Estudo Comparativo de Banco de Dados Chave-Valor com Armazenamento em Memória

Dinei A. Rockenbach[1], **Nadine Anderle**[1], **Dalvan Griebler**[1,2], **Samuel Souza**[1]

[1] Laboratório de Pesquisas Avançadas para Computação em Nuvem (LARCC)
Faculdade Três de Maio (SETREM) – Três de Maio – RS – Brasil

[2]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS/PPGCC)
Porto Alegre – RS – Brasil

{dineiar,nadianderle}@gmail.com,dalvan.griebler@acad.pucrs.br,
samuel@samuelsouza.com

Abstract. *Key-value databases emerge to address relational databases' limitations and with the increasing capacity of RAM memory it is possible to offer greater performance and versatility in data storage and processing. The objective is to perform a comparative study of key-value databases with memory storage Redis, Memcached, Voldemort, Aerospike, Hazelcast and Riak KV. Thus, the work contributed to an analysis of different databases and with results that qualitatively demonstrated the characteristics and pointed out the main advantages.*

Resumo. *Bancos de Dados (BD) chave-valor surgem para suprir limitações de BDs relacionais e com o aumento da capacidade das memórias RAM é possível oferecer maior desempenho e versatilidade no armazenamento e processamento dos dados. O objetivo é realizar um estudo comparativo dos BDs chave-valor com armazenamento em memória Redis, Memcached, Voldemort, Aerospike, Hazelcast e Riak KV. Assim, o trabalho contribuiu para uma análise de diferentes BDs e com resultados que demonstraram qualitativamente as características e apontaram as principais vantagens.*

1. Introdução

As necessidades de alta disponibilidade e velocidade, bem como o aumento e consumo de dados nos sistemas e aplicações atuais, influenciaram no desenvolvimento de novas formas para o armazenamento de dados. Estas vão além dos bancos de dados relacionais, impulsionando o movimento NoSQL (*Not only SQL*) [Fowler 2015]. Não obstante, com a popularidade em alta em um mercado sem um líder estabelecido, há uma consequente explosão no número de sistemas de armazenamento disponíveis, o que dificulta a tomada de decisão quanto à opção que melhor supre as necessidades organizacionais.

Com o objetivo de solucionar problemas específicos, os bancos NoSQL foram categorizados de acordo com suas características e otimizações. Por exemplo, a Amazon utiliza seu sistema chave-valor (*key-value*) Dynamo [DeCandia et al. 2007] para gerenciar as listas de mais vendidos, carrinhos de compras, preferências do consumidor, gerenciamento de produtos, entre outras aplicações. Exitem também sistemas de família de colunas (*column family* ou *columnar*) que foram influenciados pelo Bigtable da Google [Chang et al. 2008]. Enquanto isso, os assim chamados de sistemas de documentos (*document*) resultaram, por exemplo, no MongoDB[1]. Por fim, do chamado banco triplo

[1]https://www.mongodb.com

(*graph database* ou *triple*), tem-se como exemplo o Neo4j[2]. Cada uma destas categorias traz sistemas que cobrem diferentes limitações dos bancos relacionais tradicionais.

O foco deste trabalho está nos bancos de dados chave-valor com armazenamento em memória, a fim de realizar um estudo comparativo e qualitativo de banco de dados ainda pouco estudados na literatura (Seção 2). Este tipo de banco de dados NoSQL é o representante com as estruturas mais simples dentre os existentes [Pokorny 2013]. No entanto, cabe ressaltar que eles possuem um amplo espectro de casos de uso, sendo largamente adotados tanto como armazenamento primário quanto para auxiliar outros sistemas de armazenamento [Carlson 2013]. Ainda, o aumento na capacidade das memórias RAM fez com estes sistemas se adaptassem para efetuar o armazenamento e processamento de dados em memória, com a finalidade de melhorar o desempenho [Zhang et al. 2015]. A realidade corrobora a afirmação de Jim Gray de que memória é o novo disco e o disco é a nova fita [Robbins 2008], o que justifica o aumento da popularidade destes sistemas.

Este artigo está organizado em 5 seções, incluindo esta seção introdutória. Na Seção 2 estão os trabalhos relacionados. A Seção 3 traz o embasamento sobre os bancos de dados pesquisados. Na Seção 4 está detalhado o estudo comparativo destes sistemas. Por fim, na Seção 5 estão as conclusões e propostas para trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção é apresentada uma discussão sobre os trabalhos relacionados publicados recentemente na literatura. De forma semelhante, todos os trabalhos escolhidos buscam caracterizar e comparar bancos de dados NoSQL. No entanto, não voltam os estudos para um determinado tipo de banco de dados, o que é importante para avaliar características específicas. Tanto [Hecht and Jablonski 2011] quanto [Han et al. 2011] se propõem a fazer um estudo e avaliação dos bancos de dados NoSQL, com o mesmo objetivo principal: prover informações para auxiliar na escolha do banco NoSQL que melhor atende às necessidades. Por não delimitarem um tipo específico de banco NoSQL, ambos possuem um escopo mais abrangente do que o presente trabalho. No ponto de intersecção entre este trabalho e os citados, [Hecht and Jablonski 2011] inclui em seu trabalho os bancos Project Voldemort, Redis e Membase, enquanto que [Han et al. 2011] avalia Redis, Tokyo Cabinet-Tokyo Tyrant e Flare.

Em [Deka 2014] é apresentada uma visão geral de vários sistemas NoSQL e os representantes chave-valor incluídos na avaliação são Hypertable, Voldemort, Dynomite, Redis e Dynamo. Nota-se a falta, porém, de uma visão comparativa mais clara sobre aspectos de garantias de durabilidade, disponibilidade, protocolos suportados, e outras informações que podem vir a ter uma influência significativa na escolha de um banco de dados chave-valor. Já [Zhang et al. 2015] traz uma visão bem estruturada dos objetivos que nortearam o projeto de cada um dos sistemas descritos no trabalho, o qual foca em sistemas com gerenciamento e processamento de dados em memória. Dentre os sistemas estudados, os representantes dos bancos chave-valor são MemepiC, RAMCloud, Redis, Memcached, MemC3 e TxCache. Dos sistemas, o trabalho descreve as cargas de dados mais adequadas ao sistema, a estratégia para construção de índices, o controle de concorrência, tolerância a falhas, tratamentos para conjuntos de dados maiores do que a memória disponível e o suporte a consultas personalizadas em baixo nível (como *stored procedures* e *scripts* em linguagem nativa, por exemplo), porém com pouca abordagem de alto nível que auxilie na escolha de um sistema em favor de outro.

[2]https://neo4j.com

Ainda que o tema NoSQL tenha sido bastante explorado na academia, e a bibliografia focada no armazenamento de dados em memória tenha crescido muito nos últimos anos, nota-se a falta de um estudo comparativo entre os sistemas chave-valor em memória Redis, Memcached, Voldemort, Aerospike, Hazelcast e Riak KV.

3. Banco de Dados com Armazenamento em Memória

O crescimento dos bancos de dados com armazenamento de dados em memória (IMDB ou *in-memory databases*) segue uma tendência que teve seu início no hardware, com a capacidade da memória dobrando em média a cada três anos e seu preço caindo uma casa decimal a cada cinco anos [Zhang et al. 2015]. As memórias não-voláteis (NVM ou *Non-Volatile Memory*) como o SSD (*Solid State Disk*), também têm evoluído, mas seu custo [Kasavajhala 2011], durabilidade e confiabilidade [Schroeder et al. 2016] continuam sendo impeditivos para a maioria das aplicações.

A vantagem em manter os dados na memória ao invés do disco está relacionada à latência de acesso a estes dados, pois remove-se a necessidade de acessar a camada mais lenta da hierarquia de memória, conforme demonstrado pela Figura 1 (adaptada de [Zhang et al. 2015]), que detalha as camadas de armazenamento, bem como uma estimativa de sua capacidade atual e da latência de acesso aos dados nela armazenados.

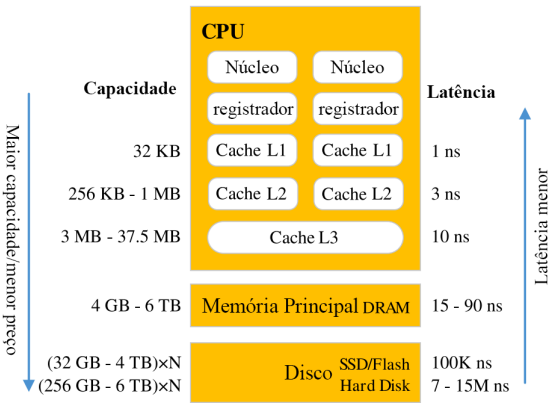

Figura 1. Hierarquia de memória.

Para que os dados possam ser processados pela CPU é necessário que estes estejam nos registradores e para tal é preciso que estes dados passem por todas as camadas da hierarquia de memória até chegarem aos registradores [Zhang et al. 2015]. Como pode ser visto na Figura 1, o disco é a camada mais distante e mais lenta, porém com a maior capacidade de armazenamento. Com o aumento da capacidade da camada de memória principal (composta pela memória RAM) os IMDB buscam trazer os dados para esta camada e evitar o nível mais lento da hierarquia de memória.

Dentre a miríade de bancos de dados com armazenamento em memória, os bancos chave-valor podem ser considerados os representantes mais versáteis, simples e com melhor desempenho, advindo principalmente da sua simplicidade [Pokorny 2013]. Portanto, muitos sistemas desta categoria sacrificam garantias de consistência em favor do desempenho [DeCandia et al. 2007] [Fowler 2015]. Nestes bancos, cada valor armazenado está vinculado a uma chave que identifica unicamente um valor [Han et al. 2011], sendo que este valor pode ser tanto um conteúdo binário quando uma estrutura de dados complexa, conforme as funcionalidades oferecidas pelo banco [Pokorny 2013]. Nas pró-

ximas seções são apresentados e discutidos os sistemas de armazenamento chave-valor em memória Redis, Memcached, Voldemort, Aerospike, Hazelcast e Riak KV.

3.1. Redis

Redis (*REmote DIctionary Server*) [Redis 2009] é um sistema de armazenamento de dados estruturados em memória que pode ser utilizado como banco de dados, *cache* e *message broker* [Cao et al. 2016]. Ele opera em um modelo cliente-servidor através de conexões TCP utilizando um protocolo próprio chamado RESP (REdis Serialization Protocol).

O modelo de dados do Redis é composto por cinco estruturas de dados diferentes para os valores (*string*, *list*, *set*, *sorted set* e *hash*), a persistência dos dados da memória em disco através de dois métodos (*snapshopts* chamados RDB e *append-only file* ou AOF) [Zhang et al. 2015]. A possibilidade de escalabilidade horizontal através do Redis Cluster foi adicionada apenas em 2015, na versão 3.0 do sistema. Segundo os autores de [Sanfilippo 2010], um sistema deve ser eficiente em um único nó quando for escalado.

3.2. Memcached

O Memcached [Memcached 2003] caracteriza-se como um sistema genérico de *cache* em memória. Ele foi construído pensando na melhoria de desempenho de aplicações *web* através da redução na demanda de requisições ao banco de dados em disco. Brad Fitzpatrick desenvolveu ele para melhorar o desempenho do site Livejournal.com através de uma solução melhor de *cache* [Galbraith 2009]. A sua implementação é na linguagem Perl e posteriormente reescrito em C. O Memcached utiliza uma arquitetura *multi-thread* e o controle de concorrência interno é feito através de uma *hash-table* estática de *locks* [Zhang et al. 2015].

A classificação do Memcached como banco de dados é discutível, uma vez que o mesmo não implementa persistência, failover [Galbraith 2009] nem escalabilidade horizontal, pois a distribuição dos dados entre múltiplas instâncias do sistema deve ser feita pelo cliente [Zhang et al. 2015]. O funcionamento do Memcached segue o modelo cliente-servidor e a comunicação ocorre através de conexões TCP ou UDP utilizando um protocolo próprio que suporta textos puros em ASCII ou dados binários [Soliman 2013].

3.3. Voldemort

O Voldemort [Voldemort 2009] foi desenvolvido pelo LinkedIn em linguagem Java com o objetivo de gerenciar funcionalidades dependentes de associações entre dados da rede social, tais como a recomendação de relacionamentos através da análise dos relacionamentos atuais [Sumbaly et al. 2012].

O Voldemort é inspirado no Dynamo, da Amazon [DeCandia et al. 2007], oferece comandos simples (put, get e delete) [Sumbaly et al. 2013] e uma arquitetura completamente distribuída, onde cada nó é independente e não existe um servidor principal de coordenação [Deka 2014]. O sistema é completamente modularizado e tanto a serialização dos dados quanto a persistência são oferecidas através de módulos plugáveis. Segundo [Sumbaly et al. 2012], a grande vantagem do Voldemort em relação ao Dynamo, é um mecanismo próprio de armazenamento desenhado para o pré-carregamento de grandes volumes de dados, em que o Voldemort passa a funcionar em modo somente leitura.

3.4. Aerospike

O Aerospike [Aerospike 2012] tem uma arquitetura modelada com foco em velocidade na análise de dados, escalabilidade e confiabilidade para aplicações *web*. Esse banco de

dados se apresenta como uma solução para a combinação de diferentes tipos de dados e também acessos por milhares de usuários. Pensando nisso, as suas operações são focadas em chave-valor e otimizadas para o uso da memória RAM em conjunto com memórias *flash* (NVM) [Aerospike 2012].

Diferente de vários de seus concorrentes, o próprio Aerospike disponibiliza bibliotecas de integração aos clientes, a fim de melhorar o desempenho na sua utilização. Quanto ao *cluster*, todos os nós são iguais, em uma arquitetura conhecida como *shared nothing*. A respeito do *server* é possível utilizar índices secundários e definir funções para otimizar a utilização dos dados. E por fim, a camada de armazenamento incorpora a utilização da memória RAM e de sistemas de armazenamento permanente.

3.5. Hazelcast

O Hazelcast [Hazelcast 2009] é uma ferramenta distribuída sob licença *open source* e comercial desenvolvida em Java. Possui seu foco em computação distribuída e escalabilidade horizontal, se destacando dos concorrentes por oferecer as garantias ACID (Atomicidade, Consistência, Isolamento e Durabilidade) dos bancos de dados relacionais tradicionais.

O *cluster* funciona em uma arquitetura *shared nothing*, onde não existe um ponto único de falha. Além de oferecer clientes para as linguagens comuns como Java, C, C++ e C#, o Hazelcast oferece uma API REST, está preparado para trabalhar com o protocolo de comunicação do Memcache e pode ser utilizado através do Hibernate [Hazelcast 2009].

3.6. Riak KV

O Riak KV [Basho 2009] possui como principal objetivo oferecer disponibilidade máxima, com escalabilidade horizontal em forma de *cluster*, sendo considerado um banco de dados de simples operação e fácil escalabilidade. Em sua versão comercial há suporte a *multi-cluster replication*, ou seja, é possível realizar a replicação de dados através de diferentes *clusters*, geograficamente distantes, a fim de reduzir a latência de acesso uniformemente para clientes através do globo.

Nota-se claramente a influência do Dynamo [DeCandia et al. 2007] no Riak KV, desde suas funcionalidades para execução distribuída até nas configurações do fator de replicação e arquitetura *shared nothing*.

4. Estudo Comparativo

Esta seção apresenta uma comparação entre os sistemas apresentados anteriormente. Para isso, as características foram classificadas em três grupos: (I) características mercadológicas, onde são explorados aspectos sem relação direta com funcionalidades ou o funcionamento do banco, tais como ano de lançamento, licenciamento e linguagem de desenvolvimento; (II) características do projeto, onde são descritas definições decididas no projeto do sistema, tais como escalabilidade, disponibilidade e consistência; e (III) características de manutenção, onde são explanadas os aspectos que tem relação direta com a manutenção e suporte ao sistema, tais como ferramentas internas para monitoramento e interface de gerenciamento.

Na Tabela 1 é possível avaliar: o ano em que a primeira versão do sistema foi lançada, os licenciamentos sob os quais o *software* é distribuído, a linguagem na qual o sistema foi desenvolvido, os sistemas operacionais suportados, as linguagens nas quais são oferecidos clientes para comunicação e os protocolos de comunicação suportados. A

partir dos dados disponibilizados, os interessados podem avaliar a maturidade do sistema, se a licença está alinhada com as necessidades empresariais e se a infraestrutura disponível e o esforço de implementação estão dentro do esperado.

Vale ressaltar que o Redis não suporta oficialmente Windows, mas uma versão para Windows x64 é mantida pela equipe da MS Open Tech (Microsoft Open Technologies). Quanto ao Voldemort e ao Hazelcast, como os mesmos rodam na JVM (Java Virtual Machine), podem-se considerar os sistemas operacionais suportados por esta tecnologia. Tanto Aerospike quanto Riak KV oferecem pacotes para sistemas baseados nas distribuições Linux Red Hat, Debian e Ubuntu. O Aerospike ainda oferece sua execução no OS X e Windows através de máquinas virtuais.

Tabela 1. Características mercadológicas

	Redis	Memcached	Voldemort	Aerospike	Hazelcast	Riak KV
Lançamento	2009	2003	2009	2012	2009	2009
Licenciamento	BSD-3 e comercial	BSD-3	Apache 2	AGPL e comercial	Apache 2 e comercial	Apache 2 e comercial
Desenvolvido	C	C	Java	C	Java	Erlang
SO Suporte	Linux, BSD, OSX e Windows	Debian/Ubuntu e Windows	JVM	Linux, OS X e Windows	JVM	Linux
Clientes	48 linguagens	Não existe listagem oficial	4 linguagens	12 linguagens	6 linguagens	21 linguagens
Protocolos	Próprio (RESP)	Próprio	HTTP, Socket, NIO	Próprio e JDBC	Próprio e Memcached	API HTTP e próprio

Quanto ao item linguagens com cliente, é importante notar que foram considerados apenas as linguagens e clientes listados no site oficial de cada sistema e que o site do Memcached não oferece uma listagem oficial das linguagens suportadas. Nota-se também que as linguagens C++, Java e Python são as únicas para as quais todos os sistemas possuem clientes. Os protocolos de comunicação são o meio de comunicação entre o cliente e o servidor, porém, na maioria dos casos a comunicação é feita através de um dos clientes já construídos e a empresa não precisa se preocupar com o protocolo utilizado pelo cliente.

Na Tabela 2 é possível avaliar: as opções para escalabilidade horizontal (ou *clusterização*), a classificação do sistema segundo o teorema CAP [Brewer 2000], como é feito o controle de concorrência, o suporte à transações ACID, as opções de persistência dos dados em disco, o suporte a dados complexos e as opções para autenticação do cliente. Analisando a Tabela 2 é possível avaliar se as funcionalidades e características do banco de dados atendem às demandas e características referentes aos dados que se pretende armazenar nos mesmos.

Quanto à escalabilidade, todos os bancos exceto o Memcached incluem suporte a *sharding* e replicação, sendo que a maioria (a exemplo do Dynamo [DeCandia et al. 2007]) oferecem fator de replicação configurável para leituras e escritas. O Redis utiliza o modelo master/slave, comumente utilizado nas bases de dados relacionais tradicionais.

O teorema CAP (*Consistency*, *Availability* e *Partition tolerance*) foi proposto por Eric Brewer em [Brewer 2000] e verificado em [Gilbert and Lynch 2002], desde então passou a ser largamente aceito pela academia. O teorema afirma que na existência de uma falha de comunicação (*partition*) cada nó de um sistema distribuído deve escolher entre responder requisições, mantendo a disponibilidade (*availability*) e assumindo o risco de não retornar os dados mais atuais, ou rejeitar requisições para garantir a consistência dos dados (*consistency*). Sistemas classificados como AP priorizam a disponibilidade, en-

quanto que sistemas classificados como CP priorizam a consistência. O teorema tem sido alvo de muitas críticas e Brewer explora algumas de suas limitações em [Brewer 2012], enquanto Abadi propõe o teorema PACELC como alternativa em [Abadi 2012].

Quanto à classificação do Redis, é importante mencionar que o ele não atende todos os requisitos de um sistema CP de [Brewer 2000], por usar replicação assíncrona entre os nós do *cluster*. O teorema CAP também não se aplica ao Memcached pelo fato de ele não suportar a criação de *clusters* e, portanto, não haver comunicação entre nós. O Riak KV possui uma configuração onde é possível definir qual dos atributos devem ser preservados (disponibilidade ou consistência).

Tabela 2. Características do projeto

	Redis	Memcached	Voldemort	Aerospike	Hazelcast	Riak KV
Escalabilidade horizontal	Mestre - Escravo	Não	Fator de Replicação	Fator de Replicação	Fator de Replicação	Fator de Replicação
Teorema CAP	CP	N/A	AP	AP	AP	Config.
Controle Concorrência	Single-thread	Mutex lock	MVCC	Test-and-set	Multi-single-thread	MVCC
Transações	Parcial	Não	Não	Parcial	Sim	Não
Persistência em disco	RDB e AOF	Não	Config.	Assínc.	Banco auxiliar	Banco auxiliar
Suporte a dados complexos	Sim	Não	Sim	Sim	Sim	Sim
Autenticação	Simples	SASL	Kerberos	Somente comercial	Simples, SSL, Kerberos,IP	Sim, e autorização

O controle de concorrência é implementado de diferentes maneiras. No Redis a execução é *single-thread* e as requisições são processadas de forma assíncrona internamente [Zhang et al. 2015], sendo que apenas um *master* responde por uma determinada chave, portanto não há concorrência. O Memcached utiliza *mutex* (*mutual exclusive*) *lock*. Tanto Voldemort quanto Riak KV seguem a implementação do Dynamo [DeCandia et al. 2007] e utilizam *vector clocks*, uma implementação do versionamento baseado em locking otimista conhecida por MVCC (Multi Version Concurrency Control). O Aerospike utiliza o método conhecido como *test-and-set* ou *check-and-set* (CAS), uma operação atômica implementada a baixo nível que escreve em um local de memória e retorna o valor antigo. No Hazelcast, é criada uma *thread* para atender cada uma das partições internas de dados, portanto ainda que ele seja *multi-thread*, uma chave específica está num contexto *single-thread* e, portanto, não há concorrência.

Quanto as transações, há um nível bastante variado de suporte oferecido pelos sistemas. Ainda que o Redis tenha suporte básico a transações, estas não possuem opção de rollback e a durabilidade da mesma depende da persistência em disco. Memcached, Voldemort e Riak KV declaram não suportarem transações, enquanto que o Aerospike suporta transações que envolvam uma única chave ou que sejam somente leitura, no caso de envolverem múltiplas chaves. O Hazelcast se destaca sendo o único a oferecer transações ACID completas.

A persistência em disco é oferecida por todos os bancos, exceto o memcached. No Redis são oferecidas duas formas complementares: *snapshot* (RDB), onde todos os dados na memória são gravados em disco, e *append-only file* (AOF), onde cada operação é gravada em um arquivo de log e o arquivo é reescrito quando chega em um tamanho pré-determinado. O Voldemort oferece opções para configurar a persistência como síncrona (*write through*, onde a operação é persistida antes do retorno ao cliente) ou assíncrona (*write behind*, onde o cliente recebe a confirmação e posteriormente a operação é persistida), enquanto que o Aerospike faz a persistência de forma assíncrona. Vale mencionar

que o Aerospike tem melhorias focadas no uso de SSD como dispositivo de armazenamento permanente. Tanto Hazelcast como Riak KV oferecem persistência através do acoplamento de um banco de dados auxiliar e a assincronicidade é configurável.

Referente ao suporte a dados complexos, vale notar que todos os sistemas, exceto o Memcached, suportam chaves do tipo lista e hashtable (ainda que com nomes diferentes). Hazelcast e Voldemort se baseiam fortemente nas classes do Java, Redis e Riak KV ainda oferecem suporte ao tipo HyperLogLogs, enquanto Redis e Aerospike oferecem suporte a tipagem ou comandos relativos a georreferenciamento.

Finalizando, enquanto que Redis oferece autenticação simples através de credenciais pré-configuradas, o Memcached oferece autenticação através do protocolo SASL (*Simple Authentication and Security Layer*). O Voldemort é integrado ao protocolo Kerberos. O Aerospike oferece autenticação apenas em sua versão com licenciamento comercial. O Hazelcast oferece todos os protocolos supracitados e o SSL. Por último, o Riak KV oferece um sistema próprio de usuários e grupos, com autenticação e autorização baseada em diversos mecanismos, incluindo senhas e certificados digitais.

Na Tabela 3 está descrita a existência de interfaces de gerenciamento, ferramentas de monitoramento e benchmarks para os sistemas estudados. É importante notar que foram avaliadas apenas as ferramentas oficiais dos desenvolvedores dos sistemas. Portanto, muitas destas ferramentas, ainda que não estejam declaradas aqui, já foram desenvolvidas pela comunidade e estão disponíveis. Estas informações são particularmente interessantes para avaliar o esforço de manutenção que será despendido após a implantação do sistema.

Tabela 3. Características de manutenção

	Redis	Memcached	Voldemort	Aerospike	Hazelcast	Riak KV
Interface de Gerenciamento	Não	Não	Básica	À parte	Comercial	Sim
Ferramentas de Monitoramento	'INFO'	'stats'	JMX	'asadm'	JMX	'stats'
Benchmark embutido	Sim	Não	Sim	Sim	Não	Sim

Quanto às interfaces de gerenciamento oferecidas pelos sistemas avaliados, o Redis e o Memcached são os únicos que não as oferecem nativamente (ainda que existam opções na comunidade) e o Voldemort oferece uma interface básica à parte escrita em Ruby, que está sem manutenção. O Aerospike oferece uma interface que deve ser instalada à parte. No Hazelcast, esta funcionalidade está disponível apenas na versão comercial. O Riak KV é o único onde a interface de gerenciamento já está integrada ao código principal do programa, não requerendo nenhuma instalação extra.

A respeito de ferramentas de monitoramento, o Redis oferece comandos como *INFO*, *MEMORY* e *LATENCY*, enquanto que o Memcached oferece o comando *stats* e o Aerospike oferece o comando *asadm*. O Voldemort possui uma interface completa de monitoramento exposta através de *Java Management Extensions* (JMX). Esta é a mesma estratégia utilizada pelo Hazelcast. O Riak KV oferece os comandos *stat* e *stats* em sua interface de linha de comando (CLI) *riak-admin* e a URL */stats* em sua API HTTP.

No item *benchmark* embutido foi avaliado se são disponibilizados *benchmarks* junto com o sistema, cujo principal objetivo é avaliar o desempenho do sistema em determinada infraestrutura. Enquanto que Memcached e Hazelcast não oferecem ferramentas próprias para realização de *benchmark*, no Redis existe a ferramenta *redis-benchmark* e o Voldemort oferece a *voldemort-performance-tool*. No Aerospike os *benchmarks* estão nos clientes disponibilizados e no Riak KV o nome dado ao *benchmark* é *Basho Bench*.

Após comparar as características dos bancos de dados chave-valor com armazenamento em memória, Redis, Memcached, Voldemort, Aerospike, Hazelcast e Riak KV, é fácil entender o motivo pelo qual o Memcached não é considerado um banco de dados, pois seu foco destoa bastante de seus semelhantes. É possível perceber também que, mesmo que o Redis tenha oferecido suporte à clusterização em suas versões mais recentes, os outros sistemas ainda estão à frente quando o assunto é funcionalidades para execução distribuída. É possível perceber também como Voldemort e Hazelcast se utilizam do ecossistema Java para prover funcionalidades interessantes e como o paper do Dynamo [DeCandia et al. 2007] influencia principalmente Voldemort e Riak KV.

5. Conclusões

Após estudar os bancos chave-valor com armazenamento em memória, é possível notar que mesmo um subconjunto específico de bancos NoSQL traz muitas variáveis. Neste sentido, destaca-se a grande quantidade de características que devem ser cuidadosamente avaliadas pelo analista para a correta tomada de decisão quanto ao banco mais adequado às necessidades. Dentre estas características, destacam-se o padrão de busca e gravação de dados da aplicação cliente, a importância da durabilidade dos dados, o comportamento desejado frente a partições no *cluster*, o ambiente de infraestrutura onde a solução será implantada, o ambiente de desenvolvimento da aplicação cliente e as perspectivas de crescimento na demanda da aplicação.

Com as características dos sistemas esclarecidas, percebe-se que outro fator importante para a escolha do banco de dados chave-valor com armazenamento em memória a ser adotado é o desempenho, que é justamente o ponto que traz mais interesse a esta categoria de sistemas. Como trabalho futuro, propõe-se uma avaliação do desempenho dos sistemas aqui estudados, comparando o desempenho das variadas características compartilhadas pelos mesmos.

Referências

[Abadi 2012] Abadi, D. (2012). Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. *Computer*, 45(2):37–42.

[Aerospike 2012] Aerospike (2012). Aerospike | High Performance NoSQL Database. Access on <http://www.aerospike.com/>.

[Basho 2009] Basho (2009). Riak KV. Access on <http://basho.com/products/riak-kv/>.

[Brewer 2000] Brewer, E. (2000). Towards Robust Distributed Systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, pages 7–, New York, NY, USA. ACM.

[Brewer 2012] Brewer, E. (2012). CAP twelve years later: How the "rules"have changed. *Computer*, 45(2):23–29.

[Cao et al. 2016] Cao, W., Sahin, S., Liu, L., and Bao, X. (2016). Evaluation and Analysis of In-Memory Key-Value Systems. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 26–33.

[Carlson 2013] Carlson, J. L. (2013). *Redis in Action*. Manning, Shelter Island, NY, USA.

[Chang et al. 2008] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Trans. on Computer Systems (TOCS)*, 26(2):4.

[DeCandia et al. 2007] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220.

[Deka 2014] Deka, G. C. (2014). A survey of cloud database systems. *IT Professional*, 16(2):50–57.

[Fowler 2015] Fowler, A. (2015). *NoSQL For Dummies*. John Wiley & Sons, 111 River Street, Hoboken, New Jersey, USA.

[Galbraith 2009] Galbraith, P. (2009). *Developing Web Applications with Apache, MySQL, memcached, and Perl*. John Wiley & Sons.

[Gilbert and Lynch 2002] Gilbert, S. and Lynch, N. (2002). Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *SIGACT News*, 33(2):51–59.

[Han et al. 2011] Han, J., E, H., Le, G., and Du, J. (2011). Survey on NoSQL database. In *2011 6th International Conference on Pervasive Computing and Applications*, pages 363–366.

[Hazelcast 2009] Hazelcast (2009). Hazelcast the Leading In-Memory Data Grid - Hazelcast.com. Access on <https://hazelcast.com/>.

[Hecht and Jablonski 2011] Hecht, R. and Jablonski, S. (2011). NoSQL evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing (CSC)*, pages 336–341.

[Kasavajhala 2011] Kasavajhala, V. (2011). Solid State Drive vs. Hard Disk Drive Price and Performance Study. *Proc. Dell Technical White Paper*, pages 8–9.

[Memcached 2003] Memcached (2003). memcached - a distributed memory object caching system. Access on <https://memcached.org/>.

[Pokorny 2013] Pokorny, J. (2013). NoSQL databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, 9(1):69–82.

[Redis 2009] Redis (2009). Redis.io. Access on <http://redis.io/>.

[Robbins 2008] Robbins, S. (2008). RAM is the new disk... Access on <https://www.infoq.com/news/2008/06/ram-is-disk>.

[Sanfilippo 2010] Sanfilippo, S. (2010). On Redis, Memcached, Speed, Benchmarks and The Toilet . Access on <http://antirez.com/post/redis-memcached-benchmark.html>.

[Schroeder et al. 2016] Schroeder, B., Lagisetty, R., and Merchant, A. (2016). Flash Reliability in Production: The Expected and the Unexpected. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16)*, FAST '16, pages 67–80, Santa Clara, CA, USA.

[Soliman 2013] Soliman, A. (2013). *Getting Started with Memcached*. Packt.

[Sumbaly et al. 2012] Sumbaly, R., Kreps, J., Gao, L., Feinberg, A., Soman, C., and Shah, S. (2012). Serving large-scale batch computed data with Project Voldemort. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, page 18.

[Sumbaly et al. 2013] Sumbaly, R., Kreps, J., and Shah, S. (2013). The Big Data Ecosystem at LinkedIn. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1125–1134, New York, NY, USA. ACM.

[Voldemort 2009] Voldemort (2009). Project Voldemort. Access on <http://www.project-voldemort.com/>.

[Zhang et al. 2015] Zhang, H., Chen, G., Ooi, B. C., Tan, K.-L., and Zhang, M. (2015). In-Memory Big Data Management and Processing: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1920–1948.