Achieving Automated Elasticity on NoSQL Document-Store Clusters

A Simple Protocol for Resizing MongoDB Clusters

Manousos Papadosifos, Konstantinos Tsakogiannis, Athanasios-Christos Typaldos
School of Electrical and Computer Engineering, National Technical University of Athens, Greece
Email: {el17150, el17205, el17115}@mail.ntua.gr

***Abstract*—This work presents the process of implementing an extension to MongoDB's native elasticity capabilities; mainly sharding. The extension aims to provide a simple mechanism for dynamically resizing a cluster horizontally, based on an empirical, user-defined policy, whose main goal is to ensure, the functional requirements of the cluster are met using the minimum computational resources possible. Such mechanism requires having real-time awareness of the cluster's state in regards to the workloads it undertakes as well as the resulting resource usage. To enable the cluster to evaluate and adapt accordingly, thus adding or removing a mongo shard, specialized open-source tools are utilized and orchestrated to work in synergy, resulting in -almost- real-time, dynamic horizontal resizing of the cluster.**

***Index Terms*—Cloud Resource Provisioning; NoSQL; Elasticity; Protocol; Distributed Datastores; Open-Source; MongoDB; YCSB; Ganglia; Python;**

I. Introduction

Elasticity is defined as the ability to expand or contract the available resources of a system, platform or application in accordance with a defined policy. The cloud service platforms (IaaS) such as Amazon EC2 or Open source alternatives (e.g., OpenStack) are inherently flexible [1], allowing applications to adjust resources on demand. Using this mechanism, many NoSQL implementations are able to adjust their performance: query rendering, data entry, etc., depending on the amount of resources available (e.g., the number of nodes in the NoSQL cluster). In this project, we were called upon to implement one mechanism for automatically applying elasticity to a MongoDB document-store cluster. This resulted into implementing one simple protocol where, after -almost- real-time analysis of cluster performance data, via the monitoring platform Ganglia, it is decided whether to extend or reduce the cluster's sharding.

The development of the framework described above was decided to be done as a well defined series of steps; although for the development of every individual component of these series, an agile methodology was applied. Consequently, the overall development ended up being an encapsulation of the agile methodology inside each step of a waterfall scheme. More specifically, the separate components are listed as bellow:

- A cloud-hosted cluster consisting of 3 Virtual Machines (VMs), used as a deployment platform for the framework,
- The establishment of a MongoDB topology upon the 3 VMs,
- The deployment of the monitoring platform Ganglia accompanied by it's MongoDB extension,
- An automated procedure for adding or removing mongo shards in a robust and consistent manner,
- A rendition of the YCSB benchmarking tool, used as a vessel to force a synthetic workload on the DB, capable of triggering multiple scenarios for the protocol to confront and last,
- A configurable, user-defined policy to evaluate the cluster's state at a given time and decide whether the deployment or redeployment of mongo shards is needed, or just simply keep the state of the cluster as is.

The resulting synergy of the components above, yields a complete, proof-of-concept framework that demonstrates automated MongoDB elasticity.

II. Setup

Below lies a thorough analysis of how each individual component of the framework is being set up, aiming at facilitating experimentation and fine tuning of the protocol that would follow; eventually giving it it's final form and constitute the final product together with the components below.

A. Cluster

The main component for being able to deploy and experiment with the framework is a cluster of computers. For this purpose, 3 VMs were given by the supervisors. More specifically, the VMs were hosted by GRNET's okeanos IaaS, given a total of 12 virtual CPUs, 24 GB of RAM, 90 GB of hard disk storage, all equally divided between the 3 VMs, resulting in 3 equivalent nodes, from now on referred to as BigData{1,2,3}, respectively. With BigData1 having the IPv4 interface, it was selected as the contact terminal for the whole cluster. Each node runs Ubuntu 20.04.3 LTS, supporting Python v2.7 and v3.8 and Java v11.0.13, which are needed for the development process.

B. DBMS

The database management system used is MongoDB v5.0.5, a NoSQL document store. MongoDB supports horizontal scaling via a mechanism called sharding. Three major participants are required in order to operate a sharded MongoDB cluster. Lowest in the hierarchy are the shards, which consist of replica sets of distributed mongod processes and store a part of the actual data. Each replica set (or shard) has one primary node and many secondary nodes, where data is replicated granting availability and redundancy. Required next is the configuration server (or a replica set of them), which stores metadata about the distribution of data to shards. Last a query router named mongos (or a replica set of them) is used for answering client queries against a collection of the sharded database. Utilizing the configuration server, they find and perform queries upon the data residing in the corresponding shards. Our setup is as follows: on node BigData1 reside one mongos query router and one configuration server. The other two nodes, BigData2 and BigData3 hold each a number of different shards (each one is a single member replica set of mongod processes). Each shard is added to the node with the lowest number of shards or removed from it dynamically, as the protocol progresses. The naming convention used for the shards is "shardr<replicareplica_set_number>s<server_number>". For example "shardr2s3" implies that the referenced server (or actually the mongod process) is the third member of the second replica set, thus is a member of the second shard of the cluster. For the experiments discussed below, the replica sets used have only one member since data replication is not of our primary concern.

C. Resource Supervision

In order to supervise our cluster of VMs and mongod processes we utilized the Ganglia monitoring tool [5]. We compiled from source the latest version available, v3.7.2, including the latest web tool, Ganglia web v3.7.2. Also apache v2.4.41 and php v7.2.34 needed for the web interface of ganglia. Our Ganglia cluster consists of one gmetad residing on node BigData1, responsible for polling gmond processes, aggregating the resulting metrics, storing them locally and updating the web interface. One gmond process runs in every VM with total of three. Each gmond is responsible of collecting metrics, utilizing plugin or built-in modules and sending them to other gmonds. In our cluster the two gmonds from nodes BigData2 and BigData3 collect and send their metrics to gmond in BigData1, which is later polled from gmetad, collecting the latter this way all the metrics of the cluster [6]. As far as metrics from MongoDB shards are concerned, we utilized and modified according to our needs the open source ganglia plugin found in [12]. So the metrics that a gmond in either nodes BigData2 or BigData3, collects and sends to gmond in BigData1 contain vm-wide metrics, such cpu or load usage and metrics referring to every residing mongod process, which is practically a shard member (belongs to shard's replica set).

D. Synthetic Workload

At a stage in which Ganglia was established as the resource supervision tool, the question of what is the best way for a load to be given to the cluster emerged. After a short amount of research, it emerged naturally that a benchmarking tool was the optimal choice, as such tool provides an easy-to-use interface, well defined and configurable workloads and as a bonus, it outputs an oversupply of valuable metrics, useful both for the experimentation stage and potential utilization within the protocol itself. The choice of the benchmark, was trivial as well. YCSB is the de facto benchmarking tool for NoSQL DBMSs [7], with the added benefit of natively supporting MongoDB. In the context of the project version 0.17 of YCSB was used. The following table presents the built-in workloads YCSB offers, with the ability of tweaking those or defining new ones from ground up.

TABLE I
YCSB CORE WORKLOADS USED

Workload A	50/50 reads/ rites
Workload B	95/5 reads/writes
Workload C	100% reads
Workload F	read a record, modify it, and write back the changes

As it is also suggested in YCSB's documentation, the DB was populated using the load function combined with Workload A. With the DB populated, started the process of experimentation, aimed at determining what effects each of the workloads had on the cluster's resource usage with the help of Ganglia monitoring and YCSB metrics. With the knowledge of these effects, its was then possible for a series of workload initiations to be planed, so that they trigger the protocol to order the addition or removal of one or more mongo shards.

III. THE PROTOCOL

Our protocol is implemented via 3 modules:
monitor.py Every ten (10) seconds, the monitor module receives the "gmetad" metrics via the XML API service that runs on the main ganglia server. It utilizes the *mdp* (Markov Decision Process) module [10] to model the cluster's state and select one action of {'add', 'rmv', 'nop'} based on the metrics. Then, if necessary, a newly created thread is tasked to inform the *actuator* of the action decided, in order to apply it on the cluster. This way the *actuator* module and the *monitor* module can run in parallel, without one blocking the other. After the *actuator* has finished, the state of the *mdp* is updated according to the success or failure of the *actuator*. Meanwhile *monitor* keeps collecting every ten (10) seconds new metrics and taking decisions based on them, but they are not used in any way in

this version of the protocol. As odd as it may seem, having the choice of continuous monitoring, rather than blocking, while actuator is running is important on any future modification of the protocol.

mdp.py The *mdp* module implements the process model of the "Markov Decision Process" for finite states and actions, providing a solution by utilizing the value iteration algorithm (see ***Algorithm 1***) [10]. More importantly, there is the implementation of MDP with states, that represent the number of nodes that are up and running. The possible actions are 'add', 'rmv', 'nop', the discount factor is made equal to 0.9 and the transition probabilities are updated dynamically by observing whether the actuator succeeds or not. The Reward(s) function is formulated by observing whether certain metric thresholds have been violated and by taking into account the normalized rewards of the previous metrics.

Algorithm 1 Value Iteration Algorithm

repeat
  $\Delta \leftarrow 0$
  for each s $\in \mathcal{S}$ **do**
    $\upsilon \leftarrow V(s)$
    $V(s) \leftarrow max_\alpha \sum_{s'} \mathcal{P}^\alpha_{ss'}[\mathcal{R}^\alpha_{ss'} + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |\upsilon - V(s)|)$
  end for
until $\Delta < \theta$ (a small positive number)
Output a deterministic policy, π, such that
$\pi(s) = argmax_\alpha \sum_{s'} \mathcal{P}^\alpha_{ss'}[\mathcal{R}^\alpha_{ss'} + \gamma V(s')]$

actuator.py The *actuator* module is responsible for propagating a given action to the underlying VMs, specifically BigData2 and BigData3. After querying mongos (query router) in order to retrieve the distribution of the shards in the mongodb cluster, according to the given action, the *actuator* either removes the last inserted shard from cluster or adds a new shard accordingly. Concerning the time needed, in order of a successful action, the 'add' action needs about three (3) seconds and the 'rmv' action needs about fifteen (15) minutes, because immediate rebalancing should be finished first. MongoDB calls this rebalancing "draining of a shard", because the data residing in the removing shard need to be distributed to the remaining cluster, before the successful removal. On the other hand, rebalancing in an newly added shard is performed in the background without stalling the 'add' action.

IV. Experimentation Process

At first, began the search for the throughput of the cluster, in a configuration with only one shard. This process highlighted the fact that with only one shard, the wright and read throughputs plateaued at about 3K and 6K operations/sec respectively. Parallel to the above process, the VM metrics where supervised via Ganglia, so that an essence of the various thresholds to be gained and thus obtain the appropriate thresholds that would next be used in the *monitor* module.

With the appropriate thresholds set for *monitor* to use, a series of experiments begun by first starting the *monitor*, in order to evaluate its behaviour.

Due to the empirical way of the threshold choosing, the resulting frequency of change between states, meaning adding or removing a shard, was low, which implies that the deviation of the resulting thresholds was equally low, similarly to the reference thresholds. This finding, confirms the difficult nature of finding the appropriate combination of generic thresholds, so that they fit well with (almost) any kind of workload.

V. Conclusion

In the end, a protocol was developed, based on a static-rule-base reactive decision making. There are certain disadvantages to this approach such as vulnerability to load spikes, as well as advantages such as low memory requirements and processing on each iteration. One observed key point is, on node removal when the rebalancing process should take place, the mongodb cluster is simultaneously trying to service the incoming load and rebalance the data residing in the removing nodes. That justifies the low frequency of removal decisions that should take place. The developing process was quite interesting, because a lot of new software programs were introduced, such as MongoDB, Ganglia and we were given the chance to develop a protocol similar to Tiramola. The experimenting part was more difficult than it seemed, because it involved a lot of time and empirical observations. Aside of a rule-based reactive protocol there are many different techniques, which include proactive decision making through prediction protocols, even utilizing machine learning. There is always room for improvement, especially on such a vast area of decision making protocols.

VI. Source Code

The source code for this project is accessible within a public **GitHub** repository [11], for details see:

KTsakogiannis/NTUA_BigData2022

Acknowledgment

This project was prepared in the context of the "Big Data Information Systems" course, at NTUA, under the valuable guidance of Professor Dimitrios Tsoumakos.

It is also important to mention that a large portion of this project was significantly influenced by the very useful work put into the "TIRAMOLA" protocol [1].

References

[1] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas and N. Koziris, "Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA," In proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2013.

[2] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). "Benchmarking cloud serving systems with YCSB." Proceedings of the 1st ACM Symposium on Cloud Computing
[3] Swaminathan, S. N. and Elmasri, R. (2016). "Quantitative analysis of scalable NoSQL databases." Proceedings - 2016 IEEE International Congress on Big Data, BigData Congress 2016, 323–326.
[4] Naskos, A., Gounaris, A. and Sioutas, S. (2016). "Cloud elasticity: A survey. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)," 9511.
[5] Massie, M., Li, B., Nicholes, B. and Vuksan, V. (2012). "Monitoring with Ganglia."
[6] Ganglia Source Code. [Online]. Available: https://github.com/ganglia
[7] YCSB Source Code. [Online]. Available: https://github.com/brianfrankcooper/YCSB
[8] Cloud Elasticity And Scalability. [Online]. Available: https://www.mongodb.com/cloud-elasticity-and-scalability
[9] Generating Workloads Using Ycsb. [Online]. Available: https://medium.com/idomongodb/mongodb-generating-workloads-using-ycsb-f0acbe704374
[10] Richard S. Sutton and Andrew G. Barto. "Reinforcement Learning: An Introduction," Chapter 4, Paragraph 4.
[11] Papadosifos, M., Tsakogiannis, K. and Typaldos, A. (2022). A Simple Protocol for Resizing MongoDB Clusters (Version 1.0.0) [Source Code]. https://github.com/KTsakogiannis/NTUA_BigData2022
[12] Ganglia MongoDB Plugin. [Online]. Available: https://github.com/skeeved/ganglia-mongodb-plugin