam a mercê de possíveis invasões, vírus e problemas em *hardwares*. Guardá-los em um ambiente seguro e disponível em qualquer local a qualquer momento é primordial e as empresas têm investido cada vez mais em armazenamento em nuvem (internet).

Para atender todas estas demandas grandes data center's de empresas como o *GOOGLE* e sua aplicação chamada "*google drive*" são alocados centenas de servidores para um único propósito: armazenar o que o usuário tem de mais importante que são seus dados. Estes dados são

enviados e armazenados em servidores com total segurança de seus arquivos e acessível 24 horas por dia de qualquer lugar do mundo.

Utilizar o melhor de cada tecnologia para maximizar o desempenho criando uma aplicação leve e rápida é um ponto principal para qualquer empresa do ramo. Essas atribuições podem ser encontradas no framework Node.js que utiliza-se de métodos onde as requisições são processadas assim que chegam, eliminando a espera na fila e ociosidade para receber uma resposta.

Com a crescente utilização do armazenamento em nuvem é necessário entender e melhorar as suas funcionalidades e principalmente sua comunicação Cliente-Servidor, utilizando ferramentas e funções atuais no mercado, dentre elas o Node.js.

Diante do exposto, este estudo teve como objetivo geral construir uma aplicação de armazenamento em nuvem usando funções de Single Thead, programação orientada a eventos e processos assíncronos para obter maior desempenho na comunicação.

Busca-se compreender se a ferramenta desenvolvida melhorará o desempenho Cliente-Servidor de uma aplicação e a usabilidade do usuário em diversas situações, para que seus dados sejam armazenados, processados e organizados de forma mais rápida. Desta forma criando um ambiente amigável para o usuário, onde ele possa acessar seus arquivos mesmo em condições não favoráveis de acessos 3G/4G.

2 REFERENCIAL TEÓRICO

2.1 Modelo tradicional de servidores

Segundo McKie (1997) a arquitetura Cliente/Servidor vem sendo desenvolvida por muitos anos, mas a passos lentos. Desde aplicações em Mainframes de plataformas rodando o Sistema Operacional UNIX, passando por Sistemas de Banco de Dados Relacional até a importância da capacidade gráfica dos pacotes front-end existentes, melhorando a interação com o usuário.

Para Ferneda (2014) as definições da arquitetura Cliente/Servidor são: o termo Cliente/Servidor se refere às aplicações computacionais através de várias plataformas; são divididas em um servidor que detém o serviço; e, vários usuários (clientes) o acessam por meio de uma interface gráfica, ou não, para acesso e manipulação de dados. Cliente/Servidor é um modelo onde dois ou mais computadores se conectam e interagem afim de oferecer serviços uns aos outros independente do lugar em que se encontram.

Um sistema Cliente/Servidor é entendido como uma interação entre Software e Hardware em diferentes níveis. Salemi (1993) e Hulquist (1997) definem algumas características:

Cliente

- Também chamado de *font-end* ou *WorkStation*, é um processo em que a interação com o usuário, utilizando ou não uma interface gráfica permite consultas ou comandos resultando em uma apresentação dos dados.
- É um processo ativo na relação Cliente/Servidor.
- Inicia e finaliza interações com os Servidores, solicitando serviços.
- Não há comunicação com outros Clientes.

Servidor

- Também chamado de *back-end* fornece serviços aos Clientes que necessitam.
- É um processo reativo na relação Cliente/Servidor.
- Recebe e responde às solicitações dos Clientes.
- Atende a diversos Clientes simultaneamente.

Salemi (1993) ainda cita vantagens para melhor uso da arquitetura Cliente/Servidor, como: confiabilidade, mesmo como problemas parte do sistema do Servidor continua ativo; capacidade de

processamento, as tarefas são feitas sem monopolização de recursos, podendo assim, usuários trabalharem localmente em suas estações; o sistema se torna fácil de atualizar, quando for necessário; e, sistemas abertos, a utilização várias plataformas para diversos usuários atendendo as suas necessidades. Mas também há suas desvantagens como: manutenção, o desenvolvimento de ferramentas próprias para suporte, treinamento e gerenciamento da aplicação e *hardware*.

2.2 Single Thread

Segundo Reis (2013) um programa é uma sequência de instruções, composta por desvios, repetições e chamadas de funções e procedimentos. Silva, Franco e Avelino (2005) explicam que essa linha de código sendo executada no processo é chamada de *thread*. A estrutura de thread é fornecida pelo próprio sistema operacional e permite controlar o fluxo de um programa.

Segundo Welsh (ACM Symposium on Operating Systems Principles, 2001), embora haja certo grau de facilidade no desenvolvimento de aplicativos baseados em thread, por conta da adoção das mesmas em boa parte das linguagens de programação, o overhead associado as threads pode levar a uma degradação do desempenho à medida em que o número de threads aumenta. Isso se dá devido ao fato de que nesse modelo, cada requisição é atendida por uma thread separada, ou seja, quanto maior o número de conexões simultâneas, maior será o número de threads abertas.

2.3 Programação orientada em eventos

A programação orientada a eventos é um paradigma de programação que não segue um fluxo de controle padronizado, sendo que seus fluxos de controles são guiados por sinais externos. Portanto sua aplicação está diretamente ligada com o desenvolvimento de interfaces voltada para o usuário (SILVEIRA et al., 2007).

Norris (2015) define programação orientada a eventos como uma aplicação com fluxo de controle determinados por eventos ou mudanças de estado. Um mecanismo central que escuta os eventos e chama uma função de *callback* (retorno) assim que o evento for detectado.

2.4 Node.js

Pereira (2014) diz que em 2009 Ryan Dahal, com a ajuda de outros colaboradores, criou o Node.js. Uma tecnologia inovadora usando uma arquitetura *non-blocking thread* (não bloqueante), apresentando bom desempenho no consumo de memória e o uso ao máximo e de forma eficiente do poder dos processadores, principalmente em sistemas com alto fluxo de processamento.

Segundo Lopes (2014) Node.js é uma plataforma de desenvolvimento de aplicações do lado do servidor (*server-side*) que se baseia na utilização do interpretador V8 *JavaScript Engine* do *Google* que utiliza apenas código em *JavaScript*, o mesmo utilizado pelo navegador *Google Chrome*

Moreira (2013) entende que o objetivo número um do Node é fornecer uma maneira fácil para construir programas de rede escaláveis. Ao invés de criar uma nova *thread* de conexão, cada requisição dispara um evento que executa dentro da *engine*, evitando *deadlock*, não permitindo bloqueios. Suportando assim dezenas de milhares de conexões simultâneas.

2.5 NPM

Moreira (2013) explica que NPM significa *Node Package Manager* (Gerenciador de Pacotes do Node) e que pode ser representado de duas maneiras: a primeira, e mais importante, é um repositório online de publicação de projeto de código aberto para Node.js; e o segundo, é um utilitário de linha de comando que interage com este repositório online que ajuda na instalação de pacotes gerenciamento de versão e dependências. A NPM são bibliotecas e aplicações de código aberto e que uma vez encontrado o pacote que se deseja, ele pode ser instalado com uma única linha de comando.

3 MATERIAL E MÉTODOS

O projeto foi dividido em três partes, elaboração, pesquisa e implementação.

Na elaboração, foi discutido qual enfoque seria tomado pelo projeto e buscando por possíveis temas a serem abordados. Após toda a discussão chegou-se ao consenso de que o Armazenamento em Nuvem era uma tecnologia muito utilizada e com uma carência de um aplicativo que atendesse as verdadeiras necessidades dos usuários.

Depois de escolhido o tema a ser abordado, foi necessário uma intensa pesquisa em torno de quais tecnologias seriam utilizadas para o desenvolvimento do sistema, onde se procurou por plataformas de desenvolvimento, softwares gerenciadores de banco de dados, e ferramentas práticas para o desenvolvimento web.

Após, foi colocado em prática à implementação da aplicação, montando-se o servidor, onde se alocou o banco de dados, também desenvolvido nesta etapa, implementou-se as funções de comunicação necessárias para o funcionamento da nuvem e as telas de apresentação e utilização do site.

3.1 Tecnologias utilizadas

3.1.1 Node.JS

Baseado na tecnologia do runtime V8 do Chrome, o node é basicamente um conjunto de bibliotecas desenvolvidas em C++, que permite a execução de códigos em javascript. Utiliza uma arquitetura non-blocking thread (não bloqueante) e é baseada em eventos que o torna mais leve e eficiente, sendo ideal para aplicativos com alta taxa de uso e fluxo de dados, como aplicativos que de dados em tempo real. Esta arquitetura viabiliza a utilização de conexões simultâneas, exigindo menos investimento em hardware e trabalhando com mais eficiência. Neste processo entendem-se os processos síncrono e assíncrono.

Em um processo síncrono o servidor precisa espera uma requisição acabar para que seja processado o próximo evento, isto significa que os recursos dependem de uma resposta para que o próximo seja executado, como se vê nas Fig. 1, onde se criou em uma linha do tempo, criados 5 arquivos. No processo assíncrono não é necessário um tempo de resposta para que o outro evento seja acionado, como se observa na Fig. 2, de forma assíncrona, a criação dos mesmos 5 arquivos da Fig. 1, de maneira não bloqueante.

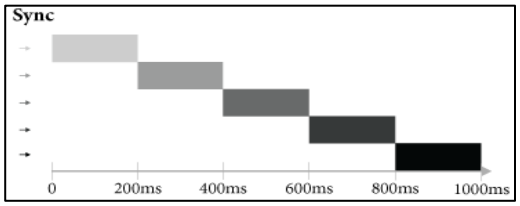

Figura1. TimeLine síncrona bloqueante

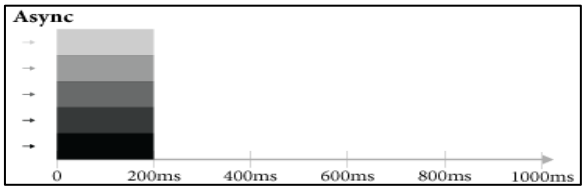

Figura 2. TimeLine assíncrona não bloqueante

Observa-se que o processo assíncrono levou 1/5 do tempo de execução, maximizando o processamento dos eventos.

3.1.2 HTML

Neste projeto, o HTML foi utilizado juntamente com o CSS, para criar telas favoráveis ao sistema. Com a ajuda dessas ferramentas criou-se todo o layout pensando na facilidade de acesso por parte do usuário, fazendo com que o sistema seja auto explicável.

Mediante os esforços para a formatação, viu-se que poderia ser melhorado as telas com uma template utilizada atualmente em vários sites já conhecidos, o bootstrap. Esse nada mais é do que uma biblioteca CSS pronta e de plataforma livre, onde se utiliza para realizar botões e terminações de telas, para que o sistema se torne mais agradável e com um layout limpo.

3.1.2 MongoDB

Lançado em 2009, o MongoDB é um banco de dados orientado documentos auto contidos e auto descritos, contém código aberto e licenciado pela AGPL (Affero General Public License). Foi criado com o intuito de servir a servidores de grande porte devido a facilidade de manipulação dos dados que este possibilita.

Diferente dos demais Sistemas Gerenciadores de Banco de Dados, o Mongo armazena seus arquivos em documentos, facilitando a alteração de campos e dados, as representações hierárquicas, o armazenamento de matrizes e outras estruturas mais complexas, além de utilizar a sintaxe JSON que armazena os dados utilizando pares de chaves/valor.

O MongoDB possui dois tipos de construções separadas para topologias multi nodes, replica sets e replica sets distribuídos. Quando se aumenta o número de instancias em uma replica set, aumenta a tolerância a falhas, enquanto aumentar o número de shards pode-se distribuir os dados para processos mongodb separados, o que provê uma escalabilidade horizontal para desempenho de leitura e escrita.

3.1.4 Configurações do Servidor

O sistema operacional utilizado pelo servidor é o linux server, pois processos do Linux rodam separadamente e isso faz com que não haja a necessidade de reiniciar o sistema inteiro caso um deles apresente qualquer problema. Juntamente com o sistema operacional foi utilizado o BIND, um servidor de protocolos DNS mais utilizado na internet, criado para sistemas Unix.

Foi instalado o RAID (redundant array of independent disks), que tem como objetivo melhorar o desempenho e segurança dos arquivos. Foram feitas as configurações de RAID1 (espelhamento) que utiliza um ou mais discos em conjunto para que os dados sejam espelhados em tempo real, mas as funções de segurança é o Iptables, uma ferramenta para criar e administrar regras e assim filtrar pacotes de redes, podendo funcionar baseado no endereço, porta de origem, destino do pacote, prioridade.

Para o servidor web, foi utilizado o Nginx que tem como objetivo suportar um tráfego intenso de serviços, por isso utiliza uma arquitetura orientada a eventos, possibilitando uma taxa de conexão maior e de mais eficiência, exigindo menos esforço de hardware.

3.2 TAQUI CLOUD

O TaquiCloud é um sistema de Computação em nuvem, no qual, os usuários fazem o seu cadastro, podendo assim, ter acessibilidade à todas as funções do sistema, que concerne em armazenar os arquivos no servidor, uma vez salvos, eles ficam acessíveis de quaisquer lugar, contando que haja internet.

3.2.1 Descrição do sistema.

O sistema desenvolvido tem como objetivo utilizar ao máximo todas as vantagens da plataforma Node.js. No sistema o usuário tem as funcionalidades básicas de um sistema de armazenamento em nuvem. Sendo implementadas as funcionalidades *download* e *upload*.

As funções têm a mesma finalidade da forma convencional, no *download* o usuário pode escolher qual arquivo ele já previamente enviou para o servidor e assim baixá-lo para seu dispositivo e manipulá-lo. Enquanto no *upload* ele enviará um arquivo de seu dispositivo para o

servidor, onde ficará armazenado e vinculado a sua conta. Estes processos utilizam a comunicação assíncrona, para enviar ou buscar requisições.

3.2 Comunicação Assíncrona

A Comunicação assíncrona é um dos pontos principais da plataforma node.js, pelo fato de atender todas as requisições em um único thread, é fundamental que o código Node.js invoque o mínimo possível de funções bloqueantes. Toda função síncrona impedirá, naquele instante, que o Node.js continue executando os demais códigos até que aquela função seja finalizada.

Pode-se destacar isso no código-fonte da aplicação abaixo (Fig. 3), cuja função se pode observar onde carrega os arquivos da pasta referente ao caminho. Primeiramente vasculha todos os arquivos da pasta do usuário, depois em um segundo passo definido como "assíncrono.map" percorre a lista de forma assíncrona chamando a função "verifica", que também será descrito a seguir, e logo após retorna em Json o resultado entregando a lista ao usuário.

```
27      fs.readdir(caminho, function(err, files){
28        assincrono.map(files, verificaarquivo,
29          function(err, results){
30            console.log(results);
31            return res.json({
32              Nome : user.nome,
33              Email : user.email,
34              Caminho : caminho,
35              Arquivos : results});
36          });
37       });
38    });
```

Figura 3. Função assíncrona listagem

A função "verifica", descrita na Fig. 4 observa qual o tipo de arquivo é encontrado na busca, o tamanho e o nome deste arquivo, retornando em callbak para a função principal com estas informações.

```
39  function verificaarquivo(f,callback)
40  {
41    var aux;
42    var file =caminho + f;
43     fs.stat(file,function(error,stats)
44     {
45       if(error){
46         console.log("erro verifica");
47       }
48       if(local=="1"){
49         var url = f;
50       }
51       else {
52         var url = local +'$'+f;
53       }
54         if(stats.isDirectory()){
55           aux = {    nome: f,
56                      tipo:'Pasta',
57                      url: url};
58         }
59         else{
60          aux = {     nome: f.substring(0,f.lastIndexOf('.')),
61                      tipo:f.substring(f.lastIndexOf('.')+1),
62                      size: tam(stats.size),
63                      url: url};
64         }
65         callback(null, aux);
66       });
67  }
```

Figura 4. Função assíncrona de verificação dos arquivos

Todo esse processo é efetuado em apenas uma theread, sendo assim quando a função é solicitada o servidor não fica aguardando a resposta do final da execução, assim ela esta pronta para receber outra requisição, isso só e possível graças ao desenvolvimento orientado a eventos.

3.2.1 Programação Orientada a Evento

Um dos principais aspectos do node.js e a programação a eventos, onde todos são monitorados no *event-loop*, como se observa na Fig. 5, a função tem como trabalho deletar arquivo

e/ou diretório, onde a mesma recebe o caminho que será apagado, caso for um diretório ele primeiramente verifica se ele esta vazio, pois por se tratar de um servidor Linux existe esta permissão por definição, caso o diretório esteja vazio a função retorna em callbak para que seja apagada.

```
function deleteDiretorio(diretorio, cb) {
  diretorio += "\\";
  console.log("deletar Diretorio "+ diretorio);
    var files = [];
    var curPath = '';
    if (fs.existsSync(diretorio)) {
      fs.readdir(diretorio, function(err, files){
        assincrono.map(files, function(file,callback){
          var aux = diretorio + file
          console.log("vericando "+ aux);
            if (fs.statSync(aux).isDirectory()){
                deleteDiretorio(aux, callback);
            }else{
              console.log("deletar arquivo "+ aux);
                fs.unlinkSync(aux);
                return callback(null);
              }
        },function(err, results){
            console.log("Diretorio Deletado "+ diretorio);
            fs.rmdirSync(diretorio);
            cb(null);
          });
        });
    } else {
        cb(new Error('Erro o Diretorio Não Existe'));
    }
};
```

Figura 5. Função para deletar arquivo

Quando esse evento é acionado é enviado para o event-loop que monitora e executa esse evento durante o processo, esperando uma resposta do sistema para dar continuidade ao processo, esse conceito de tratamento de evento só é possível graças ao Node.js ser executado em modo single-theread.

3.3 Angular

Diferentemente de outros frameworks JavaScript, ele adota uma abordagem mais ligada à sintaxe HTML, funcionando como uma espécie de extensão da linguagem. Para interação com a plataforma são gerados os JavaScripts então quando necessário é criado este para a interação do script com o HTML. Foi utilizado para fazer a verificação de login no sistema conforme demonstra a Fig. 6.

```
app.controller('LoginCtrl',function($scope, $http, $location,usuariosService) {
    // Quando clicar no botão Criar, envia informações para a API Node
    $scope.LogarUsuario = function() {
      $http.post('/api/login', $scope.formUsuario)
          .success(function(data) {
            if (data.status) {
                $scope.userForm.email.$setValidity("incorreto",true);
              if(data.user.senha==$scope.formUsuario.senha)
              {
                $scope.userForm.senha.$setValidity("incorreto",true);
                usuariosService.validaLogin(data.user);
              }
              else {
                $scope.userForm.senha.$setValidity("incorreto",false);
              }
            } else {
              $scope.userForm.email.$setValidity("incorreto",false);
            }
          })
          .error(function(data) {
          });
    };
});
```

Fig. 6. Função angular para verificar login

Onde o app.controller verifica o login e senha e depois repassa ao HTML a validação ou erro para que o usuário possa ou não acessar sua conta.

3.4 Mongo

O MongoDB foi utilizado para armazenar as informações do usuário como se observa na Fig. 7.

Figura 7. Campos do banco de dados

O MongoDB, no desenvolvimento da aplicação obteve grande desempenho graças a sua estrutura Orientado a objeto, onde a cada novo usuário que é cadastrado no sistema um novo objeto é criado para ele, como pode-se ver na Fig. 8.

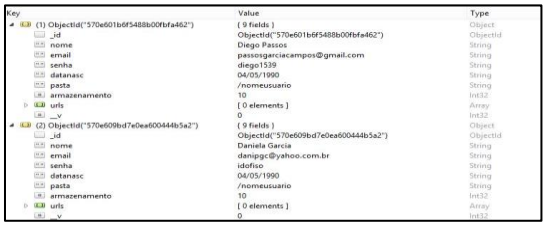

Figura 8. Estrutura do banco de dados

Outro conceito que se pode destacar do MongoDB, como já explicado anteriormente é o seu trabalha com o sistema NoSQL.

3.5 Utilização do Taqui Cloud

A ferramenta tem como objetivo o armazenamento em nuvem, e para separação dos arquivos de cada usuário foi utilizado o controle de acesso por usuário, onde o mesmo fará a autenticação com um Login e Senha que serão verificados e seu acesso liberado. Na tela de acesso ele preenchera conforme pedido, ou ate mesmo criar uma nova conta.

Figura 9. Tela de acesso

Se o usuário não estiver cadastrado no sistema este poderá fazer um cadastro para acesso, devendo preencher alguns campos obrigatórios para que a conta seja liberada.

Figura 10. Tela de cadastro

Com acesso concedido antes ou depois do cadastro o usuário poderá ver seus arquivos que já estão armazenados e assim manipulá-los da maneira que desejar, podendo ter as opções de baixar para sua máquina local ou deletar, onde será excluído permanentemente. Os arquivos de cada

usuário serão listados conforme a Fig. 11 e num menu lateral poderá escolher qual funcionalidade deseja.

Figura 11. Listagem de arquivos

4 RESULTADOS E DISCUSSÕES

Com aumento dos clientes acessando o aplicativo e demais funções seria necessário a adição de servidores extras para que seja suportado o fluxo, sem interferir na performance da aplicação. Com a ferramenta Node.js pode se fazer comparativos de criação e leitura de arquivos em rede local, para que isso possa ser escalável e se tenha uma visão do seu funcionamento na web.

Para verificar o desempenho da plataforma foram feitos testes de criação e leitura de arquivos textos simples em uma máquina convencional com um processador Intel Core I5, 4GB de memoria RAM e armazenamento em disco de estado sólido. Estes testes foram feitos de forma síncrona e assíncrona. E os tempos de leitura e gravação foram analisados.

Pode-se ver um comparativo onde foram criados 200.000 arquivos primeiramente foram analisados de forma síncrona onde podemos ver no Script abaixo.

```
var fs = require('fs');
var inicio = new Date().getTime();

        for(var i = 1; i <= 200000; i++) {
            var file = "Sincrono " + i + ".txt";
            fs.writeFileSync("E:\\TCC_Testes_NodeJs\\arq_sync\\" + file, "Arquivo Sincrono !!!");
}

var fim = new Date().getTime();
console.log("Tempo p/ Criacao Modo SINCRONO: "+(fim - inicio)+ " ms");
```

Figura 12. Script de criação de arquivos de forma síncrona

O tempo de resposta foi de 1.219 milissegundos como demonstrado no teste (Fig. 13).

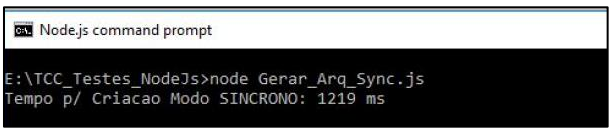

Figura 13. Tempo de criação de arquivos de forma síncrona

A criação de arquivos de forma assíncrona seguiu o mesmo padrão de criação conforme o script (Fig. 14).

```
var fs = require('fs');
var inicio = new Date().getTime();


for(var i = 1; i <= 200000; i++)
{
        var file = "Assincrono " + i + ".txt";
        fs.writeFile("E:\\TCC_Testes_NodeJs\\arq_async\\" + file, "Arquivo Assincrono !");
};

var fim = new Date().getTime();
console.log("Tempo p/ Criacao Modo ASSINCRONO: "+(fim - inicio)+ " ms");
```

Figura 14. Script de criação de arquivos de forma assíncrona

O tempo de resposta para a criação de 200.000 arquivos de forma assíncrona foi de 156 milissegundos mostrando um desempenho de pouco mais de 7 vezes maior que de forma síncrona como pode ser comprovado no tempo abaixo (Fig. 15).

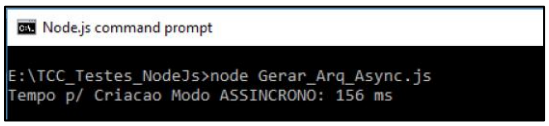

Figura 15. Tempo de criação de arquivos de forma assíncrona

Outro comparativo feito foi a leitura destes arquivos utilizando o método a partir de um script para a leitura de todos os 200.000 arquivos. Para melhor análise realizaram-se 3 leituras e uma média entre estas foram comparadas.

```
var fs = require('fs');
var  tam = require('prettysize');
var inicio = new Date().getTime();
i=0;
var lista=[];

        var files=fs.readdirSync("E:\\TCC_Testes_NodeJs\\arq_sync\\");

            files.forEach(function(f)
            {
              var stats = fs.statSync("E:\\TCC_Testes_NodeJs\\arq_sync\\" + f);
                if(stats.isDirectory())
                {
                  lista.push({ nome: f,
                               tipo:'Pasta',
                             });
                               //console.log(lista.pop().nome);
                }
                else
                {
                   lista.push({ nome: f.substring(0,f.lastIndexOf('.')),
                                tipo:f.substring(f.lastIndexOf('.')+1),
                                size: tam(stats.size) });
                                console.log(lista.pop().nome);
                }
            });
```

Figura 16. Script de leitura síncrona

A média de leitura depois de três análises foi de 26309 milissegundos de forma síncrona como pode ser visto na figura 17.

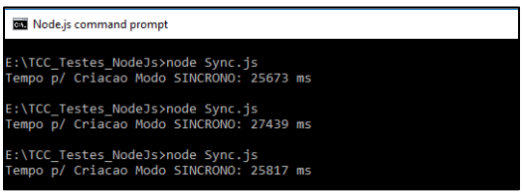

Figura 17. Tempos de leituras síncrona

Foi realizado o mesmo teste de forma assíncrona utilizando o Script da Fig. 18, sendo o mesmo utilizado na aplicação para leitura do diretório do usuário.

```
var fs = require('fs');
var assincrono = require('async');
var  tam = require('prettysize');
var inicio = new Date().getTime();
var lista=[];

fs.readdir("E:\\TCC_Testes_NodeJs\\arq_sync\\", function(err, files){
  assincrono.map(files, verificaarquivo,
    function(err, results){
      console.log("Terminado");
      tempo();
    });
 });

 function verificaarquivo(f,callback)
 {
   //console.log("Vamos verificar os arquivos");
   var aux;
   var file = "E:\\TCC_Testes_NodeJs\\arq_sync\\" + f;
    fs.stat(file,function(error,stats)
    {
      if(error){
        console.log("erro verifica");
      }
        if(stats.isDirectory()){
          aux = {    nome: f,
                     tipo:'Pasta'};
                //  console.log(lista.pop().nome);
        }
        else{
         aux = {     nome: f.substring(0,f.lastIndexOf('.')),
                     tipo:f.substring(f.lastIndexOf('.')+1),
                     size: tam(stats.size)};
                //  console.log(lista.pop().nome);
        }
        console.log(aux.nome);

        callback(null, aux);
      });
 }


function tempo(){
var fim = new Date().getTime();
console.log("Tempo p/ Leitura Modo ASSINCRONO: "+(fim - inicio)+ " ms");
}
```

Figura 18. Script de leitura assíncrona

O tempo de leitura destes arquivos foi feita uma média a partir de 3 análises e teve como média 12057 milissegundos, uma diferença de 2 vezes na leitura dos mesmos 200.000 arquivos visto anteriormente. Na Fig. 19 pode-se observar os tempos comprovando as leituras assíncronas.

```
Node.js command prompt

E:\TCC_Testes_NodeJs>node Async.js
Terminado
Tempo p/ Leitura Modo ASSINCRONO: 12572 ms

E:\TCC_Testes_NodeJs>node Async.js
Terminado
Tempo p/ Leitura Modo ASSINCRONO: 11819 ms

E:\TCC_Testes_NodeJs>node Async.js
Terminado
Tempo p/ Leitura Modo ASSINCRONO: 11781 ms
```

Figura 19. Tempos de leituras assíncrona

Pode-se ver o grande impacto que a ferramenta trabalha em maiores escalas. Tanto o tempo de leitura quanto de gravação teve um desempenho superior quando foi submetido aos testes assíncrono.

5 CONCLUSÃO

Pode-se concluir que utilizando a ferramenta Node.js, para a criação da aplicação, tem-se um desempenho muito superior comparado com outras ferramentas síncronas convencionais, pois ele se baseia no conceito de programação assíncrona.

Tal conceito possibilita o aumento de desempenho na leitura dos arquivos/diretórios, com isso o servidor consegue processar mais requisições simultâneas.

Seu aumento no desempenho prove da utilização do tempo ocioso do processamento no servidor, sem interferir no seu SO. Com os testes realizados o tempo de geração e leitura dos arquivos, foi extremamente satisfatório, com isso, a aplicação de armazenamento em nuvem se tornou viável e pensando no atendimento de requisições em larga escala, o seu ganho de processamento e sua alta escalabilidade, fez do Node.js a melhor escolha para este projeto.

5 REFERENCIAS

ANGULAR MATERIAL. **Getting started**. Disponível em: <https://material.angularjs.org/latest/getting-started>. Acesso em: 20 ago. 2016.

CARLOS E. MORIMOTO. **Thread**. Disponível em: <http://www.hardware.com.br/termos/thread>. Acesso em: 06 ago. 2016.

DEPIJAMA. **O que é node.js**. Disponível em: <http://www.depijama.com/o-que-e-node-js/>. Acesso em: 17 jul. 2016.

DEVMEDIA. **Criando aplicações web com angularjs, node e concise css**. Disponível em: <http://www.devmedia.com.br/criando-aplicacoes-web-com-angularjs-node-e-concise-css/36777>. Acesso em: 20 ago. 2016.

DEVMEDIA. **Introducao ao mongodb**. Disponível em: <http://www.devmedia.com.br/introducao-ao-mongodb/30792>. Acesso em: 16 jul. 2016.

ENVATOTUTS. **Autenticando aplicações node.js com passport**. Disponível em: <https://code.tutsplus.com/pt/tutorials/authenticating-nodejs-applications-with-passport--cms-21619>. Acesso em: 16 jul. 2016.

ENVATOTUTS. **Autenticação com tokens usando angularjs & nodejs**. Disponível em: <https://code.tutsplus.com/pt/tutorials/token-based-authentication-with-angularjs-nodejs--cms-22543>. Acesso em: 22 jul. 2016.

EVENTIALS. **Construindo uma api rest com expressjs - nodejs**. Disponível em: <https://www.eventials.com/wbruno.moraes/construindo-uma-api-rest-com-expressjs-nodejs-2/>. Acesso em: 23 jul. 2016.

IBM DEVELOPERWORK. **Desenvolva um aplicativo de pesquisas em tempo real com node.js, express, angularjs e mongodb**. Disponível em: <https://www.ibm.com/developerworks/br/library/wa-nodejs-polling-app/>. Acesso em: 16 jul. 2016.

IFTM. **Tic - programacao visual**. Disponível em: <http://www.esj.eti.br/iftm/disciplinas/grau02/pv/pv_unidade_13.pdf>. Acesso em: 06 ago. 2016.

INFOBRASIL. **Programação orientada a eventos no lado do servidor utilizando node.js**. Disponível em: <http://www.infobrasil.inf.br/userfiles/16-s3-3-97136-programa%c3%a7%c3%a3o%20orientada___.pdf>. Acesso em: 22 jul. 2016.

INFOQ. **Apresentando o node.js**. Disponível em: <https://www.infoq.com/br/presentations/apresentando-o-node-js>. Acesso em: 22 jul. 2016.

INTEL. **Raid 0,1,5,10,raid em matriz,pronto para raid para a tecnologia de armazenamento intel® rapid intel®**. Disponível em: <http://www.intel.com.br/content/www/br/pt/support/boards-and-kits/000005867.html>. Acesso em: 17 jul. 2016.

MATERA. **Controle de acesso com angular-route**. Disponível em: <http://www.matera.com/br/2016/04/25/controle-de-acesso-com-angular-route/>. Acesso em: 20 ago. 2016.

MCKIE. **Fundamentos da arquitetura cliente/servidor**. Disponível em: <https://www.marilia.unesp.br/home/instituicao/docentes/edbertoferneda/fundamentos_da_-arquitetura_cliente-servidor.pdf>. Acesso em: 06 ago. 2016.

NODESOURCE. **Understanding the node.js event loop**. Disponível em: <https://nodesource.com/blog/understanding-the-nodejs-event-loop/>. Acesso em: 06 ago. 2016.

PABLO JUAN MSP. **Configurando ambiente mongodb no windows**. Disponível em: <https://pablojuancruz.wordpress.com/2014/09/03/configurando-ambiente-mongodb-no-windows/>. Acesso em: 16 jul. 2016.

RCDEV LABS. **12 feb como criar uma api restfull em nodejs e autenticar usando json web token jwt?**. Disponível em: <http://rcdevlabs.github.io/2015/02/12/como-criar-uma-api-restfull-em-nodejs-e-autenticar-usando-json-web-token-jwt/>. Acesso em: 18 ago. 2016.

RUSCHEL, Henrique; ZANOTTO, Mariana Susan; MOTA, Wélton Costa Da. Computação em Nuvem. **PPGIA**, [S.L], abr. 2010. Disponível em: <http://www.ppgia.pucpr.br/~jamhour/RSS/TCCRSS08B/Welton%20Costa%20da%20Mota%20-%20Artigo.pdf>. Acesso em: 05 ago. 2016.

SILVA, Daniel Da; FRANCO, Carla E. De Castro; Diogo Florenzano Avelino. Implementação de sockets e threads no desenvolvimento de sistemas cliente / servidor: um estudo em VB.NET. **Inf aedb**, [S.L], abr. 2010. Disponível em: <http://inf.aedb.br/seacIV/SI/Artigos/G3.pdf>. Acesso em: 05 ago. 2016.

SITEPOINT. **Accessing the file system in node.js**. Disponível em: <https://www.sitepoint.com/accessing-the-file-system-in-node-js/>. Acesso em: 23 jul. 2016.

SLIDEPLAYER. **Thead**. Disponível em: <http://slideplayer.com.br/slide/1870257/>. Acesso em: 06 ago. 2016.

SOUSA, Flávio R. C.; MACHADO, Leonardo O. Moreira E Javam C.. Computação em Nuvem: Conceitos, Tecnologias,Aplicações e Desafios. **Rechargate**, [S.L], set. 2015. Disponível em: <https://www.researchgate.net/publication/237644729_Computacao_em_Nuvem_Conceitos_Tecnologias_Aplicacoes_e_Desafios>. Acesso em: 05 ago. 2016.

TABLESS. **Raspagem de dados com o node.js**. Disponível em: <http://tableless.com.br/raspagem-de-dados-com-node-js/>. Acesso em: 26 set. 2016.

TAURION, Cezar. **Transformando o mundo da tecnologia da informação**. [S.L.]: Brasport, 2009.

TUTORIALS POINT. **Node.js file system**. Disponível em: <http://www.tutorialspoint.com/nodejs/nodejs_file_system.htm>. Acesso em: 23 jul. 2016.

UNDERGROUND WEBDEV. **Trabalhando com validators no node.js**. Disponível em: <https://udgwebdev.com/trabalhando-com-validators-no-node-js>. Acesso em: 18 ago. 2016.

WBRUNO. **Validando formulários apenas com html5**. Disponível em: <http://wbruno.com.br/html/validando-formularios-apenas-com-html5/>. Acesso em: 18 ago. 2016.

YOUTUBE. **Node.js - #1 - introdução - rodrigo branas**. Disponível em: <https://www.youtube.com/watch?v=ktdwdoxql4a>. Acesso em: 22 jul. 2016.