ProRenaTa: Proactive and Reactive Tuning to Scale a Distributed Storage System

Ying Liu*‡, Navaneeth Rameshan*†, Enric Monte†, Vladimir Vlassov* and Leandro Navarro†
*KTH Royal Institute of Technology, Sweden
Email: yinliu@kth.se, rameshan@kth.se, vladv@kth.se
†Universitat Politècnica de Catalunya, Barcelona, Spain
Email: rameshan@ac.upc.edu, enric.monte@upc.edu, leandro@ac.upc.edu
‡Université catholique de Louvain, Belgium

***Abstract*—Provisioning stateful services in the Cloud that guarantees high quality of service with reduced hosting cost is challenging to achieve. There are two typical auto-scaling approaches: predictive and reactive. A prediction based controller leaves the system enough time to react to workload changes while a feedback based controller scales the system with better accuracy. In this paper, we show the limitations of using a proactive or reactive approach in isolation to scale a stateful system and the overhead involved. To overcome the limitations, we implement an elasticity controller, ProRenaTa, which combines both reactive and proactive approaches to leverage on their respective advantages and also implements a data migration model to handle the scaling overhead. We show that the combination of reactive and proactive approaches outperforms the state of the art approaches. Our experiments with Wikipedia workload trace indicate that ProRenaTa guarantees a high level of SLA commitments while improving the overall resource utilization.**

Keywords*—*Elasticity, Auto-scaling, Workload prediction, Resource utilization, SLA.

I. Introduction

Hosting services in the Cloud are becoming more and more popular because of a set of desired properties provided by the platform, which include low setup cost, professional maintenance and elastic provisioning. Services that are elastically provisioned in the Cloud are able to use platform resources on demand, thus saving hosting costs by appropriate provisioning. Specifically, instances are spawned when they are needed for handling an increasing workload and removed when the workload drops. Enabling elastic provisioning saves the cost of hosting services in the Cloud, since users only pay for the resources that are used to serve their workload.

A well-designed elasticity controller helps reducing the cost of hosting services using dynamic resource provisioning and, in the meantime, does not compromise service quality. Levels of service quality are usually defined in SLAs (Service Level Agreements), which are negotiated and agreed between service consumers and the service providers. A violation of SLA affects both the provider and consumer. When a service provider is unable to uphold the agreed level of service, they typically pay penalties to the consumers. From the consumers perspective, a SLA violation can result in degraded service to their clients and consequently lead to loss in profits. Hence, SLA commitment is essential to the profit of both Cloud service providers and consumers.

In general, Cloud services can be coarsely characterized in two categories: state-based and stateless. Scaling stateless services is easier since no overhead of state migration is involved. However, scaling state-based services often requires state-transfer/replication, which introduces additional overhead during the scaling. In this paper, we investigate the elastic scaling of distributed storage systems, which provide indispensable services in the Cloud and are typical state-based systems. One of the most commonly defined SLAs in a distributed storage system is its service latency. Guaranteeing latency SLAs in back-end distributed storage systems is desirable in supporting many latency sensitive services, such as Web 2.0 services. However, it is a challenging task for an elasticity controller since it needs to achieve the following properties:

1. Resource allocation that satisfy both constraints: minimize provisioning cost and SLA violations.

2. Swift adaptation to workload changes without causing resource oscillation.

3. Be aware of scaling overheads, including the consumption of system resources and time, and prevent them from causing SLA violations.

4. Efficient use of resources under SLA during scaling. Specifically, when scaling up, it is preferable to add instances at the very last possible moment. In contrast, during scaling down, it is better to remove instances as soon as they are not needed anymore. The timings are challenging to control.

To the best of our knowledge, none of the state of the art systems achieve all these properties in their controller designs. Broadly speaking, elasticity in a distributed storage system is achieved in two ways. One solution relies on reacting to real-time system metrics, including CPU usage, incoming load, I/O operations, and etc. It is often referred as **reactive control**. Another approach is to explore historic access patterns of a system in order to conduct workload prediction and controls for the future. It is called **proactive control**.

The first approach can scale the system with a good accuracy since scaling is based on observed workload characteristics. However, a major disadvantage of this approach is that the system reacts to workload changes only after it is observed. As a result, SLA violations are observed in the initial phase of scaling because of data/state migration in order to add/remove instances in a distributed storage system and causes a period of disrupted service. The latter approach, on the other hand, is able to prepare the instances in advance and avoid any disruption in the service. However, the accuracy of workload prediction largely depends on application-specific access patterns. Worse, in some cases workload patterns are not even predictable. Thus, proper methods need to be designed and applied to deal with the workload prediction inaccuracies, which directly influences the accuracy of scaling that in turn impacts SLA guarantees and the provisioning costs.

In essence, proactive and reactive approach complement each other. Proactive approach provides an estimation of future workloads giving a controller enough time to prepare and react to the changes but having the problem of prediction inaccuracy. Reactive approach brings an accurate reaction based on current state of the system but without leaving enough time for the controller to execute scaling decisions.

We present ProRenaTa, which is an elasticity controller for distributed storage systems combining both proactive and reactive scaling techniques. ProRenaTa achieves the previously identified properties and makes the following contributions:

- ProRenaTa helps a storage system to adapt to workload changes without causing resource oscillation. A study of prediction methods for a typical web application (Wikipedia) focusing on pattern characterizations, recognitions and accurate predictions help us to achieve this property.
- ProRenaTa explicitly considers scaling overhead, i.e., data migration cost, to achieve high resource utilization and low latency SLA violation. A cost model and scheduler for data/state migration generates a premium scaling plan to execute the predicted scaling decision.
- ProRenaTa uses a reactive module to further guarantee accurate resource allocation, which minimizes latency SLA violations.

Our evaluation shows that ProRenaTa outperforms the state of the art approaches in terms of resource utilization (saving cost) and SLA commitment (achieving quality of service).

II. Observations and background

It is challenging to achieve elasticity in stateful systems especially under the constraint of SLA. There are two major reasons. One reason is that scaling stateful systems require state migrations, which often introduces an additional overhead. The other reason is that the scaling of such systems are often associated with delays. To be specific, adding or removing instances cannot be completed immediately because of the waiting time for state transfer. These are the two major reasons that cause SLA violations while scaling stateful systems. In this section, we briefly introduce the concepts of distributed storage systems. Then, we justify the above arguments with experimental observations.

A. Distributed storage systems

A distributed storage system provides an unified storage service to its clients by integrating and managing a large number of backend storage instances. Compared to traditional storage systems, distributed storage systems usually have the advantages of high scalability and high availability. Distributed storage systems can be organized using many different approaches. For example, Hadoop Distributed File System [1] organizes its storage instances using a centralized naming service provided by a single NameNode; Cassandra [2] and Dynamo [3] like systems employ distributed hash tables (DHTs) to decentralize the maintenance of the namespace and request routing; Spanner [4] and PNUTs [5] adopts multiple name service components to manage the namespace and request routing. In our work, we take a kind of distributed storage system that is organized similar to Cassandra. Background of such storage systems can be obtained in [2], [3]. Specifically, we use GlobLease [6] as the underlying storage system serving the workload. GlobLease is a key-value store that uses DHT for request routing and namespace maintenance similar to Cassandra [2]. Virtual tokens are implemented to even the workload distribution and overhead on addition or removal of nodes in the overlay. We setup GlobLease very similar to Cassandra using the read/write consistency level ”ONE”. More details of GlobLease is presented in [6].

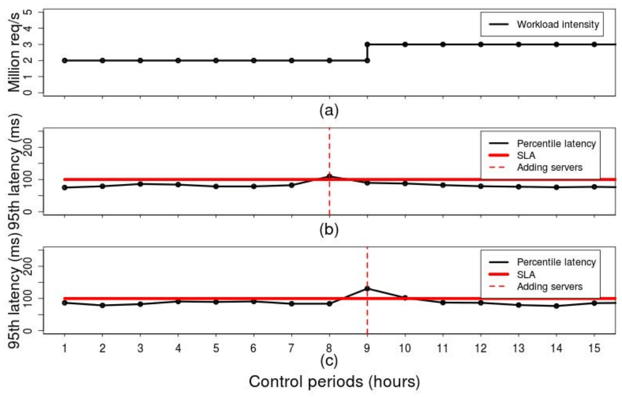

Fig. 1: Observation of SLA violations during scaling up. (a) denotes a simple increasing workload pattern; (b) scales up the system using a proactive approach; (c) scales up the system using a reactive approach

B. Observations

We setup experiments to investigate the scaling of GlobLease with respect to a simple workload pattern described in Figure 1 (a). The experiment is designed as simple as possible to demonstrate the idea. Specifically, we assume a perfect prediction of the workload patterns in a prediction based elasticity controller and a perfect monitor of the workload in a feedback based elasticity controller. The elasticity controllers try to add storage instances to cope with the workload increase in Figure 1 (a) to keep the low latency of requests defined in SLAs. Figure 1 (b) and Figure 1 (c) present the latency outcome using naive prediction and feedback based elasticity controller respectively. Several essential observations can be formalized from the experiments.

It is not always the workload that causes SLA violations. Typically, a prediction based elasticity control tries to bring up the capacity of the storage cluster before the actual workload increase. In Figure 1 (b), a prediction based controller tries to add instances at control period 8. We observe the SLA violation during this period because of the extra overhead, i,e, data redistribution, imposed on the system when adding storage instances. The violation is caused by the data transfer process, which competes with client requests in terms of servers' CPUs, I/Os, and bandwidths. We solve this problem by presenting a data migration model that controls the data transfer process based on the spare capacity of the server with respect to latency SLA commitment.

Another interesting observation can be seen from Figure 1 (c), which simulate the scaling of the system using a feedback approach. It shows that **scaling up after seeing a workload peak (at control period 9) is too late.** The SLA violation is observed because the newly added instances cannot serve the increased workload immediately. Specifically, proper portion of data needs to be copied to the newly added instances before they can serve the workload. Worse, adding instances at the last moment will even aggravate the SLA violation because of the scaling overhead like in the previous case. Thus, it is necessary to scale the system before the workload changes like using a prediction based approach.

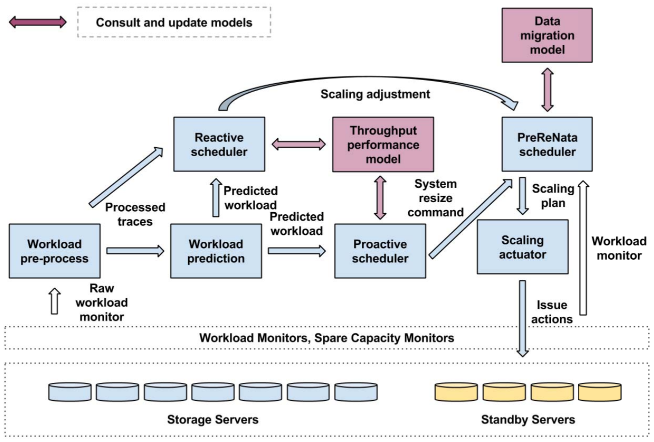

Fig. 2: ProRenaTa control framework

However, only using prediction based approach is not enough even though we can handle the data transfer overhead using a data migration model. It is because that **the prediction is not always accurate.** Even using Wikipedia workload [7] where the pattern is very predictable, a small amount of prediction errors are expected. We adjust those errors using a feedback approach. Combing the usage of prediction based approach and the feedback approach yields much better performance in terms of SLA commitments and resource utilization shown in our later evaluations.

III. System design

In this section, we present the design of ProRenaTa, an elasticity controller for distributed storage systems that combines both reactive and proactive control in order to achieve high system utilization and prevent SLA violations. Figure 2 shows the architecture of ProRenaTa. It follows the idea of MAPE-K (Monitor, Analysis, Plan, Execute - Knowledge) control loop with some customizations and improvements.

A. Monitor

The arrival rate of reads and writes on each node is monitored and defined as input workload in ProRenaTa. Then, the workload is fed to two modules: workload pre-processing and ProRenaTa scheduler.

Workload pre-process: The workload pre-processing module aggregates the monitored workload in a predefined window interval. We define this interval as smoothing window (SW). The granularity of SW depends on workload patterns. Very large SW will smooth out transient/sudden workload changes while very small SW will cause oscillation in scaling. The size of SW in ProRenaTa can be configured in order to adjust to different workload patterns.

The monitored workload is also fed to ProRenaTa scheduler to estimate the utilization of the system and calculate the spare capacity that can be used to handle scaling overhead. Detailed design of ProRenaTa is explained in Section III-C.

B. Analysis

Workload prediction: The pre-processed workload is forwarded to the workload prediction module for workload forecasting. The prediction methods will be explained in Section IV. Workload prediction is conducted every prediction window (PW). Specifically, at the beginning of each PW, the prediction module forecasts the workload intensity at the end of the current PW. Workload pre-processing provides an aggregated workload intensity at the beginning of each SW. In our setup, SW and PW have the same size and are synchronized. The output of the prediction module is an aggregated workload intensity marked with a time stamp that indicates the deadline for the scaling to match such workload. Workload aggregations and predictions are conducted at a key granularity. The aggregation of the predicted workload intensity on all the keys is the total workload, which is forwarded to the proactive scheduler and the reactive scheduler.

C. Plan

Proactive scheduler: The Proactive scheduler calculates the number of instances needed in the next PW using the performance model in Section V-A2. Then, it sends the number of instances to be added/removed to the ProRenaTa scheduler.

Reactive scheduler: The reactive scheduler in ProRenaTa is different from those that reacts on monitored system metrics. Our reactive scheduler is used to correct the inaccurate scaling of the system caused by the inaccuracy of the workload prediction. It takes in the pre-processed workload and the predicted workload. The pre-processed workload represents the current system status while the predicted workload is a forecast of workload in a PW. The reactive scheduler stores the predicted workload at the beginning of each PW and compares the predicted workload with the observed workload at the end of each PW. The difference from the predicted value and the observed value represents the scaling inaccuracy. Using the differences of the predicted value and the observed value as an input signal instead of monitored system metrics guarantees that the reactive scheduler can operate along with the proactive scheduler and not get biased because of the scaling activities from the proactive scheduler. The scaling inaccuracy, i,e, workload difference between prediction and reality, needs to be amended when it exceeds a threshold calculated by the throughput performance model. If scaling adjustments are needed, the number of instances that need to be added/removed is sent to the ProRenaTa scheduler.

ProRenaTa scheduler: The major task for ProRenaTa scheduler is to effectively and efficiently conduct the scaling plan for the future (provided by the proactive scheduler) and the scaling adjustment for now (provided by the reactive scheduler). It is possible that the scaling decision from the proactive scheduler and the reactive scheduler are contradictory. ProRenaTa scheduler solves this problem by consulting the data migration model, which quantifies the spare system capacity that can be used to handle the scaling overhead. The data migration model estimates the time needed to finish a scaling decision taking into account the current system status and SLA constraints explained in Section V-B. Assume that the start time of a PW is t_s and the end time of a PW is t_e. The scaling plan from the reactive controller needs to be carried out at t_s while the scaling plan from the proactive controller needs to be finished before t_e. Assume the workload intensity at t_s and t_e is W_s and W_e respectively. We assume a linear evolving model between current workload intensity and the future workload intensity. Thus, workload intensity at time t in a PW can be calculated by $W(t) = \gamma * t + W_s$ where $\gamma = (W_e - W_s)/(t_e - t_s)$. let $Plan_r$ and $Plan_p$ represent the scaling plan from the reactive controller and the proactive controller respectively. Specifically, a $Plan$ is a integer number that denotes the number of instances that needs to be added or removed. Instances are added when $Plan$ is positive, or removed when $Plan$ is negative. Note that the plan of the proactive controller needs to be conducted based on the completion of the reactive

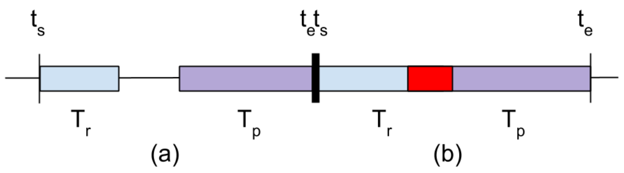

Fig. 3: Scheduling of reactive and proactive scaling plans

controller. It means that the actual plan that needs to be carried out by the proactive plan is $Plan_p^{'} = |Plan_p - Plan_r|$. Given workload intensity and a scaling plan to the data migration model, it needs T_r and T_p to finish the scaling plan from the reactive controller and the proactive controller respectively.

We assume that $T_r < (t_e - t_s)\&\&T_p < (t_e - t_s)$, i,e, the scaling decision by either of the controller alone can be carried out within a PW. This can be guaranteed by understanding the applications' workload patterns and tuning the size of PW accordingly. However, it is not guaranteed that $(T_r + T_p) < (t_e - t_s)$, i,e, the scaling plan from both controllers may not get finished without having an overlapping period within a PW. This interference needs to be prevented because having two controllers being active during an overlapping period violates the assumption, which defines only current system workload influence data migration time, in the data migration model.

In order to achieve the efficient usage of resources, ProRenaTa conduct the scaling plan from the proactive controller at the very last possible moment. In contrast, the scaling plan of the reactive controller needs to be conducted immediately. The scaling process of the two controllers are illustrated in Figure 3. Figure 3(a) illustrates the case when the reactive and proactive scaling do not interfere with each other. Then, both plans are carried out by the ProRenaTa scheduler. Figure 3(b) shows the case when the system cannot support the scaling decisions of both reactive and proactive controller. Then, only the difference of the two plans ($|Plan_r - |Plan_p - Plan_r||$) is carried out. And this plan is regarded as a proactive plan and scheduled to be finished at the end of this PW.

D. Execute

Scaling actuator: Execution of the scaling plan from ProRenaTa scheduler is carried out by the scaling actuator, which interacts with the underlying storage system. Specifically, it calls add server or remove server APIs exposed by the storage system and controls the data migration among storage servers. The quota used for data migration among servers are calculated by Prerenata scheduler and indicated to the actuator. The actuator limits the quota for data migration on each storage servers using BwMan [8], which is a bandwidth manager that allocates bandwidth quotas to different services running on different ports. In essence, BwMan uses Netem tc tools to control the traffic on each storage server's network interface.

E. Knowledge

To facilitate the decision making to achieve elasticity in ProRenaTa, there are three knowledge bases. The first one is the throughput model presented in Section V-A2, which correlates the server's capability of serving read and write requests under the constraint of SLA latency. The second one is the migration overhead model presented in Section V-B, which quantify the spare capacity that can be used to handle data migration overhead while performing system reconfiguration. The last one is the monitoring, which provides real-time workload information, including composition and intensity, in the system to facilitate the decision making in ProRenaTa.

Algorithm 1 illustrates the control flow of ProRenaTa. In the procedure of ProactiveControl, $PW.T_{i+1}$ (line 2) is the workload predicted at T_{i+1}, namely the start of the next control interval. The prediction of the workload ($workloadPrediction()$) is presented in Algorithm 2. A positive value of $\Delta VMs.T_{i+1}$ (line 4) indicates the number of instances to launch (scale up). A negative value of $\Delta VMs.T_{i+1}$ indicates the number of instances to remove (scale down). In the procedure of ReactiveControl, $W.T_i$ (line 7) is the workload observed at T_i In the procedure of ProRenaTaScheduler, $RS.T_i$ (line 11) is the maximum data rebalance speed achievable at T_i. $T.p$ and $T.r$ are the time to finish data rebalance for proactive and reactive scaling respectively.

Algorithm 1 ProRenaTa Control Flow

```
1: procedure PROACTIVECONTROL()
                                    ▷ Program starts at time $T_i$
2:     $PW.T_{i+1}$ ←workloadPrediction(Trace)
3:     $VMs.T_{i+1}$ ←throughputModel(PW.$T_{i+1}$)
4:     $\Delta VMs.T_{i+1} \leftarrow VMs.T_{i+1} - VMs.T_i$
5:
6: procedure REACTIVECONTROL()
                                    ▷ Program starts at time $T_i$
7:     $\Delta W.T_i \leftarrow W.T_i - PW.T_i$
8:     $\delta VMs.T_i \leftarrow$ throughputModel($\Delta W.T_i$)
9:
10: procedure PRORENATASCHEDULER()
                                    ▷ Program starts at $T_i$
11:     $RS.T_i$ ←dataMigrationModel($T_i$)
12:     $T.p$ ←analyticalModel($\Delta VMs.T_{i+1}$,$RS.T_i$)
13:     $T.r$ ←analyticalModel($\delta VMs.T_i$,$RS.T_i$)
14:     if $T.p + T.r > T_{i+1} - T_i$ then
15:         $VMsToChange \leftarrow \Delta VMs.T_{i+1} + \delta VMs.T_i$
16:         $t$ ←analyticalModel(VMsToChange,$RS.T_i$)
17:         $TimeToAct \leftarrow T_{i+1} - t$
18:         WaitUntil TimeToAct
19:         ConductSystemResize(VMsToChange)
20:     else
21:         ConductSystemResize($\delta VMs.T_i$)
22:         $TimeToAct \leftarrow T_{i+1} - T.p$
23:         WaitUntil TimeToAct
24:         ConductSystemResize($\Delta VMs.T_{i+1}$)
```

IV. WORKLOAD PREDICTION

The prediction of wikipedia workload is a specific problem that does not exactly fit the common prediction techniques found in literature. This is due to the special characteristics of the workload. On the one hand, the workload associated can be highly periodic, which means that the use of the context (past samples), will be effective for making an estimation of the demand. On the other hand the workload time series may have components that are difficult to model with demand peaks that are random. Although the demand peaks might have a periodic component (for instance a week), the fact that the amplitude is random, makes the use of linear combination seperated by week intervals unreliable. The classical methods are based on linear combinations of inputs and old outputs with a random residual noise, and are known as ARIMA, (Autoregressive-Integrated-Moving-Average) or Box-Jenkins models [9].

ARIMA assumes that the future observation depends on values observed a few lags in the past, and a linear combination of a set of inputs. These inputs could be of different origin, and the coefficients of the ARIMA model takes care of both, the importance of the observation to the forecast, and the scaling in case that the input has different units than the output. However, an important limitation of the ARIMA framework is that it assumes that the random component of the forecasting model is limited to the residual noise. This is a strong limitation because the randomness in the forecasting of workload, is also present in the amplitude/height of the peaks. Other prediction methodologies are based on hybrid methods that combine the ideas from ARIMA, with non-linear methods such as Neural Networks, which do not make hypothesis about the input-output relationships of the functions to be estimated. See for instance [10]. The hybrid time series prediction methods use Neural Netwoks or similar techniques for modeling possible non-linear relationships between the past and input samples and the sample to be predicted. Both methods, ARIMA and a hybrid method assume that the time series is stationary, and that the random component is a residual error, which is not the case of the workload time series.

A. Representative workload types

We categorize the workload to a few generic representative types. These categories are important because they justify the architecture of the prediction algorithm we propose.
Stable load and cyclic behavior: This behaviour corresponds to an waveform that can be understood as the sum of a few (i.e. 3 or 4) sinusoids plus a random component which can be modeled as random noise. The stable load and cyclic behavior category models keywords that have a clear daily structure, with a repetitive structure of maxima and minima. This category will be dealt with a short time forecast model.
Periodic peaks: This behaviour corresponds to peaks that appear at certain intervals, which need not be harmonics. The defining characteristic is the sudden appearance of the peaks, which run on top of the stable load. The periodic peaks category models keywords that have a structure that depends on a memory longer than a day, and is somehow independent of the near past. This is the case of keywords that for instance, might be associated to a regular event, such as chapters of a TV series that happen certain days of the week.This category will be dealt with a long time forecast model.
Random peaks and background noise: This behaviour corresponds to either rarely sought keywords which have a random behaviour of low amplitude or keywords that get popular suddenly and for a short time. As this category is inherently unpredictable, unless there is outside information available, we deal with his category using the short term forecasting model, which accounts for a small percentage of the residual error.

B. Prediction methodology

The forecasting method consists of two modules that take into account the two kind of dependencies in the past: short term for **stable load, cyclic behavior and background noise** and long term for **periodic peaks**.

The short term module will make an estimate of the actual value by using a Wiener filter [11] which combines linearly a set of past samples in order to estimate a given value, in this case, the forecasted sample. In order to make the forecast the short term module uses information in a window of several hours. The coefficients of the linear combination are obtained by minimizing the Mean Square Error (MSE) between the forecast and the reference sample. The short term prediction is denoted as $\tilde{x}_{Shrt}[n]$. The structure of the filter is as follows.

$$\tilde{x}_{Shrt}[n + N_{FrHr}] = \sum_{i=0}^{L_{Shrt}} w_i x[n - i]$$

where; L_{Shrt} is the length of the Wiener filter, N_{FrHr} is the forecasting horizon, $x[n]$ is the n-th sample of the time series, and w_i is the i-th coefficient of the Wiener filter. Also, as the behaviour of the time series is not stationary, we recompute the weights of the Wiener filter forecaster when the prediction error (MSE) increases for certain length of time [11].

The long term module $\tilde{x}_{Lng}[n]$ takes into account the fact that there are periodic and sudden rises in the value to be forecasted. These sudden rises depend on the past values by a number of samples much higher than the number of past samples of the short term predictor L_{Shrt}. These rises in demand have an amplitude higher than the rest of the time series, and take a random value with a variance that empirically has been found to be variable in time. We denote these periodicities as a set $\{P_0 \dots P_{N_p}\}$, where P_i indicates the i-th periodicity in the sampling frequency and N_p the total number of periodicities. Empirically, in a window of one month, the periodicities of a given time series were found to be stable in most cases, i.e. although the variance of the peaks changed, the values of P_i were stable. In order to make this forecast, we generated a train of periodic peaks, with an amplitude determined by the mean value taken by the time series at different past periods. This assumes a prediction model with a structure similar to the auto-regressive (AR), which combines linearly past values at given lags. The structure of this filter is

$$\tilde{x}_{Lng}[n + N_{FrHr}] = \sum_{i=0}^{N_p} \sum_{j=0}^{L_j} h_{i,j} x[n - jP_i]$$

where, N_{FrHr} is the forecasting horizon, N_p is the total number of periodicities, L_j is the number of weighted samples of the i-th periodicity, $h_{i,j}$ is the weight of each sample used in the estimation, $x[n]$ is the n-th sample of the time series. We do not use the moving average (MA) component, which presupposes external inputs. A model that takes into account external features, should incorporate a MA component.

The final estimation is as follows:

$$\tilde{x}[n+N_{FrHr}] = \begin{cases} \tilde{x}_{Lng}[n + N_{FrHr}] & \text{if n+}N_{FrHr} = k_0 P_i \\ \tilde{x}_{Shrt}[n + N_{FrHr}] & \text{if n+}N_{FrHr} \neq k_0 P_i \end{cases}$$

where the decision on the forecast to use is based on testing if $n+N_{FrHr}$ is a multiple of any of the periodicities P_i . Specific details on the implementation of each of the predictors and the pseudo code can be found in the Appendix A.

V. Controller models

In this section, we present the models that we use in ProRenaTa controllers. In general, there are two models that are constantly consulted and updated in every control period. One is a throughput performance model, which defines the correlation between the throughput of the system and the number of participating instances. The other one is a data migration model, which quantifies the available capacity that can be used to handle the scaling overhead under the current system load.

A. Throughput performance model

The throughput performance model determines the minimum number of servers that is needed to meet the SLA requirements with respect to a specific workload. In order to build a suitable performance model for a distributed storage system, the first thing that needs to be understood is how requests are distributed to servers.

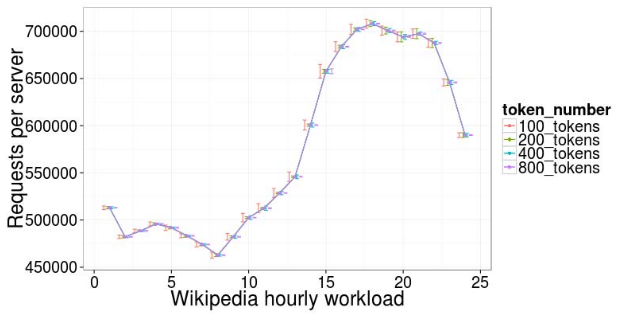

Fig. 4: Standard deviation of load on 20 servers running a 24 hours Wikipedia workload trace. With larger number of virtual tokens assigned to each server, the standard deviation of load among servers decreases

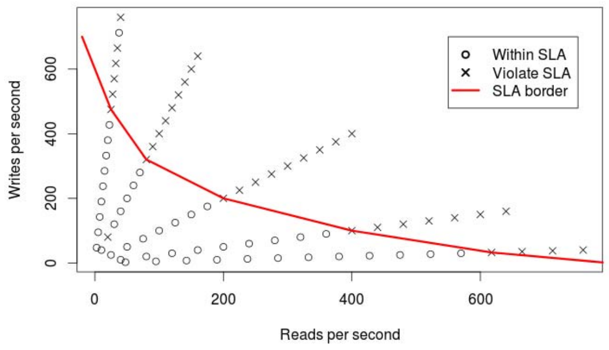

Fig. 5: Throughput performance model

1) Load balance in distributed storage systems: We target distributed storage systems organized using DHTs as introduced in Section II. We assume that virtual tokens are implemented in the target distributed storage system. Enabling virtual tokens allows a physical server to host multiple discrete virtual tokens in DHT namespace and storing the corresponding portions of data. The number of virtual tokens on a physical server is proportional to the server's capability. The virtual tokens assigned to a physical server are randomly selected in our case. Figure 4 shows that with sufficient number of virtual tokens, requests tend to evenly distributed among physical servers with different workload intensities replayed using a 24 hour Wikipedia access trace. Thus, we can derive a performance model for the distributed storage system by modeling its individual servers. Specifically, under the assumption of evenly distributed workload, if any server in the system does not violate the SLA constraints, the system as a whole does not violate the SLA constraints proposed and argued in [12].

2) Performance model based on throughput: Figure 5 shows an off-line trained throughput-based performance model for a physical server. Different models can be build for different server flavors using the same profiling method. The workload is represented by the request rate of read and write operations. Under a specified SLA latency constraint, a server can be in 2 states: satisfy SLA (under the SLA border) or violate SLA (beyond the SLA border). We would like to arrange servers to be utilized just under the SLA border. This translates to having a high resource utilization while guaranteeing the SLA requirements. The performance model takes a specific workload intensity as the input and outputs

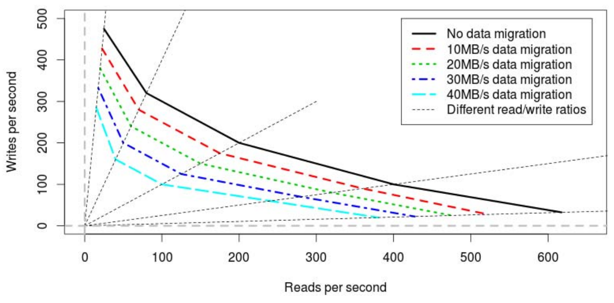

Fig. 6: Data migration model under throughput and SLA constraints

the minimum number of instances that is needed to handle the workload under SLA. It is calculated by finding the minimum number of servers that results in the load on each server ($Workload/NumberOfServers$) closest to and under the SLA border in Figure 5. In the real experiment, we have setup a small margin for over-provisioning. This can not only better achieve SLA, but also allows more spare capacity for data transfer during scaling up/down. This margin is set as 2 servers in our experiment and can be configured differently case to case to tradeoff the scaling speed and SLA commitment with resource utilization.

B. Data migration model

The data migration model in ProRenaTa monitors the load in the system and outputs the maximum data transfer rate that can be used for scaling activities without compromising the SLA. By using the data migration model, ProRenaTa is able to obtain the time that is needed to conduct a scaling plan, i,e, adding/removing n instances, under current system status and SLA constraint. A detailed analytical model can be found in Appendix B.

1) Statistical model: In this section, we describe the profiling of a storage instance under three parameters: read request rate, write request rate and data migration speed. The data migration is manually triggered in the system by adding new storage instances. We observe that during data migration, network bandwidth is the major resource that compete between data transfer and client request handling in storage servers. Thus, the bandwidth used for data migration is controlled as a parameter. Under the constraint of SLA latency, we have multiple SLA borders under different data transfer rate shown in Figure 6. Using this statistical model, given current workload and servers in the system, we are able to find the spare capacity that can be used for data migration. Specifically, a given total workload consisting of read and write request rate is uniformly mapped to each server and expressed as a data point in Figure 6. The closest border below this data point indicates the data migration speed that can be offered.

VI. Evaluation

In this section, we present the evaluation of ProRenaTa elasticity controller using a workload synthesized from Wikipedia access logs during 2009/03/08 to 2009/03/22. The access traces are available here [7]. We first present the setup of the storage system (GlobLease) and the implementation of the workload generator. Then, we present the evaluation results of ProRenaTa and compare its SLA commitments and resource utilization with feedback and prediction based elasticity controller.

TABLE I: GlobLease and workload generator setups

Specifications	GlobLease VMs	Workload VMs
Instance Type	m1.medium	m1.xlarge
CPUs	Intel Xeon 2.8 GHz*2	Intel Xeon 2.0 GHz*8
Memory	4 GiB	16 GiB
OS	Ubuntu 14.04	Ubuntu 14.04
Instance Number	5 to 20	5 to 10

A. Implementation

1) Deployment of the storage system: GlobLease [6] is deployed on a private OpenStack Cloud platform hosted at out university (KTH). Homogeneous virtual machine instance types are used in the experiment for simplicity. It can be easily extended to heterogeneous scheduling by profiling capabilities of different instances types using the methodology described in Section V-A2. Table I presents the virtual machine setups for GlobLease and the workload generator.

2) Workload generator: We implemented a workload generator in JAVA that emulates workloads in different granularities of requests per second. To setup the workload, a couple of configuration parameters are fed to the workload generator including the workload traces, the number of client threads, and the setup of GlobLease.

Construction of the workload from raw Wikipedia access logs. The access logs from Wikepedia provides the number of accesses to each page every 1 hour. The first step to prepare a workload trace is to remove the noise in accesses. We removed non-meaningful pages such as "Main_Page", "Special:Search", "Special:Random", etc from the logs, which contributes to a large portion of accesses and skews the workload pattern. Then, we chose the 5% most accessed pages in the trace and abandoned the rest. There are two reasons for this choice: First, these 5% popular keys constructs nearly 80% of the total workload. Second, access patterns of these top 5% keys are more interesting to investigate while the remaining 95% of the keys are mostly with 1 or 2 accesses per hour and very likely remain inactive in the following hours. After fixing the composition of the workload, since Wikipedia logs only provide page views, i,e, read accesses, we randomly chose 5% of these accesses and transformed them as write operations. Then, the workload file is shuffled and provided to the workload generator. We assume that the arrivals of clients during every hour follow a Poisson distribution. This assumption is implemented in preparing the workload file by randomly placing accesses with a Poisson arrival intensity smoothed with a 1 minute window. Specifically, 1 hour workload has 60 such windows and the workload intensities of these 60 windows form a Poisson distribution. When the workload generator reads the workload file, it reads the whole accesses in 1 window and averages the request rate in this window, then plays them against the storage system. We do not have the information regarding the size of each page from the logs, thus, we assume that the size of each page is 10 KB. We observe that the prepared workload is not able to saturate GlobLease if the trace is played in 1 hour. So, we intensify the workload by playing the trace in 10 minutes instead.

The number of client threads defines the number of concurrent requests to GlobLease in a short interval. We configured the concurrent client threads as 50 in our experiment. The size of the interval is calculated as the ratio of the number of client threads over the workload intensity.

The setup of GlobLease provides the addresses of the storage nodes to the workload generator. Note that the setup of GlobLease is dynamically adjusted by the elasticity controllers during the experiment. Our workload generator also implements a load balancer that is aware of the setup changes from a programmed notification message sent by the elasticity controllers (actuators). Table II summaries the parameters configured in the workload generator.

TABLE II: Wikipedia Workload Parameters

Concurrent clients	50
Request per second	roughly 3000 to 7000
Size of the namespace	around 100,000 keys
Size of the value	10 KB

3) Handling data transfer: Like most distributed storage systems, GlobLease implements data transfer from nodes to nodes in a greedy fashion, which puts a lot of stress on the available network bandwidth. In order to guarantee the SLA of the system, we control the network resources used for data transfer using BwMan [8]. BwMan is a previous work in our group that arbitrate network resources between user-oriented workload and system maintenance workload, data transfer in this case, under the constraint of SLAs. The amount of available network resources allocated for data transfer is calculated using the data migration model in Section V-B.

B. Evaluation results

We compare ProRenaTa with two baseline approaches: feedback control and prediction-based control.

Most recent feedback-based auto-scaling literature on distributed storage systems are [13], [12], [14], [15]. These systems correlate monitored metrics (CPU, workload, response time) to a target parameter (service latency or throughput). They then periodically evaluate the monitored metrics to verify the commitment to the required level of service. Whenever the monitored metrics indicates a violation of the service quality or a waste of provisioned resources, the system decides to scale up/down correspondingly. Our implementation of the feedback control for comparison relies on similar approach and represents the current state of the art in feedback control. Our feedback controller is built using the throughput model described in section V-A2. Dynamic reconfiguration of the system is performed at the beginning of each control window to match the averaged workload collected during the previous control window.

Most recent prediction-based auto-scaling work are [16], [17], [18]. These systems predict interested metrics. With the predicted value of the metrics, they scale their target systems accordingly to match the desired performance. We implemented our prediction-based controller in a similar way by predicting the interested metric (workload) described in section IV. Then, the predicted value is mapped to system performance using an empirical performance model described in section V-A2. Our implementation closely represents the existing state of the art for prediction based controller. System reconfiguration is carried out at the beginning of the control window based on the predicted workload intensity for the next control period. Specifically, if the workload increase warrants addition of servers, it is performed at the beginning of the current window. However, if the workload decreases, the removal of servers are performed at the beginning of the next window to ensure SLA. Conflicts may happen at the beginning of some windows because of a workload decrease followed by a workload increase. This is solved by simply adding/merging the scaling decisions.

ProRenaTa combines both feedback control and prediction-based control but with more sophisticated modeling and

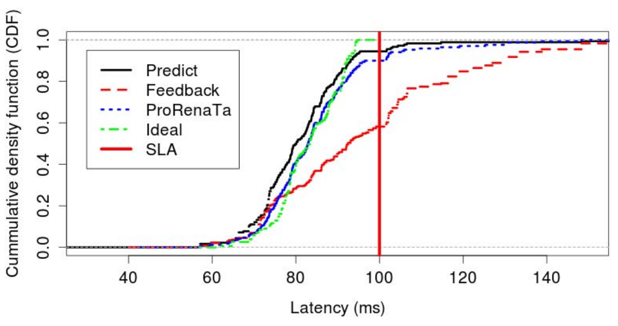

Fig. 7: Aggregated CDF of latency for different approaches

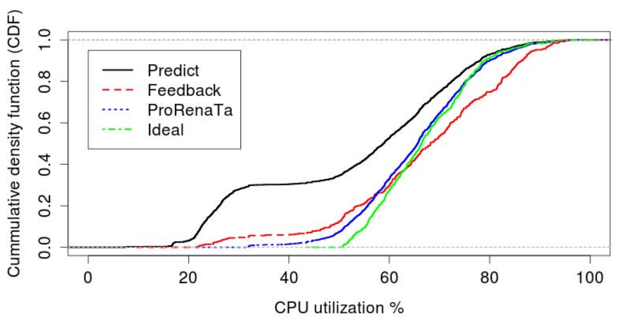

Fig. 8: Aggregated CDF of CPU utilization for different approaches

scheduling. Prediction-based control gives ProRenaTa enough time to schedule system reconfiguration under the constraint of SLAs. The scaling is carried out at the last possible moment in a control window under the constraint of SLA provided by the scaling overhead model described in Section V-B. This model guarantees ProRenaTa with less SLA violations and better resource utilization. In the meantime, feedback control is used to adjust the prediction error at the beginning of each control window. The scheduling of predicted actions and feedback actions is handled by ProRenaTa scheduler.

In addition, we also compare ProRenaTa with an ideal case. The ideal case is implemented using a theoretically *perfect elasticity controller*, which knows the future workload, i,e, predicts the workload perfectly. The ideal also uses ProRenaTa scheduler to scale the cluster. So, comparing to the prediction based approach, the ideal case not only uses more accurate prediction results but also uses better scheduling, i,e, the ProRenaTa scheduler.

1) Performance overview: Here, we present the evaluation results using the aforementioned 4 approaches with the Wikipedia workload trace from 2009/03/08 to 2009/03/22. We focus on the 95th percentile latency of requests calculated from each hour and the CPU utilization monitored every second.

SLA commitment. Figure 7 presents the cumulative distribution of 95 percentile latency by running the simulated Wikipedia workload from 2009/03/08 to 2009/03/22. The vertical red line demonstrates the SLA latency configured in each elasticity controller.

We observe that the feedback approach has the most SLA violations. This is because the algorithm reacts only when it observes the actual workload changes, which is usually too late for a stateful system to scale. This effect is more obvious when the workload is increasing. The scaling overhead along with the workload increases lead to a large percent of high latency requests. ProRenaTa and the prediction-based approach achieve nearly the same SLA commitments as shown in Figure 7. This is because we have an accurate workload prediction algorithm presented in IV. Also, the prediction-based algorithm tries to reconfigure the system before the actual workload comes, leaving the system enough time and resources to scale. However, we shown in the next section that the prediction-based approach does not efficiently use the resources, i,e, CPU, which results in more provision cost.

CPU utilization. Figure 8 shows the cumulative distribution of the aggregated CPU utilization on all the storage servers by running the two weeks simulated Wikipedia workload. It shows that some servers in the feedback approach are under utilized (20% to 50%), which leads to high provision cost, and some are saturated (above 80%), which violates SLA latency. This CPU utilization pattern is caused by the nature of reactive approach, i,e, the system only reacts to the changing workload when it is observed. In the case of workload increase, the increased workloads usually saturate the system before it reacts. Worse, by adding storage servers at this point, the data migration overhead among servers aggravate the saturation. This scenario contributes to the saturated CPU utilization in the figure. On the other hand, in the case of workload decrease, the excess servers are removed only in the beginning of the next control period. This leads to the CPU under utilization in some scenarios.

It is shown in figure 8 that a large portion of servers remain under utilized when using the prediction based elasticity control. This is because of the prediction-based control algorithm. Specifically, in order to guarantee SLA, in the case of workload increase, servers are added in the previous control period while in the case of workload decrease, servers are removed in the next control period. Note that the CPU statistics are collected every second on all the storage servers. Thus, the provisioning margin between control periods contributes to the large portion of under utilized CPUs.

In comparison with the feedback or prediction based approach, ProRenaTa is smarter in controlling the system. Figure 8 shows that most servers in ProRenaTa have a CPU utilization from 50% to 80%, which results in a reasonable latency that satisfies the SLA. Under/over utilized CPUs are prevented by a feedback mechanism that corrects the prediction errors. Furthermore, there is much less over provision margins observed in the prediction based approach because of the data migration model. ProRenaTa assesses and predicts system spare capacity in the coming control period and schedules system reconfigurations (scale up/down) to an optimized time (not in the beginning or the end of the control period). This optimized scheduling is calculated based on the data migration overhead of the scaling plan as explained in Section V-B. All these mechanisms in ProRenaTa leads to an optimized resource utilization with respect to SLA commitment.

2) Detailed performance analysis: In the previous section, we presented the aggregated statistics about SLA commitment and CPU utilization by playing a 2 weeks Wikipedia access trace using four different approaches. In this section, we zoom in the experiment by looking at the collected data during 48 hours. This 48 hours time series provides more insights into understanding the circumstances that different approaches tend to violate the SLA latency.

Workload pattern. Figure 9 (a) shows the workload pattern and intensity during 48 hours. The solid line presents the actual workload from the trace and the dashed line depicts

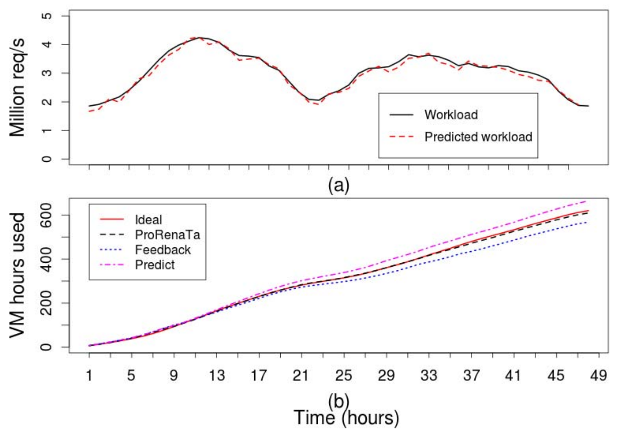

Fig. 9: Actual workload and predicted workload and aggregated VM hours used corresponding to the workload

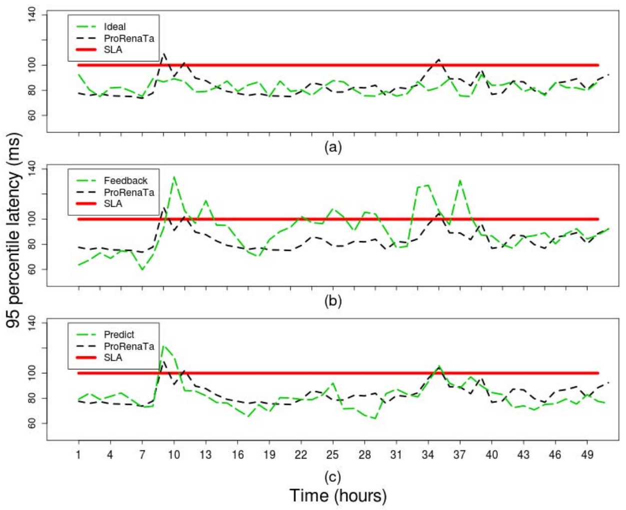

Fig. 10: SLA commitment comparing ideal, feedback and predict approaches with ProRenaTa

the predicted workload intensity by our prediction algorithm presented in Section IV.

Total VM hours used. Figure 9 (b) demonstrates the aggregated VM hours used for each approach under the workload presented in Figure 9 (a) during 48 hours. The ideal provisioning is simulated by knowing the actual workload trace beforehand and feeding it to the ProRenaTa scheduler, which generates an optimized scaling plan in terms of the timing of scaling that takes into account the scaling overhead. It is shown that ProRenaTa is very close to the VM hours used by the ideal case. On the other hand, the predict approach has consumed more VMs during this 48 hours, which leads to high provisioning cost. The feedback approach has allocated too few VMs, which has caused a lot of latency SLA violations shown in Figure 7.

SLA commitment. Figure 10 presents the comparison of SLA achievement using the ideal approach (a), the feedback approach (b) and the prediction based approach (c) compared to ProRenaTa under the workload described in Figure 9 (a). Compared to the ideal case, ProRenaTa violates SLA when the workload increases sharply. The SLA commitments are met in the next control period. The feedback approach on the other hand causes severe SLA violation when the workload increases. ProRenaTa takes into account the scaling overhead and takes actions in advance with the help of workload prediction, which gives it advantages in reducing the violation in terms of extend and period. In comparison with the prediction based approach, both approaches achieve more or less the same SLA commitment because of the pre-allocation of servers before the workload occurs. However, it is shown in Figure 8 that the prediction based approach cannot use CPU resource efficiently.

3) Utility Measure: An efficient elasticity controller must be able to achieve high CPU utilization and at the same time guarantee latency SLA commitments. Since achieving low latency and high CPU utilization are contradictory goals, the utility measure needs to capture the goodness in achieving both these properties. While a system can outperform another in any one of these properties, a fair comparison between different systems can be drawn only when both the aspects are taken into account in composition. To this order, we define the utility measure as the cost incurred:

$$U = VM\_hours + Penalty$$

$$Penalty = DurationOfSLAViolations * penalty\_factor$$

$DurationOfSLAViolations$ is the duration through the period of the experiment the SLA is violated. We vary the penalty factor which captures the different cost incurred for SLA violations. We analyze the results obtained by running a 48 hours Wikipedia workload trace using different auto-scaling controllers. Figure 11 shows the utility measure for 4 different scaling approaches. Without any penalty for SLA violations, feedback approach performs the best. But as the penalty for SLA violations increase, ProRenaTa and the ideal approach achieve the lowest utility (cost), which is much better than both feedback and prediction-based auto-scaling approaches.

VII. Related work

There are numerous works done in the field of auto-scaling recently under the context of Cloud computing. Broadly speaking, they can be characterized as scaling stateless services [16], [19], [18] and stateful service [12], [13], [14], [20]. We characterize the approaches into two categories: reactive and proactive. Typical methods used for auto-scaling are threshold-based rules, reinforcement learning or Q-learning (RL), queuing theory, control theory and time series analysis.

The representative systems that use threshold-based rules to scale a service are Amazon Cloud Watch [21] and RightScale [22]. This approach defines a set of thresholds or rules in advance. Violating the thresholds or rules to some extent will trigger the action of scaling. Threshold-based rule is a typical implementation of reactive scaling.

Reinforcement learning are usually used to understand the application behaviors by building empirical models. Simon [23] presents an elasticity controller that integrates several empirical models and switches among them to obtain better performance predictions. The elasticity controller built in [20] uses analytical modeling and machine-learning. They argued that by combining both approaches, it results in better controller accuracy.

Ali-Eldin [24] uses the queueing theory to model a Cloud service and estimates the incoming load. It builds proactive controllers based on the assumption of a queueing model. It presents an elasticity controller that incorporates a reactive controller for scale up and proactive controllers for scale down.

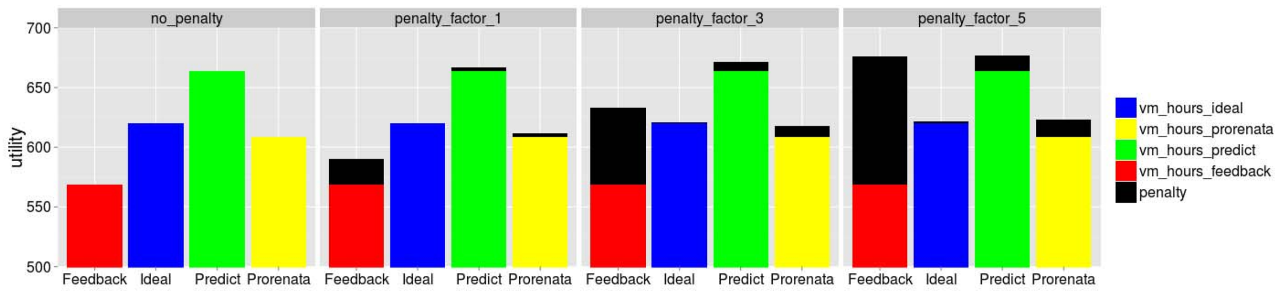

Fig. 11: Utility for different approaches

Recent influential works that use control theory to achieve elasticity are [12], [13], [14]. The reactive module in ProRenaTa uses similar technique to achieve auto-scaling as the ones applied in Scads director [12] and ElastMan [13]. Specifically, both approaches build performance models on system throughput using model predictive control. Scads director tries to minimize the data migration overhead associated with the scaling by arranging data into small data bins. However, this only alleviate the SLA violations. Lim [14] uses CPU utilization as the monitored metrics in a classic feedback loop to achieve auto-scaling. A data migration controller is also modeled in this work. However, it is only used to tradeoff the SLA violation with the scaling speed.

Recent approaches using time-series analysis to achieve auto-scaling are [17], [18]. [17] adapts second order ARMA for workload forecasting under the World Cup 98 workload. [18] proves that it is accurate to use wavelets to provide a medium-term resource demand prediction. With the help of the prediction results, VMs can be spawned/migrated before it is needed in order to avoid SLA violations. [16] uses on-line resource demand prediction with prediction errors corrected.

ProRenaTa differs from the previous approaches in two aspects. First, most of the previous approaches either use reactive or proactive scaling techniques, ProRenaTa combines both approaches. Reactive controller gives ProRenaTa better scale accucacy while proactive controller provides ProRenaTa enough time to handle the scaling overhead (data migration). The complimentary nature of both approaches provide ProRenaTa with better SLA guarantees and higher resource utilizations. Second, to our best knowledge, when scaling a stateful system, e,g, a storage system, none of the previous systems explicitly model the cost of data migration when scaling. Specifically, ProRenaTa assesses the scaling cost under the scaling goal with continuous monitoring of the spare capacity in the system. In essense, ProRenaTa employs time-series analysis to realize a proactive scaling controller. This is because of the workload characteristics discussed in Section IV and analyzed in [25]. A reactive module is applied to correct prediction errors, which further guarantees the SLA and boosts the utilization of resources.

VIII. Conclusion

In this paper, we investigate the efficiency of an elasticity controller that combines both proactive and reactive approaches for auto-scaling a distributed storage system. We show the limitations of using proactive or reactive approach in isolation to scale a stateful system. We design ProRenaTa that combines both proactive and reactive approaches. ProRenaTa improves the classic prediction based scaling approach by taking into account the scaling overhead, i.e., data/state migration. Moreover, the reactive controller helps ProRenaTa to achieve better scaling accuracy, i.e., better resource utilization and less SLA violations, without causing interference with the scaling activities scheduled by the proactive controller. Our results indicate that ProRenaTa outperforms the state of the art approaches by guaranteeing a high level of SLA commitments while also improving the overall resource utilization.

As for future work, we consider to make the training process of the models used in ProRenaTa online. The online training will ease the deployment of ProRenaTa and make the scaling decision more accurate by having finer-grained of parameters trained. We also consider to extend the data migration model to be able to estimate the cost of handling skewed workloads.

Acknowledgment

This work was supported by the Erasmus Mundus Joint Doctorate in Distributed Computing funded by the EACEA of the European Commission under FPA 2012-0030, the FP7 project CLOMMUNITY funded by the European Commission under EU FP7 GA number 317879, the End-to-End Clouds project funded by the Swedish Foundation for Strategic Research under the contract RIT10-0043, the project SpeechTech4All (TEC2012-38939-C03-02), and by the Spanish government under contract TIN2013-47245-C2-1-R. The authors would like to give sincere thanks to Bogdan Nicolae for his useful comments and suggestions to improve the quality of the paper. We would also like to thank the anonymous reviewers for their valuable comments.

References

[1] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[2] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[3] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007.

[4] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 251–264, Berkeley, CA, USA, 2012. USENIX Association.

[5] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, August 2008.

[6] Ying Liu, Xiaxi Li, and Vladimir Vlassov. Globlease: A globally consistent and elastic storage system using leases. http://dx.doi.org/10.13140/2.1.2183.7763. (To

appear in) Proceedings of the 2014 IEEE International Conference on Parallel and Distributed Systems (ICPADS '14).

[7] Wikipedia traffic statistics v2. http://aws.amazon.com/datasets/4182.

[8] Ying Liu, Vamis Xhagjika, Vladimir Vlassov, and Ahmad Al Shishtawy. Bwman: Bandwidth manager for elastic services in the cloud. In *Parallel and Distributed Processing with Applications (ISPA), 2014 IEEE International Symposium on*, pages 217–224, Aug 2014.

[9] George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*. John Wiley & Sons, 2013.

[10] Timothy Masters. *Neural, novel and hybrid algorithms for time series prediction*. John Wiley & Sons, Inc., 1995.

[11] Thomas Kailath, Ali H Sayed, and Babak Hassibi. Linear estimation. 2000.

[12] Beth Trushkowsky, Peter Bodík, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. The scads director: Scaling a distributed storage system under stringent performance requirements. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

[13] Ahmad Al-Shishtawy and Vladimir Vlassov. Elastman: Elasticity manager for elastic key-value stores in the cloud. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, pages 7:1–7:10, New York, NY, USA, 2013. ACM.

[14] Harold C. Lim, Shivnath Babu, and Jeffrey S. Chase. Automated control for elastic storage. In *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, pages 1–10, New York, NY, USA, 2010. ACM.

[15] Simon J. Malkowski, Markus Hedwig, Jack Li, Calton Pu, and Dirk Neumann. Automated control for elastic n-tier workloads based on empirical modeling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 131–140, New York, NY, USA, 2011. ACM.

[16] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 5:1–5:14, New York, NY, USA, 2011. ACM.

[17] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 500–507, July 2011.

[18] Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Sethuraman Subbiah, and John Wilkes. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proc. of the USENIX International Conference on Automated Computing (ICAC13). San Jose, CA*, 2013.

[19] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Optimal cloud resource auto-scaling for web applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 58–65, May 2013.

[20] Diego Didona, Paolo Romano, Sebastiano Peluso, and Francesco Quaglia. Transactional auto scaler: Elastic scaling of in-memory transactional data grids. In *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, pages 125–134, New York, NY, USA, 2012. ACM.

[21] Amazon cloudwatch. http://aws.amazon.com/cloudwatch/.

[22] Right scale. http://www.rightscale.com/.

[23] Simon J. Malkowski, Markus Hedwig, Jack Li, Calton Pu, and Dirk Neumann. Automated control for elastic n-tier workloads based on empirical modeling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 131–140, New York, NY, USA, 2011. ACM.

[24] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212, April 2012.

[25] Ahmed Ali Eldin, Ali Rezaie, Amardeep Mehta, Stanislav Razroev, Sara Sjostedt de Luna, Oleg Seleznjev, Johan Tordsson, and Erik Elmroth. How will your workload look like in 6 years? analyzing wikimedia's workload. In *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, IC2E '14, pages 349–354, Washington, DC, USA, 2014. IEEE Computer Society.

APPENDIX A
PREDICTOR IMPLEMENTATION

Short term forecast the short term component is initially computed using as data the estimation segment, that is the same initial segment used in order to determine the set of periods P_i of the long term forecaster. On the forecasting component of the data, the values of the weights w_i of the Wiener filter are updated when the forecasting error increases for a certain length of time. This assumes a time series with a statistical properties that vary with time. The procedure for determining the update policy of the Wiener filter is the following: first the forecasting error at a given moment

$$Error[n] = |\tilde{x}[n] - x[n]|^2$$

note that this is computed taking into account a delay equal to the forecasting horizon N_{FrHr}, that is $\tilde{x}[n]$ is compute form the set of samples: $\{x[n - N_{FrHr}] \dots x[n - N_{FrHr} - L_{Shrt}]\}$. In order to decide when to update the coefficients of the Wiener filter, we compute a long term MSE and a short term MSE by means of an exponential window. Computing the mean value by means of an exponential window is justified because it gives more weight to the near past. The actual computation of the MSE at moment n, weights the instantaneous error $Error[n]$, with the preceding MSE at $n-1$. The decision variable $Des[n]$ is the ratio between the long term MSE at moment n $MSE_{lng}[n]$ and the the short term MSE at moment n $MSE_{srt}[n]$:

$$MSE_{lng}[n] = (1 - \alpha_{lng})Error[n] + \alpha_{lng}MSE_{lng}[n-1]$$
$$MSE_{srt}[n] = (1 - \alpha_{shrt})Error[n] + \alpha_{srt}MSE_{shrt}[n-1]$$

where α is the memory parameter of the exponential window, with $0 < \alpha < 1$ and for our experiment α_{lng} was set to 0.98, which means that the sample $n - 100$ is given 10 times less weight that the actual sample and α_{shrt} was set to 0.9, which means that the sample $n - 20$ is given 10 times less weight that the actual sample. The decision value is defined as:

$$Des[n] = MSE_{lng}[n]/max(1, MSE_{srt}[n])$$

if $Des[n] > Thrhld$ it is assumed that the statistics of the time series has changed and a new set of coefficients w_i are computed for the Wiener filter. The training data sample consists of the near past and are taken as $\{x[n] \dots x[n - MemL_{Shrt}]\}$. For our experiments we took as threshold $Thrhld = 10$ and $Mem = 10$. Empirically we have found that the performance does not change much when these values are slightly perturbed. Note that the max() operator in the denominator of the expression that computes $Des[n]$ prevents a division by zero in the case of keywords with low activity.

Long term forecast In order to compute the parameters P_i of the term $\tilde{x}_{Lng}[n]$ we reserved a first segment (estimation segment) of the time series and we computed the auto-correlation function on this segment. The autocorrelation function measures the similarity of the time series to itself as a function of temporal shifts and the maxima of the autocorrelation function indicates it's periodic components denoted by P_i. These long term periodicities are computed from the lags of the positive side of the auto-correlation function with a value above a threshold. Also, we selected periodicities corresponding to periods greater than 24 hours. The amplitude threshold was defined as a percentage of the auto correlation at lag zero (i.e. the energy of the time series). Empirically we found that the 0.9 percent of the energy allowed to model the periods of interest. The weighting value $h_{i,j}$ was taken as $1/L_j$ which gives the same weight to each of the periods used for the estimation. The number of weighted periods L_j was selected to be two, which empirically gave good results.
Algorithm 2 illustrates the prediction module of ProReNata.

APPENDIX B
ANALYTICAL MODEL FOR DATA MIGRATION

We consider a distributed storage system that runs in a Cloud environment and uses ProRenaTa to achieve elasticity while respecting the latency SLA. The storage system is organized using DHT and virtual tokens are implemented. Using virtual tokens in a distributed storage system provides it with the following properties: 1. The amount of data stored in each physical server is proportional to its capability. 2. The amount of workload distributed to each physical server is proportional to its capability. 3. If enough bandwidth is given, the data migration speed from/to a instance is proportional to its capability.

Algorithm 2 ProReNata prediction module

```
1:  procedure InitializeLongTermModule($L_{ini}$)
                                        ▷ Uses $\{x[0] \dots x[L_{ini}]\}$
2:      $P_i \leftarrow$ ComputeSetOfPeriodicities()
    ▷ $P_i$ is the set of $N_p$ long term periodicities computed in
    an initial segment from the auto-correlation function.
3:      $L_i \leftarrow$ ValuesOfLengthFor$P_i$($P_i$)
                         ▷ For this experiment $L_i = 2 \; \forall$ period $P_i$
4:      $h_{i,j} \leftarrow$ValuesOfFilter($P_i$,$L_i$)
                         ▷ For this experiment $h_{i,j} = \frac{1}{L_i}$ for $1 \cdots L_i$
5:
6:  procedure InitializeShortTermModule($L_{ini}$)
                 ▷ Uses $\{x[0] \dots x[L_{ini}]\}$ for computing $w_i$.
7:      $\{w_i\} \leftarrow$ InitalValuesOfPredictor()
            ▷ Weights $w_i$ are initialized by solving the Wiener
    equations.
8:      $\{N_{FrHr}, L_{Shrt}, MemL_{Shrt}\} \leftarrow$ TopologyOfShort-
    TermPredictor()
9:      $\{Thrhld, \alpha_{srt}, \alpha_{lng}\} \leftarrow$ UpDatingParamOfWienerFil-
    ter()
    ▷ Parameters $\{\alpha_{srt}, \alpha_{lng}\}$ define the memory of the filter
    that smooths the MSE, and $Thrhld$, is the threshold that
    determines the updating policy.
10:
11: procedure ShortTimePrediction($w_i, N_{FrHr}, x$)
12:     $\tilde{x}_{Shrt}[n + N_{FrHr}] = \sum_{i=0}^{L_{Shrt}} w_i x[n-i]$
    ▷ Compute $\tilde{x}_{Shrt}[n + N_{FrHr}]$ from a linear combination
    of the data in a window of length $L_{Shrt}$.
13:
14: procedure UpdateShortTermPredictor()
15:     $Error[n] = |\tilde{x}[n] - x[n]|^2$
16:
        $MSE_{lng}[n] = (1 - \alpha_{lng})Error[n] + \alpha_{lng} MSE_{lng}[n-1]$   (1)
        $MSE_{srt}[n] = (1 - \alpha_{shrt})Error[n] + \alpha_{srt} MSE_{shrt}[n-1]$   (2)
    $Des[n] = MSE_{lng}[n] / max(1, MSE_{srt}[n])$
    ▷ Estimation of the short term and long term value of the
    $MSE$, and the decision variable $Des[n]$.
17:     if $Des[n] > Thrhld$ then
18:         Compute values of the Wiener filter using data
19: $\{x[n] \dots x[n - MemL_{Shrt}]\}$
20:
21: procedure LongTimePrediction($h_{i,j}, P_i, L_i, x$)
22:     $\tilde{x}_{Lng}[n + N_{FrHr}] = \sum_{i=0}^{N_p} \sum_{j=0}^{L_j} h_{i,j} x[n - jP_i]$
                                                        ▷ Compute
    $\tilde{x}_{Lng}[n + N_{FrHr}]$ from a linear combination of the data
    in a window of length corresponding to the periods $P_i$.
23:
24: procedure FinalEstimation()
25:
    $\tilde{x}[n+N_{FrHr}] = \begin{cases} x_{L\tilde{n}g}[n + N_{FrHr}] & \text{if n+}N_{FrHr} = k_0 P_i \\ \tilde{x}_{Shrt}[n + N_{FrHr}] & \text{if n+}N_{FrHr} \neq k_0 P_i \end{cases}$
```

At time t, Let D be the amount of data stored in the storage system. We consider that the amount of data is very large so that reads and writes in the storage system during a small period do not significantly influence the data amount stored in the system. We assume that, at time t, there are N storage instances. For simplicity, here we consider that the storage instances are homogeneous. Let C represents the capability of each storage instance. Specifically, the maximum read capacity in requests per second and write capacity in requests per second is represented by $\alpha * C$ and $\beta * C$ respectively under the SLA latency constraint. The value of α and β can be obtained from our performance model.

Let L denotes the current workload in the system. Therefore, $\alpha^{'} * L$ are read requests and $\beta^{'} * L$ are write requests. Under the assumption of uniform workload distribution, the read and write workload served by each physical server is $\alpha^{'} * L/N$ and $\beta^{'} * L/N$ respectively. We define function f to be our data migration model. It outputs the maximum data migration rate that can be obtained under the current system load without compromising SLAs. Thus, function f depends on system load ($\alpha^{'} * L/N, \beta^{'} * L/N$), server capability ($\alpha * C, \beta * C$) and the SLA ($SLA_{latency}$).

We assume the predicted workload is $L_{predicted}$. According to the performance model introduced in the previous section, we know that a scaling plan in terms of adding or removing instances can be given. Let us consider a scaling plan that needs to add or remove n instances. When adding instances, n is a positive value and when removing instances, n is a negative value.

First, we calculate the amount of data that needs to be reallocated. It can be expressed by the difference of the amount of data hosted on each storage instance before scaling and after scaling. Since all the storage instances are homogeneous, the amount of data stored in each storage instance before scaling is D/N. And the amount of data stored in each storage instance after scaling is $D/(N+n)$. Thus, the amount of data that needs to be migrated can be calculated as $|D/N - D/(N+n)| * N$, where $|D/N - D/(N+n)|$ is for a single instance. Given the maximum speed that can be used for data migration ($f()$) on each instance, the time needed to carry out the scaling plan can be calculated.

$$Time_{scale} = \frac{|D/N - D/(N+n)|}{f(\alpha * C, \beta * C, \alpha^{'} * L/N, \beta^{'} * L/N, SLA_{latency})}$$

The workload intensity during the scaling in the above formula is assumed to be constant L. However, it is not the case in the real system. The evolving pattern of the workload during scaling is application specific and sometimes hard to predict. For simplicity, we assume a linear evolving pattern of the workload between before scaling and after scaling. However, any workload evolving pattern during scaling can be given to the data migration controller with little adjustment. Remind that the foreseeing workload is $L_{predicted}$ and the current workload is L. If a linear changing of workload is assumed from L to $L_{predicted}$, using basic calculus, it is easy to know that the effective workload during the scaling time is the average workload $L_{effective} = (L + L_{predicted})/2$. The time needed to conduct a scaling plan can be calculated using the above formula with the effective workload $L_{effective}$.

We can obtain α and β from the performance model for any instance flavor. $\alpha^{'}$ and $\beta^{'}$ are obtained from workload monitors. Then, the problem left is to fine a proper function f that defines the data migration speed under certain system setup and workload condition with respect to SLA constraint. The function f is obtained using the statistical model explained in section V-B1.