Performance and Cost Analysis between Elasticity Strategies over Pipeline-structured Applications

Vinícius Meyer[1][a], Miguel G. Xavier[1][b], Dionatra F. Kirchoff [1][c], Rodrigo da R. Righi[2][d]
and Cesar A. F. De Rose[1][e]

[1]*Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil*
[2]*University of Vale do Rio dos Sinos (UNISINOS), São Leopoldo, Brazil*

Keywords: Resource Management, Elasticity, Pipeline-structured Applications.

Abstract: With the advances in eScience-related areas and the growing complexity of scientific analysis, more and more scientists are interested in workflow systems. There is a class of scientific workflows that has become a standard to stream processing, called pipeline-structured application. Due to the amount of data these applications need to process nowadays, Cloud Computing has been explored in order to accelerate such processing. However, applying elasticity over stage-dependent applications is not a trivial task since there are some issues that must be taken into consideration, such as the workload proportionality among stages and ensuring the processing flow. There are studies which explore elasticity on pipeline-structured applications but none of them compare or adopt different strategies. In this paper, we present a comparison between two elasticity approaches which consider not only CPU load but also workload processing time information to reorganize resources. We have conducted a number of experiments in order to evaluate the performance gain and cost reduction when applied our strategies. As results, we have reached an average of 72% in performance gain and 73% in cost reduction when comparing non-elastic and elastic executions.

1 INTRODUCTION

Scientific workflows are widely performed to model processes in eScience-related areas and are defined as a collection of tasks that are processed in a specific order to perform a real application. Scientific workflow has been proved an efficient and popular method to model various scientific computing problems in parallel and distributed systems (Li et al., 2018). However, as the complexity of scientific computing demand has increased, this class of application becomes increasingly data-intensive, communication-intensive and computation-intensive. Moreover, traditional infrastructures such as clusters and grids are expensive appliances and sometimes complex to expand. Therefore, it is crucial to perform workflows in cloud computing environments as they implement a pay-as-you-go cost model (Aldossary and Djemame, 2018).

Cloud computing is a large-scale distributed computing ecosystem that provides resources solution delivered on demand to its customers, driven by economies (Juve and Deelman, 2011). Those who benefit from this technology are generating a large amount of data on various services, which increasingly comes to show all typical properties of scientific applications. The technological advantages of clouds, such as elasticity, scalability, accessibility, and reliability, have generated significantly better conditions to execute scientific experiments than private in-house IT infrastructures (Li et al., 2018). Elasticity is a fundamental characteristic of cloud computing aimed at autonomous and timely provisioning and releasing of shared resources in response to variation in demands dynamically with time (Mera-Gómez et al., 2016). Managing elasticity implies in being effective and efficient to provide resource management. Moreover, a cloud provider, in general, intends to reduce their costs by an efficient resource sharing and minimizing energy consumption of their infrastructure (Liu et al., 2018).

There are studies focused on performing elasticity in cloud computing (Anderson et al., 2017; Aldos-

[a] https://orcid.org/0000-0001-5893-5878
[b] https://orcid.org/0000-0003-4306-5325
[c] https://orcid.org/0000-0002-6604-8723
[d] https://orcid.org/0000-0001-5080-7660
[e] https://orcid.org/0000-0003-0070-0157

Meyer, V., Xavier, M., Kirchoff, D., Righi, R. and F. De Rose, C.
Performance and Cost Analysis between Elasticity Strategies over Pipeline-structured Applications.
DOI: 10.5220/0007729004040411
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 404-411
ISBN: 978-989-758-365-0

sary and Djemame, 2018) and studies focused on exploring stage-dependent applications (Li et al., 2010; Meyer et al., 2019), however, none of them compare or adopt more than one strategy. In this context, this paper presents and examines two elasticity strategies for pipeline-structured applications, in order to achieve a gain of performance and cost reduction.

2 BACKGROUND

2.1 Cloud Computing and Elasticity

Cloud computing is an emerging technology that is becoming increasingly popular because, among other advantages, allows customers to easily deploy elastic applications, greatly simplifying the process of acquiring and releasing resources to a running application, while paying only for the resources allocated (pay-per-use or pay-as-you-go model). Elasticity and dynamism are the two key concepts of cloud computing (Lorido-Botran et al., 2014). Using the definition from NIST [1] "Cloud computing is a model for enabling ubiquitous, convenient and on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction". This definition means that elasticity defines the ability of the cloud infrastructure provider to change quickly the amount of allocated resource capacity, over time, according to the actual users' demand.

Vertical elasticity (scaling up/down) is the ability that cloud providers have to add/remove processing, memory and/or storage resources inside a running virtual instance. Horizontal elasticity consists of adding/removing instances of virtual machines in/from the cloud environment. These instances can be virtual machines, containers, or even application modules (in SaaS). Replication is a wide method used to provide elasticity, being used in most public providers (Galante and d. Bona, 2012).

2.2 Elasticity Methods

There are two classes of automatic policies on cloud computing: Proactive and Reactive. Proactive approach normally uses heuristics to anticipate the system's load behavior and based on these results, to decide when and how to scale in/out resources. On the other hand Reactive approach is based on Rule-Condition-Action mechanism. A rule is composed of a set of conditions that when satisfied trigger some actions over the underlying cloud. Every condition considers an event or a metric of the system which is compared against a threshold. The information about metrics values and events is provided by the infrastructure monitoring system or by the application (Galante and d. Bona, 2012). Furthermore, using Reactive method implies that the system reacts to changes in the workload only when those changes have been detected (Lorido-Botran et al., 2014). In this paper, we have adopted a Reactive approach in our experiments.

2.3 Pipeline-structured Applications

Pipeline-structured workflows enable the decomposition of a repetitive sequential process into a succession of distinguishable sub-processes called stages, each of which can be efficiently executed on a distinct processing element or elements which operate concurrently. Pipelines are exploited at fine-grained level in loops through compiler directives and in operating system file streams, and at coarse-grained level in parallel applications employing multiple processors (Gonzalez-Velez and Cole, 2008).

A pipeline-structured application initially receives an input, and the last stage returns the result. The processing is done from stage to stage, and the output of each stage is the input of the next one. Each stage is allocated to a processing element, Virtual Machines (VMs) in this case, in order to compose a parallel pipeline. The performance of a pipeline can be characterized in terms of latency, the time is taken for one input to be processed by all stages, and throughput, the rate at which inputs can be processed when the pipeline reaches a steady state. Throughput is primarily influenced by the processing time of the slowest stage or bottleneck (Bharathi et al., 2008).

3 PROBLEM STATEMENT AND ELASTICITY STRATEGIES

3.1 Problem Description

A pipeline application is a type of workflow that receives a set of tasks, which must pass through all stages of this application in a sequential manner, which can lead to a prohibitive execution time (Meyer et al., 2019). The input stream for applications that use pipeline standards can be intense, erratic or irregular (Righi et al., 2016a). However, specific dependencies and distributed computing problems arise

[1] http://www.nist.gov/

due to the interaction between the processing stages and the mass of data that must be processed. Here, we highlight the most critical problem coming from pipeline-structured applications: According to the task flow behavior, some stages may have degraded performance, delaying subsequent stages and ultimately interfering in the entire application's performance. In this context, we compare two elasticity strategies for pipeline applications to take advantage of the dynamic resource provisioning capabilities of cloud computing infrastructure.

3.2 System Architecture

To reach performance gain on application execution, we have created techniques to provide elasticity automatically and dynamically in the stages of pipeline-structured applications in a cloud computing environment. For this purpose, we have designed a system with some characteristics to support our solution strategies, as follows: (i) Creating a coordination mechanism between existing nodes and virtual machines, knowing which VM belongs to each stage and when a node is active or inactive; (ii) Performing load balancing at each stage, to not over-provisioning VMs in that stage; (iii) Using a communication strategy between all components that is asynchronous and operates regardless of the current dimension of the cloud infrastructure; (iv) Distributing tasks among different VMs, independent of their processing capacity; (v) Ensuring elasticity at each stage and also in the whole system.

Our techniques are designed to operate at the PaaS level, using a private cloud infrastructure, which allows non-elastic pipeline-structured applications to derive the benefits of computational cloud elasticity without the need for user intervention. To provide elasticity, the model operates with allocation, consolidation, and reorganization of instances of virtual machines over physical machines. The elasticity is supplied independently at each stage in an exclusive way. There is an agent, named Stage Controller (SC), responsible for orchestrating all stage requests. Each stage has a certain number of virtual machines in operation, distributing tasks among them. The SC receives the requests and places them in the stage's queues. After that, SC distributes the next task to the VMs available in that stage. Elasticity Manager (EM) monitors each stage and the overall application. According to the rules established in the chosen technique and SLA, EM applies elastic actions. Figure 1 shows system architecture overview.

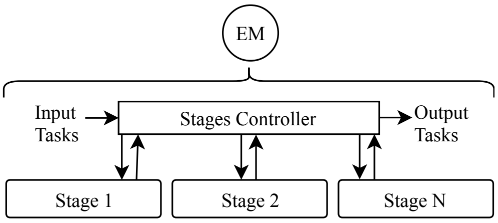


Figure 1: System architecture overview.

Here is applied the reactive, horizontal and automatic concept of elasticity, as in (Meyer et al., 2019; Righi et al., 2016b). The horizontal elasticity was chosen because the vertical one has limitations among available resource dependencies in a single node. Also, most famous operating systems do not allow on the fly changes in the resources (Dutta et al., 2012).

Each virtual machine uses only one processor of a certain node, and for each virtual machine, only one process will be sent at a time. SC is also an instantiated virtual machine that controls task queues, divides tasks into sub-tasks, and distribute them to the virtual machines allocated for the stage in question. After performing the processing is responsible for grouping and forwarding the result to the next stage of the application. Each virtual machine is allocated to a specific stage. Since it is allocated to stage 1, for example, it will only process requests requested by stage 1 until it is consolidated or transferred to another stage.

Since all stages are monitored individually, each one can undergo different elasticity actions. While some stage is instantiating more resources, another may be consolidating them. When the EM decides to consolidate resources on a given stage, a checking is made over the other stages, and if any other stage is requiring resources, the EM transfers that VM to that stage instead of being consolidated. This approach reduces time in the initialization of its operating systems. If the system needs to instantiate new resources, EM checks if there is any allocation or consolidation activity already happening in that stage. If there is, EM keeps the monitoring. If it does not, it sends a message to Cloud Front-End to add another VM in that stage.

3.3 Elasticity Decisions

The Elasticity Manager periodically monitors the CPU load of virtual machines on each application stage. It collects processing load (CPU) values from each virtual machine, which is executing the subtasks of its specific stage and applies a time-series calculation, also considering values previously collected to acquire the total load, here called as General CPU Load (GCL). Monitoring happens through a cycle of repetitions. After having a considerable number of data collections, EM applies an average on these values, for each stage. According to the upper and lower thresholds (Lorido-Botran et al., 2014), EM evaluates the necessity to instantiate/consolidate virtual machine(s) in each stage.

The General CPU Load of the system is obtained through Equation 2, where the function called *GCL (o, e)*, calculates the simple average of all processing loads of the virtual machines in observation o for stage e. Where q_e represents the number of virtual machines at the stage e. *CPU (i, o, e)* represents the CPU load of virtual machines i, at the instant o in the stage e. CPU loads are obtained through the Simple Exponential Smoothing (SES) method, represented by *SES (o, e)*, according to Equation 1.

$$SES(o,e) = \begin{cases} \frac{CPU(o,e)}{2} \text{ if } o = 0 \\ \frac{SES(o-1,e)}{2} + \frac{CPU(o,e)}{2} \text{ if } o \neq 0 \end{cases} \quad (1)$$

$$GCL(o,e) = \frac{\sum_{i=0}^{q_e} SES(i,o,e)}{q_e} \quad (2)$$

In order to evaluate the techniques, two algorithms have been tested individually over each scenario. Algorithm 1 follows the idea from (Righi et al., 2016b), in which the algorithm considers only the CPU load of the running application. If at a certain application's moment, the CPU load of some stage is above the upper threshold, EM includes more resource in that stage. If the CPU load is below the lower threshold, EM removes a resource from that stage. This algorithm is named **ReactiveC** (RC).

```
Data: monitoring data
Result: elastic decisions
initialization;
int x=0;
while application is executing do
    for (x<stages amount) do
        get GCL(stage x);
        if (GCL(stage x) > UpperT) then
            add VM in stage x;
        end
        if (GCL(stage x) < LowerT) then
            remove VM from stage x;
        end
        x++;
    end
    x=0;
end
```

Algorithm 1: ReactiveC (RC) monitoring routine.

The communication between pipeline stages might be completely asynchronous and the efficiency of this parallel model is directly dependent on the ability to balance the load between its stages (Bharathi et al., 2008). Taking this idea into consideration, Algorithm 2 not only considers CPU load but also the time execution metrics. This algorithm aims to reduce the idleness of stage resources caused by delays in the processing of the previous stage(s). *LTT* represents the Last Task Time, *avgLT* means the Last Time average and *sdLT* is the standard deviation from time metric. When CPU load violates the upper threshold or when the time of the last task performed is bigger than the average time minus standard deviation among all stages, EM adds more resources in that monitored stage. To remove resources is mandatory that the CPU load is below the lower threshold and the time of the last task executed on that stage is smaller than the other stage's average plus standard deviation. The purpose of this approach is to create an interval of time that is acceptable for not making elastic actions. This idea is an improvement from (Meyer et al., 2019) and is named as **ReactiveCT** (RCT).

```
Data: monitoring data
Result: elastic decisions
initialization;
int x=0;
boolean lc=false;
while application is executing do
    for (x<stages amount) do
        get GCL(stage x);
        get LTT(stage x);
        if (GCL(stage x) > UpperT) then
            add VM in stage x;
        end
        if (GCL(stage x) < LowerT) then
            lc=true;
        end
        if (LTT(stage x) < (avgLC-sdLT) & lc== true) then
            remove VM from stage x;
        end
        if (LTT(stage x) > (avgLC+sdLT)) then
            add VM in stage x;
        end
        x++;
    end
    x=0;
end
```

Algorithm 2: ReactiveCT (RCT) monitoring routine.

4 EVALUATION METHODOLOGY

4.1 Elastic Speedup and Elastic Efficiency

In order to observe the computational cloud elasticity gain for pipeline-structured applications, two metrics have been adopted as an extension of the concepts of speedup and efficiency: Elastic Speedup (ES) and Elastic Efficiency (EE) (Righi et al., 2016a). These metrics are exploited according to the horizontal elasticity, where virtual machine instances can be added or consolidated during application processing, changing the number of processes (CPUs) available. To evaluate the system, it is considered a homogeneous environment and each virtual machine instance can execute 100% of a computational processing core. ES is calculated by the function *ES (i, l, u)* according to Equation 3, where i represents the initial number of virtual machines while *u* and *l* represent the upper and lower limits for the number of virtual machines defined by the SLA. Moreover, t_{ne} and t_e refer to execution times of the executed application in elastic and non-elastic scenarios. Thus, t_{ne} is interpreted by the minimum number of virtual machines (*i*) to execute the whole application.

$$ES(i,l,u) = \frac{t_{ne}(i)}{t_e(i,l,u)} \tag{3}$$

The function *EE(i,l,u)*, represented by the Equation 4, calculates the elastic efficiency. Theses function parameters are the same as the function ES. Efficiency represents how effective the use of resources is, and this is positioned as the denominator of the formula. In this case, the number of resources are dynamically changed and, for that reason, a mechanism has been created to achieve a single value. EE assumes the execution of a monitoring system which captures the time spent in each configuration of virtual machines. The Equation 5 presents the metric *Resources* used in the EE calculation indicating the resource usage of the application where $pt_e(j)$ is the time interval in which the application was executed with *j* virtual machines. Equation 4 presents the parameter *i* in the numerator, which is doing multiplication with elastic speedup.

$$EE(i,l,u) = \frac{ES(i,l,u) \times i}{Resources(i,l,u)} \tag{4}$$

$$Resources(i,l,u) = \sum_{j=l}^{u}(j \times \frac{pt_e(j)}{t_e(i,l,u)}) \tag{5}$$

4.2 Energy and Cost Model

To complement the resource consumption analysis, not only Elastic Efficiency will be considered but also a way to measure the resources consumed during the execution of the application, named *Energy*. This metric is based on the close relationship between energy consumption and resources consumption (Orgerie et al., 2014). In this work, we have applied the same idea from a model utilized by Amazon and Microsoft: it is considered the VM amount in each time unit (e.g one hour). This idea is presented in Equation 6. The variable $pt_e(j)$ represents the time interval that the application was executed with *j* VM instances. The time unity depends on the pt_e value (seconds, minutes, hours) and the strategy is to sum the number of virtual machines used in each unit of time. Thus, *Energy* measures the use from *l* to *u* virtual machines instances, considering the partial time execution in each organization of the used resources. The Energy metric allows us to compare different applications that use elasticity ranges.

$$Energy(l,u) = \sum_{j=l}^{u}(j \times pt_e(j)) \tag{6}$$

To estimate the elasticity viability in several situations, a metric was adopted that calculates the execution time cost by multiplying the total execution time of the pipeline-structured application by the energy used, according to the Equation 7. This idea is an adaptation of parallel computation with elastic scenarios use cases. Energy consumption is proportional to the resources usage (Righi et al., 2016a). The goal is to obtain a lesser cost with elastic utilization compared to static resources usage. Summing up, a configuration can be considered as bad, if it is able to reduce the total execution time by half of the elastic application, but spends six times more, increasing the costs.

$$Cost = application\_time \times Energy \tag{7}$$

$$Cost_{\mathrm{e}} \leq Cost_{\mathrm{we}} \tag{8}$$

The goal is to preserve the truth of inequality 8, where $Cost_e$ represents the cost of application execution with elasticity and $Cost_{we}$ the cost of application execution without elasticity.

4.3 Application and Test Scenarios

To mitigate the elasticity strategies, we have developed a synthetic application that processes images over three different stages, following the idea from (Li et al., 2010) and (Meyer et al., 2019). The original images pass through three stages suffering sequential changes and producing a final result. In order to create load variations, the size of the image areas has been distributed in four scenarios: Constant, Increasing, Descending and Oscillating. The difference among them is the order of area size of each image. For instance, Increasing scenario starts with a small image and it goes increasing until the end of the image list. Decreasing is exactly the opposite of Increasing, the area size of each image starts large and goes decreasing. The same logic is applied to the other two scenarios. This workload variation forces the system to have different processing levels, forcing the different elasticity arrangements. Thus, the sum of all image areas in each load configuration results in the same value, so in each test, the amount of processed data is the same.

In all scenarios, with elasticity actions, many thresholds were tested. As lower thresholds, 20%, 30% and 40% have been used. For each lower threshold, the upper threshold has been set to 60%, 70% and 80%, producing nine variations to each one of the four workload scenarios. Thus, this amount of tests was executed on each elasticity algorithm. To analyze the performance improvement with elasticity strategies we also executed all proposed scenarios without making elastic actions. In this case, the application ran with only the minimum number of necessary resources to complete all the tasks (WE, Without Elasticity).

4.4 Computational Environment

All performed tests have been run over a Private Cloud Computing Environment. We have adopted OpenNebula[2] (v 5.4.13) as Cloud Front-End Middleware. The proposed study was executed over 10 Dell Optiplex 990, with 8GB of RAM and a Core i5 CPU each one. Thus, the environment is considered as homogeneous. EM runs in one of these machines, and for that purpose, this one is not used by the cloud. The communication among the machines has been made with Fast Ethernet (100 MBit/s) interconnection. All used codes and results are available in a GitHub repository[3].

5 RESULTS AND DISCUSSIONS

5.1 Analyzing Speedup and Efficiency

In the Constant scenario, the best speedup is achieved by using the RC strategy (74% of performance gain), and the worst case in this workload is adopting the RCT strategy (70% of performance gain). On the other hand, in all other scenarios, the best performance was reached by using the RCT strategy. When comparing their results with applications which do not use elasticity, Decreasing scenario improves its performance in 71% (best) and 61% (worst). The increasing scenario has 81% (best) and 68% (worst) of performance gain. Oscillating scenario achieves 76% (best) and 67% (worst) of performance gain, respectively. The reason for this behavior is that when the workload does not change, the task time factor does not affect the resource reorganization since the tasks in each stage are already balanced among them. When the workload has variations over the application execution, task time factor helps to balance the proportionality among the stages of the application, increasing the data flow and achieving a gain of performance.

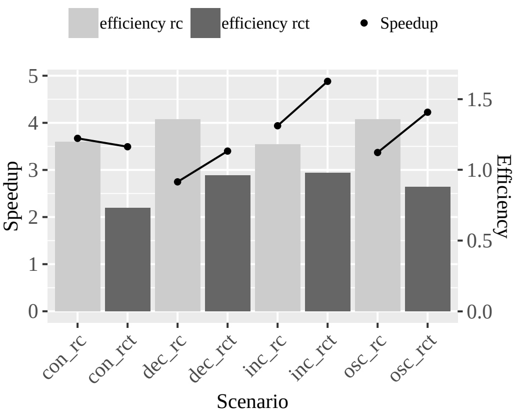


Figure 2: Speedup and Efficiency average from all proposed scenarios.

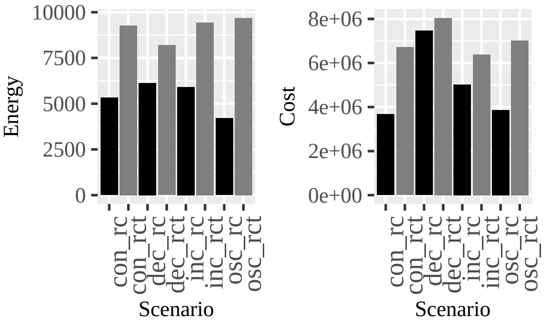
Figure 2 shows the average of all speedup and efficiency metrics in each scenario (among all thresholds combinations). It is possible to notice that in the experiments where workload varies (dec, inc, osc) the RCT strategy has a better Speedup than Constant scenario. When the workload is constant (con), the best speedup is achieved with RC strategy, enforcing our previous analysis. Hence, we claim our **Observation 1: When the workload is balanced (with no load variation), RC strategy tends to reach a gain of performance over pipeline-structured application**. The efficiency is higher using the RC strategy, the explanation from this behavior comes from the idea that not taking time tasks into consideration, the resource reorganization tends to add and remove VMs faster than RCT strategy. Removing resources happens even more frequently than adding it, because of RCT strategy has a clause that only remove VMs when thresholds and time rules are violated. By keeping more resources, RCT reaches lower Efficiency than RC strategy. That leads us to our **Observation 2: In cases that the workload is unknown, keeping more resources for pipeline-structured applications ensures the processing flow**.

5.2 Analyzing Energy and Cost

Every scenario respects the inequality 8: The application cost not making elastic decisions needs to be equal or lower than applications using elasticity strategies. In the Constant scenario, the application cost is 81% better than scenario without elasticity. The worst case is 57% better than no making elastic actions. In the Decreasing scenario, the best and worst case are 79% and 68% of gain, respectively. In

[2] http://opennebula.org/

[3] https://github.com/viniciusmeyer/closer2019



(a) Energy average (b) Cost average

Figure 3: Energy and Cost results from all scenarios.

the Increasing scenario 83% (b) and 57% (w) and in the Oscillating scenario 90% (b) and 71% (w) of gain compared with a scenario without elasticity strategy.

Figure 3a presents the result of the average's energy from all tests. The result from the energy equation is proportional to resource utilization over time application execution. As we previously mentioned, RCT strategy tends to keep more resources than RC strategy. Because of it, the energy is lower by applying the RC strategy. Furthermore, RCT is the strategy which presents the higher energy utilization index in all cases. Figure 3b presents the result of average's cost from all tests. RC strategy has the lowest cost overhead. The RCT strategy performs the highest cost. This behavior happens because Cost equation is proportional to the energy index, enforcing our previous explanation. After that, we state our **Observation 3: consuming more energy (higher costs) does not always benefit pipeline-structured application's performance**.

5.3 Analyzing Resources Utilization

In order to analyze resource utilization, we have calculated a resource usage percentage during time execution of some experiments: without elasticity (con, dec, inc, osc) and its best (_b) and worst (_w) cases with elasticity. We have chosen the best and worst scenarios based on speedup indexes. Therefore, in the Constant scenario the best performance was achieved by applying the RC strategy and in other scenarios, the best performances were reached utilizing RCT strategy. These percentages can be observed on Figure 4. As shown in Oscillating, Increasing and Decreasing scenarios, in worst cases (_w), higher levels of resource utilization percentage are more scattered over time execution than best scenarios (_b). However, the exact opposite occurs in the Constant scenario: by using more resource over time execution decreases the performance. Here, we state our **Observation 4: when the workload varies, adding more resources benefit pipeline-structured application's performance**.

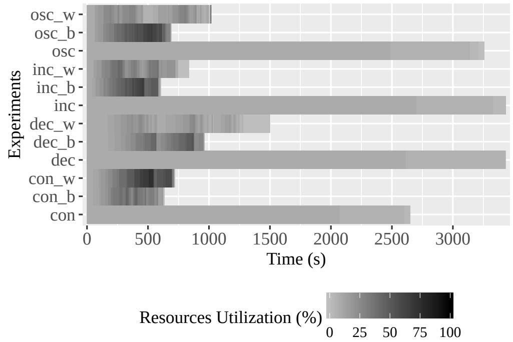


Figure 4: Resources utilization: without elasticity (con, dec, inc, osc) and its best (_b) and worst (_w) elasticity strategies.

6 RELATED WORK

A system for sequential hyperparameter optimization for pipeline tasks was proposed by (Anderson et al., 2017). The main concept is to use sequential Bayesian optimization to explore hyperparameters' space. However, these methods are not scalable, as the entire data science pipeline still must be evaluated on all the data. The author's techniques are able to gain similar performance improvements on three different pipeline tests, but by computing on substantially fewer data.

In order to avoid under or over-provisioning situations caused by strategies which use a fixed number of resources in workflow applications, (Meyer et al., 2019) creates a reactive elasticity model that uses lower and upper load thresholds and the CPU metric to on-the-fly select the most appropriated number of compute nodes for each stage along the pipeline execution. They have executed a set of experiments and as results, this elasticity model presents a gain of 38% in the application time when compared with applications which do not make elastic actions.

Techniques as Auto-Regressive Integrated Moving Average and Recurrent Neural Network–Long Short Term Memory are used for predicting the future workload of based on CPU and RAM usage rate collected from a multi-tier architecture integrated into cloud computing ecosystem (Radhika et al., 2018). After analyzing both techniques, RNN-LSTM deep learning technique gives the minimum error rate and can be applied on large datasets for predicting the future workload of web applications.

To connect a cloud platform-independent model of services with cloud-specific operations, (Alipour and Liu, 2018) presents an open source benchmark application on two cloud platforms to demonstrate the method's accuracy. As a result, the proposed method solves the vendor lock issue by model-to-configuration-to-deployment automation.

7 CONCLUSION AND FUTURE WORK

In this paper, we present two elasticity strategies for pipeline-structured applications to achieve a gain of performance and minimize execution cost. Our techniques monitor pipeline-structured application's metrics and provide elasticity over the entire system in an asynchronous and automatic way, by stage. The RC strategy only considers the CPU load thresholds while RCT strategy uses information from workload as well. In order to evaluate the results, we have conducted many experiments with different threshold combinations and workload variations among them. We have pointed out performance gain and cost reduction is totally dependent on the workload variation. As result, we achieve an average of 72% in performance gain and 73% in cost reduction when comparing non-elastic and elastic executions. When compared with related work, our study improves performance gain in up to 34%.

As future work, we plan to include our elasticity strategies in simulation tools in order to test different machine arrangements.

REFERENCES

Aldossary, M. and Djemame, K. (2018). Performance and energy-based cost prediction of virtual machines auto-scaling in clouds. In *44th Euromicro Conference on Software Engineering and Advanced Applications*.

Alipour, H. and Liu, Y. (2018). Model driven deployment of auto-scaling services on multiple clouds. In *IEEE International Conf. on Soft. Architecture Companion*.

Anderson, A., Dubois, S., Cuesta-infante, A., and Veeramachaneni, K. (2017). Sample, estimate, tune: Scaling bayesian auto-tuning of data science pipelines. In *Int. Conf. on Data Science and Advanced Analytics*.

Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M., and Vahi, K. (2008). Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science*.

Dutta, S., Gera, S., Verma, A., and Viswanathan, B. (2012). Smartscale: Automatic application scaling in enterprise clouds. In *2012 IEEE Fifth International Conference on Cloud Computing*.

Galante, G. and d. Bona, L. C. E. (2012). A survey on cloud computing elasticity. In *IEEE Fifth International Conference on Utility and Cloud Computing*.

Gonzalez-Velez, H. and Cole, M. (2008). An adaptive parallel pipeline pattern for grids. In *IEEE International Symposium on Parallel and Distributed Processing*.

Juve, G. and Deelman, E. (2011). *Scientific Workflows in the Cloud*, pages 71–91. Springer London, London.

Li, J., Humphrey, M., van Ingen, C., Agarwal, D., Jackson, K., and Ryu, Y. (2010). escience in the cloud: A modis satellite data reprojection and reduction pipeline in the windows azure platform. In *IEEE International Symposium on Parallel Distributed Processing*.

Li, Z., Ge, J., Hu, H., Song, W., Hu, H., and Luo, B. (2018). Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds. *IEEE Transactions on Services Computing*.

Liu, J., Qiao, J., and Zhao, J. (2018). Femcra: Fine-grained elasticity measurement for cloud resources allocation. In *IEEE 11th International Conference on Cloud Computing*.

Lorido-Botran, T., Miguel-Alonso, J., and Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*.

Mera-Gómez, C., Bahsoon, R., and Buyya, R. (2016). Elasticity debt: A debt-aware approach to reason about elasticity decisions in the cloud. In *9th International Conference on Utility and Cloud Computing*.

Meyer, V., Righi, R. R., Rodrigues, V. F., Costa, C. A. D., Galante, G., and Both, C. (2019). Pipel: Exploiting resource reorganization to optimize performance of pipeline-structured applications in the cloud. *International J. of Computational Systems Engineering*.

Orgerie, A.-C., de Assuncao, M. D., and Lefevre, L. (2014). A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*

Radhika, E. G., Sadasivam, G. S., and Naomi, J. F. (2018). An efficient predictive technique to autoscale the resources for web applications in private cloud. In *4th Int. Conf. on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics*.

Righi, R. R., Costa, C. A., Rodrigues, V. F., and Rostirolla, G. (2016a). Joint-analysis of performance and energy consumption when enabling cloud elasticity for synchronous hpc applications. *Concurr. Comput. : Pract. Exper.*

Righi, R. R., Rodrigues, V. F., Costa, C. A., Galante, G., de Bona, L. C. E., and Ferreto, T. (2016b). Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *IEEE Transactions on Cloud Computing*.