

# Sistemas de informação escaláveis utilizando programação orientada a eventos e NoSQL

## *Scalable information system using event oriented programming and NoSQL*

Vagner José Santana, Gabriel Spadon de Souza, Ronaldo Celso Messias Correia,

Rogério Eduardo Garcia, Danilo Medeiros Eler, Celso Olivete Júnior

Departamento de Matemática e Computação (DMC)

Faculdade de Ciências e Tecnologia – FCT/UNESP

Presidente Prudente, Brasil

vagnervjs@gmail.com, gabriel@spadon.com.br, {ronaldo, rogerio, daniloeler, olivete}@fct.unesp.br

**Resumo —** Em sistemas de informação web, a capacidade de se manter estável e disponível, enquanto o número de usuários e requisições aumentam, é um desafio; a isto denomina-se escalabilidade. Este trabalho propõe um modelo para desenvolvimento de sistemas baseados em orientação a eventos de forma não-bloqueante, sobre o ambiente Node.js em conjunto com o modelo de banco de dados não-relacional MongoDB. O objetivo deste trabalho é manter a escalabilidade do sistema apresentado, analisando seus resultados de forma comparativa com um ambiente que faz uso de um banco de dados relacional (MySQL), servidor HTTP Apache e linguagem PHP. Os resultados demostram que o modelo proposto, MongoDB e Node.js, é o mais eficiente, pois apresenta um tempo de resposta de seis à oito vezes mais rápido do que o modelo comparado, podendo assim, ser utilizado em ambientes de produção, com ganho de desempenho e escalabilidade.

**Palavras-Chave:** Node.js; MongoDB; Escalabilidade.

**Abstract —** One of many challenges in a web information systems is the capability of staying stable and available as the number of users and requests increase rapidly, what we call scalability. This paper provides a model based on development of events-oriented applications, non-blocking in the environment Node.js with the non-relational database model (MongoDB), preparing to establish scalability and a comparison with a traditional model (relational database (MySQL), HTTP Apache server and PHP language). The results obtained reveal that the proposed model is more efficient (response time of six to eight faster) than the comparative model, can be used in production environments with development quality, performance improvement and scalability.

**Keywords-component:** Node.js; MongoDB; Scalability.

### I. INTRODUÇÃO

O número de usuários que acessam a internet tem aumentado consideravelmente na última década. Pesquisas apontam que a tendência para os próximos anos é que o número de usuários apresente um nível de crescimento contínuo, tendo em vista a variedade de dispositivos conectados na internet, especialmente dispositivos móveis. No

ano de 2012 haviam 2,4 bilhões de usuários conectados na internet [1]. Atualmente, há mais de 5 bilhões de usuários [2], ou seja, em um período de três anos, houve um crescimento superior a 50% do número de usuários.

Um sistema de informação deve garantir a escalabilidade, mantendo-se disponível e funcional. Os modelos de escalabilidade existentes são amplamente utilizados e suprem as necessidades atuais [3]. Não obstante, quando aplicados em software, existem pontos de ganho de desempenho e redução de custo, assim como existem pontos de perda de desempenho, aumento de custo, além da igualdade entre ganho e perda; o que constitui o objetivo deste trabalho, uma aplicação e avaliação de desempenho do modelo computacional aplicado à software.

O objetivo deste trabalho é propor um modelo de desenvolvimento de sistemas web baseados em orientação a eventos de forma não-bloqueante, no ambiente Node.js, em conjunto com o modelo de banco de dados não-relacional MongoDB, visando à obtenção da escalabilidade e à análise de seus resultados de forma comparativa, com um ambiente que faz uso de um banco de dados relacional (MySQL), servidor HTTP Apache e linguagem PHP.

Este trabalho está organizado em sete seções. Além desta, a Seção 2 apresenta pesquisas sobre escalabilidade com utilização do MongoDB. A Seção 3 expõe o conceito de escalabilidade e demonstra os detalhes sobre as operações de entrada e saída bloqueantes e não-bloqueantes. Na Seção 4 é apresentado o modelo proposto neste trabalho, sua estrutura e as tecnologias utilizadas. A Seção 5 expõe as características e estruturas da aplicação desenvolvida. Na Seção 6 são apresentadas as análises de resultados, descrevendo as metodologias pormenorizadas, utilizadas para elaboração dos testes. Por fim, na Seção 7 são expostas as considerações gerais, analisando os pontos relevantes do trabalho, bem como alguns encaminhamentos futuros.

## II. TRABALHOS CORRELATOS

Entre os trabalhos que se utilizam do Mongo DB, dois se destacam dentre os demais. No primeiro [4] o autor demonstra a utilização do MongoDB em um *framework* de aprendizado analítico em ambiente 3D. Em seu trabalho, destaca como desafio os grandes volumes de dados e suas fontes distintas, além de apresentar o banco de dados NoSQL como uma ferramenta capaz de armazenar e organizar os dados em documentos estruturados e de fácil identificação [5].

O segundo trabalho é de Correia, Aguiar e Pinto (2014). Os autores realizam uma extensa análise de escalabilidade sobre um ambiente de correio eletrônico em nuvem, apresentando resultados que evidenciam que os fatores de adoção da computação em nuvem são a escalabilidade e a redução de custos. Seus resultados reforçam o fato de que análises de desempenho e escalabilidade reduzem custos e trazem melhorias no suporte a processos de negócio e nos impactos organizacionais [3].

## III. ESCALABILIDADE EM OPERAÇÕES DE ENTRADA E SAÍDA

O Problema C10k é um paradigma bastante conhecido quando se trata de escalabilidade em sistemas de informação web. Neste, constata-se que a maioria dos servidores web suportam no máximo 10.000 (dez mil) conexões simultâneas, número este que antes era realidade apenas para grandes provedores de serviços *online*, por exemplo, o site *cdrom.com* que, no ano de 1999, respondia cerca de 10.000 requisições simultâneas, utilizando-se de uma placa ethernet Gigabit [6]. Atualmente, este número é comum até mesmo em pequenos sistemas web, devido à dinamicidade e fluxos de informações.

O *hardware* não é o único ponto de falha de uma aplicação servidora. As operações de I/O têm se apresentado como entraves tão grandes quanto à insuficiência de *hardware*. Essas operações podem ser remodeladas para alcançar melhor desempenho da aplicação, uma vez que este problema surge no processo de tratamento de requisições simultâneas pelo sistema operacional, apresentando uma evolução em desempenho entre versões diferentes de kernel [6,7].

Para tratar o problema da concorrência é comum o uso de *threads*, uma forma de um processo se dividir em duas ou mais tarefas concorrentes, influenciado pela possibilidade de utilizar múltiplos processadores e núcleos da máquina. Apesar da facilidade de desenvolver aplicações concorrentes, à medida que o número de *threads* aumenta, o processamento ou armazenamento em excesso, conhecido como *overhead*, provoca a queda do desempenho. Cada requisição é atendida por uma *thread* diferente, sendo assim, quanto maior o número de requisições, maior o número de *threads*, e a cada nova *thread*, é gerado um novo descritor de arquivo que, para o sistema operacional é visto como um processo em execução, ativo ou em espera, o que acarreta alto consumo de memória [8,9].

Para as *threads*, deve-se considerar que as implementações com a utilização de filas provêm melhor desempenho quando se executam um grande número de tarefas, devido à um *overhead*, porém, o grande número de tarefas não significa requisições simultâneas, uma vez que, uma única requisição

pode desencadear uma grande quantidade de tarefas e, igualmente, causar um alto consumo de recursos de *hardware*.

A forma com que os servidores podem tratar as operações de I/O é relevante, pois resultará em desempenhos dispareus. O modelo bloqueante é o mais comum nos servidores web, ao chamar uma operação, por exemplo, efetuar uma leitura inexistente, a aplicação deve aguardar uma resposta da operação para poder continuar a execução; o mesmo exemplo utilizando o comportamento não-bloqueante poderia tornar a ação mais ágil, enviando uma resposta imediata, informando à inexistência de leitura [10]. O modelo não-bloqueante permite trabalhar de duas maneiras: tentar efetuar uma operação em um determinado tempo, conhecido por *polling*, ou então utilizar a notificação assíncrona, gerando um evento que informe a operação a ser realizada.

O novo modelo de servidores web propõe o uso de operações de I/O não-bloqueantes, evitando o elevado número de *threads*. Assim, torna-se possível atender a um maior número de requisições em uma única máquina.

## IV. ESPECIFICAÇÃO E IMPLEMENTAÇÃO DO MODELO PROPOSTO

O modelo proposto neste trabalho, cujo foco é o desenvolvimento de sistemas de informação web escaláveis, utiliza a arquitetura de três camadas:

- Camada Visual: Representa a camada de apresentação de conteúdo;
- Camada Lógica: Baseia-se no paradigma de orientação a eventos para gerenciar requisições e operações de concorrência, utilizando *Node.js*;
- Camada de Dados: Representa a utilização do modelo não-relacional, no caso, o *MongoDB*;

Com as diversas tecnologias empregadas, realizou-se testes para estabelecer parâmetros comparativos de escalabilidade e que permitissem a comparação com outros modelos existentes e amplamente utilizados.

A seguir é apresentado em detalhes duas das três partes do modelo anteriormente descrito.

### A. Camada de Dados - Modelo não-Relacional e *MongoDB*

O atual momento em que a área de tecnologia se encontra é denominado de “*a era dos dados*” (*Big Data Age*); esse fenômeno de crescimento contínuo do volume de dados está diretamente relacionado à quantidade de pessoas que fazem uso da internet e, principalmente, por esse número ter um aumento constante.

Anterior ao conceito de Big Data, havia um conjunto bem definido de linguagens e abordagens tecnológico-científicas para o gerenciamento de informações, com as maiores restrições e limitações relacionadas aos dados, sobretudo por conta do tipo de informação disponível, pelo custo de armazenamento de dados e ainda por conta da capacidade de processamento. Atualmente, os limites e barreiras impostas foram ampliados devido ao avanço da tecnologia e, então houve a necessidade da criação de arcabouços de métodos e

paradigmas para auxiliar no processo de coleta e manuseio de grandes volumes de dados.

A baixa flexibilidade do modelo de banco de dados relacional, e a sua limitação sobre grandes volumes de dados, influenciaram na disseminação e evolução do modelo não-relacional, ou NoSQL. Este modelo visa a escalabilidade e desempenho, sendo ideal para manipular grandes volumes de dados.

Alguns fatores influenciam na escolha do modelo não-relacional em detrimento do relacional, são: maior disponibilidade, menor tempo de resposta para consultas, paralelismo de atualização de dados e maior grau de concorrência [11].

Bancos de dados não-relacional apresentam uma vantagem quando comparados aos bancos de dados relacionais, isto é, quando trata-se de concorrência, o modelo relacional cria um bloqueio para que dois usuários não utilizem o mesmo item simultaneamente, enquanto que nos bancos de dados não-relacional, o mesmo não ocorre [11,12].

Neste trabalho optou-se por utilizar o MongoDB, devido a sua grande interação com a linguagem JavaScript. O modelo não-relacional do MongoDB é baseado em coleções de dados, que por sua vez, são formadas por documentos, os quais armazenam os valores referentes à um certo tipo de dado da aplicação (baseados em grupos de chaves e valores) [13]. Esses documentos têm o mesmo estilo de objetos da linguagem JavaScript, conhecido como JSON. Para serem tratados neste banco de dados, são um tipo ainda mais específico, denominados BSON, uma representação binária do documento JSON [14].

#### *B. Camada Lógica - Paradigma de Programação Orientada a Eventos com Node.JS*

A programação orientada a eventos é um paradigma que não segue um fluxo de controle padronizado, sendo apenas guiado por sinais externos. Esse é constantemente associado ao desenvolvimento de interface de usuário, em que, em cada interação é gerado um evento e o mesmo pode ser tratado fixando o objetivo desse paradigma.

Assim como o sistema operacional, os servidores também geram eventos, por exemplo: nova conexão estabelecida, conexão em espera ou conexão fechada. Dessa maneira, pode-se aproveitar a capacidade assíncrona e criar sistemas web baseados no paradigma de orientação a eventos. Diferente do modelo convencional requisição-resposta, foi modificado o modelo para a orientação à eventos, criando o produtor de eventos (*event producer*) e o consumidor de eventos (*event consumer*), gerando a interação não-bloqueante [8,9].

O Node.JS, por sua vez, possibilita utilizar o modo de desenvolvimento assíncrono e não-bloqueante, caracterizando-se por uma plataforma capaz de construir aplicações de rede rápidas e escaláveis [15], provendo um módulo específico para tratamento de produção e consumo de eventos desencadeados na aplicação, viabilizando a adição e criação de *call-backs*, retorno de métodos para determinado evento. Não obstante, os demais módulos desta linguagem adotam a

mesma técnica de consumo de eventos, o que torna todo o ambiente Node.JS não-bloqueante.

Apesar de estar em constante desenvolvimento, o Node.JS é uma aplicação robusta e de grande capacidade, cumprindo seu objetivo de criar um ambiente para desenvolvimento de aplicações escaláveis e de alto desempenho, porém, sua utilização deve ser ponderada em relação à necessidade do projeto, considerando o risco de optar por uma tecnologia volátil.

#### V. ESTRUTURA DA APLICAÇÃO TESTE

A aplicação desenvolvida se constitui como um módulo para um sistema acadêmico, com foco em disponibilização de materiais para alunos. A aplicação deve ser capaz de armazenar dados de milhares de alunos que possuem relação com os cursos disponíveis. Cada curso possui disciplinas e cada disciplina possui um professor. Os professores podem disponibilizar materiais no sistema para as diversas turmas e, cada aluno poderá gerar informações a partir de sua interação com a aplicação. Para cada material será possível que cada aluno, com permissão de acesso, crie comentários, gerando, assim, dados similares, porém, em grande volume.

O sistema proposto foi escolhido de forma a facilitar a abstração do leitor, devido ao fato de ser um sistema de interação simples, porém com necessidades computacionais suficientes para aplicação do estudo proposto [16].

Como proposta de avaliação de resultados, foi realizado um *benchmarking* sobre uma aplicação do *software* desenvolvido, que serviu como base para a modelagem do banco de dados e para implementação de algoritmos das aplicações servidoras.

Foram desenvolvidas duas aplicações: uma seguindo o modelo apresentado na *Seção 4*, utilizando Node.JS e o MongoDB, enquanto a outra, com o propósito de comparação, foi construída utilizando o MySQL, com banco de dados relacional, a linguagem PHP e um servidor HTTP Apache. A junção MySQL, PHP e HTTP Apache foram escolhidas por serem tecnologias em constante uso em sistemas de informação web.

A seguir serão apresentados os detalhes do desenvolvimento do modelo proposto.

#### *A. Abstração da aplicação para modelagem não-relacional*

Como base da modelagem não-relacional foi desenvolvido um modelo entidade-relacionamento e realizada uma análise para extrair os possíveis elementos globais deste modelo. Um elemento global é irreduzível e contém qualquer outro elemento.

No sistema proposto pode-se identificar como classes do problema: *professor*, *disciplina*, *curso*, *alunos*, *materiais* e *turmas*. Porém, sem a perda de generalidade, não pode ser considerado qualquer elemento diferente de *curso* como elemento global, já que, nele está contido todos os *alunos*, *disciplinas*, *turmas*, *professores* e *materiais*.

Para o modelo relacional de banco de dados, a abstração anterior é satisfatória, todavia, no modelo não relacional, um elemento global será visto como uma coleção e, ao definir

apenas uma coleção, *curso*, sobrecarrega-se o modelo e a coleção, o que acarreta em desempenho insatisfatório no acesso aos dados.

De forma a não sobrecarregar a coleção, identifica-se que um outro elemento global está relacionado à documentos, que estão diretamente relacionados a *alunos* e *materiais*, no caso, outras duas coleções são identificadas e auxiliam na divisão do modelo.

A proposta da análise é visualizar, a partir das coleções, a quantidade de tabelas próxima à quantidade de coleções, na intenção de otimizar, de forma organizacional, um sistema baseado em armazenamento não-relacional, por meio de uma visão comum do modelo relacional. Porém, deve-se distinguir os modelos, levando em consideração a arquitetura do negócio, especificamente as entidades e atributos.

#### B. Implementação lógica

O Node.JS é responsável por carregar todas as dependências da aplicação, como módulos, rotas e modelos, além de iniciar o servidor HTTP e estabelecer a conexão com o MongoDB.

Para que uma requisição seja realizada é necessário que uma rota seja criada. Cada coleção possui suas próprias rotas, em que são definidos os diversos caminhos a serem acessados, com suas respectivas operações.

Um modelo na aplicação Node.JS tem comportamento semelhante ao modelo da aplicação em PHP, entretanto, a primeira responsabilidade de um modelo é definir a estrutura no MongoDB (com seus respectivos tipos), que será armazenado na coleção, diferente da aplicação em PHP, que não tem a responsabilidade de atribuir diretamente a modelagem do banco de dados. Além de definir a estrutura do documento, o modelo define os métodos para fazer as operações de manipulação dos dados no MongoDB.

O fluxo da aplicação desenvolvida em Node.JS se resume em carregar módulos, rotas e modelos. A rota é responsável por receber a requisição e chamar o método do modelo. O modelo, por sua vez, define a estrutura do documento e executa as operações para manipulação dos dados, retornando para o cliente o resultado de forma assíncrona.

## VI. TESTES E RESULTADOS

Com o objetivo de comparar as duas aplicações desenvolvidas: Node.JS com MongoDB e PHP com MySQL, foram estabelecidas as seguintes operações para realização dos testes:

- Inserção de grande volume de dados (250 mil inserções por tabela/coleção);
- Recuperação de dados utilizando identificador aleatório;
- Remoção de dados utilizando identificador aleatório.

Os testes foram realizados em ambiente de desenvolvimento, utilizando um computador com as seguintes configurações: processador 2,3 GHz Intel Core i5, memória: 4 GB 1333 MHz DDR3, 256GB SSD.

Para análise dos testes, definiu-se duas métricas: a primeira refere-se ao tempo de resposta da execução de um *loop*, que executa a chamada ou requisição para a operação a ser testada, enquanto que a segunda refere-se ao tempo de resposta das requisições concorrentes, utilizando o *software* Apache Bench (ferramenta de linha de comando para realização de *benchmarks*, para medir o desempenho de requisições HTTP em uma aplicação web).

#### A. Tempo de resposta

A primeira análise métrica, tempo de resposta, foi realizada utilizando recursos da linguagem de programação, para recuperar o tempo inicial e final de um *loop*.

Foram coletados os tempos de processamento (em segundos), o uso de CPU e o consumo de memória durante cada execução. No Gráfico 1 é possível visualizar os resultados obtidos e pode-se concluir que a aplicação desenvolvida com Node.JS e MongoDB tem desempenho, em média, 60% superior àquela desenvolvida com PHP e MySQL. Mesmo quando feita pelo PHP, para a aplicação em Node.JS, o tempo de execução da operação é menor do que aquele obtido na aplicação em PHP e MySQL.

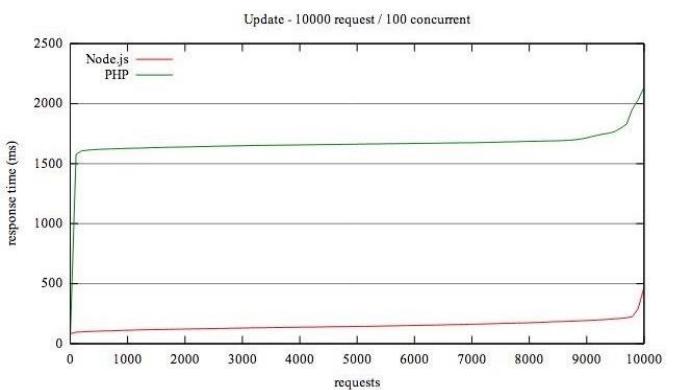


Gráfico 1 - Tempo de resposta comparativo entre aplicações

#### B. Requisições concorrentes

O teste de requisições concorrentes tem como base a utilização da ferramenta de *benchmark*, Apache Bench. Foram efetuadas as mesmas operações da primeira análise métrica, apresentada anteriormente. Para este caso, a variação do teste é quanto a simulação de requisições concorrentes.

Os valores utilizados no Apache Bench para os parâmetros de número de requisições e número de concorrência foram 10.000 (dez mil) e 100 (cem), respectivamente. A requisição de inserção, atualização e seleção podem ser visualizadas no Gráfico 2, Gráfico 3 e Gráfico 4, respectivamente.



Gráfico 2 - Análise de inserções concorrente simuladas

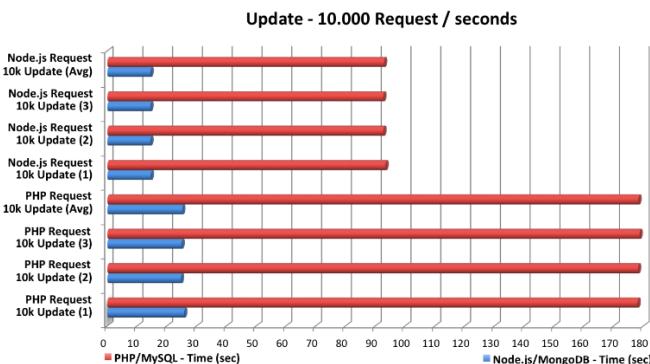


Gráfico 3 - Análise de atualizações concorrente simuladas

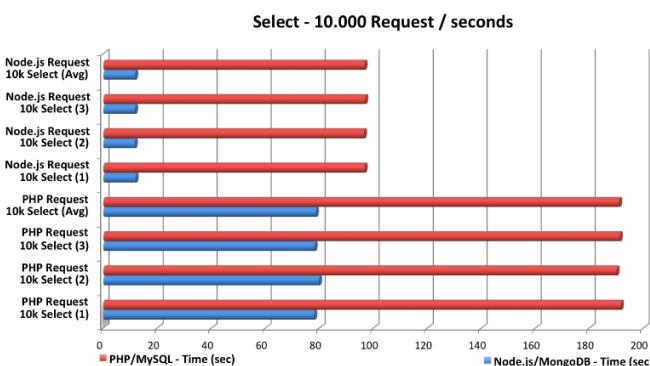


Gráfico 4 - Análise de seleções concorrente simuladas

Conforme se observa, o baixo tempo de resposta do Node.JS é proporcionado pelas requisições assíncronas, o que permite receber um número maior de requisições em relação a uma aplicação em PHP que, por sua vez, necessita esperar o retorno do resultado de uma requisição para receber outra.

Um aspecto importante a ser considerado é que o Node.JS suportou o aumento do valor do parâmetro referente ao número de requisições concorrentes, mantendo a escalabilidade, enquanto que a aplicação em PHP obteve estouro de memória durante a execução das operações referentes à manipulação do banco de dados.

## VII. CONSIDERAÇÕES FINAIS

A implementação de ambas as aplicações utilizando linguagens e bancos de dados distintos é um importante mecanismo para o estabelecimento de comparativos técnicos e teóricos.

Com este trabalho verificou-se que um dos fatores elementares para a redução do tempo de resposta em sistemas web é o fato do Node.JS trabalhar com requisições assíncronas. Nesse sentido, não é necessário a espera da execução das operações e a entrega da resposta ao cliente para execução de uma nova requisição.

Em relação ao banco de dados não-relacional, um ganho é a flexibilidade de estrutura de documentos, facilitando o sistema web tornar-se adaptável à mudanças, fato que não ocorre em um modelo relacional. Outro fator importante em banco de dados não-relacional é a capacidade de recuperar maior quantidade de dados em menor tempo, devido à ausência de operações de junção de conjuntos.

Assim, conclui-se que o modelo proposto para desenvolvimento de sistemas web, utilizando Node.JS e programação orientada a eventos, em conjunto com banco de dados não-relacional MongoDB, apresenta tempo de resposta de seis à oito vezes mais rápido do que o modelo comparado, podendo ser utilizado em ambientes de produção, com ganho de desempenho e escalabilidade.

Para pesquisas futuras, pretende-se aplicar o conceito de *sharding* e *clusterização* do Node.JS em banco de dados escaláveis, o que torna possível testes com parâmetros altos de requisição e concorrência. Possibilitando simulações cada vez mais próximas de situações reais, com grandes volumes de dados e alto número de requisições simultâneas.

## REFERÊNCIAS BIBLIOGRÁFICA

- NETCRAFT, "Netcraft: Internet Research 2012". Disponível em: <<http://goo.gl/2sWqsH>>, Acessado em 04 de Fevereiro de 2015.
- NETCRAFT, "Netcraft: Internet Research 2015". Disponível em: <<http://goo.gl/hvXqlv>>, Acessado em 04 de Fevereiro de 2015.
- J. Correia, A. Aguiar e C. Pinto, "Cloud computing E-mail systems: Analysis of the current situation in SMEs in Portugal," Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on Information Systems and Technologies, Junho 2014.
- Cruz-Benito et al, 2014.
- J. Cruz-Benito, R. Theorón, F.J. García-Péñalvo, C. Maderuelo, J.S. Pérez-Blanco, H. Zazo e A. Martín-Suárez, "Monitoring and feedback of learning processes in virtual worlds through analytics architectures: A real case," Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on Information Systems and Technologies, Junho 2014.
- D. Kegel, "The C10k problem". Disponível em: <<http://www.kegel.com/c10k.html>>. Acessado em 04 de Fevereiro de 2015.
- F.V. Leitner, "Scalable network programming". Disponível em: <<http://bulk.fefe.de/scalable-networking.pdf>>. Acessado em 04 de Fevereiro de 2015.
- M. Welsh, D. Culler e E. Brewer, "SEDA: An Architecture for Highly Concurrent Server Applications", ACM SIGOPS European Workshop, v. 35, n. 5, p. 230 - 243, Dezembro 2001.
- M. Welsh, D. Culler e E. Brewer, "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services", Eighteenth Symposium on Operating Systems Principles (SOSP-18), Banff, Canada, Outubro, 2001.
- M. Cantelon, M. Harter, T.J. Holowaychuk e N. Rajlich. "Node.JS in action", 416 p., Manning, Outubro, 2013.
- R.W. Brito, "Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa". In: InfoBrasil, vol. 3, Fotoleza, Brasil, 2010.

12. A. Silberschatz, P. Baer Galvin, G. Gagne. 2008. "Operating System Concepts", 8th ed, Wiley Publishing.
13. MongoDB, "Legacy Driver Implementation Documentation – BSON" 2015. Disponível em: <<http://goo.gl/oA6Q3c>>. Acessado em 04 de Fevereiro de 2015.
14. M. Welsh, D. Culler e E. Brewer, "SEDA: An Architecture for Highly Concurrent Server Applications", ACM SIGOPS European Workshop, v. 35, n. 5, p. 230 - 243, Dezembro 2001.
15. D.J. Abadi, "Query Execution in Column-Oriented Database Systems", 148 p. Thesis (Doctor of Philosophy in Computer Science and Engineering) – Massachusetts Institute of Technology, Massachusetts, USA, 2008.