Hochschule Reutlingen
Reutlingen University

Parallel and Distributed Computing Group
Department of Computer Science
Reutlingen University

Elastic Parallel Systems for High Performance Cloud Computing: State-of-the-Art and Future Directions

Stefan Kehrer, Wolfgang Blochinger
Reutlingen University
{stefan.kehrer, wolfgang.blochinger}@reutlingen-university.de

Manuscript (Preprint)

```
@article{Kehrer2019_PPL,
       author = {Kehrer, Stefan and Blochinger, Wolfgang},
       title = {Elastic Parallel Systems for High Performance Cloud Computing:
       State-of-the-Art and Future Directions},
       journal = {Parallel Processing Letters},
       volume = {29},
       number = {02},
       pages = {1950006},
       year = {2019},
       doi = {10.1142/S0129626419500063},
       URL = {https://doi.org/10.1142/S0129626419500063},
}
```

Preprint of an article published in Parallel Processing Letters Vol. 29, No. 02, 2019, 1950006 (DOI: 10.1142/S0129626419500063) © World Scientific Publishing Company

Journal URL:
https://www.worldscientific.com/worldscinet/ppl

The final publication is available at:
https://doi.org/10.1142/S0129626419500063

Parallel Processing Letters

Elastic Parallel Systems for High Performance Cloud Computing: State-of-the-Art and Future Directions

Stefan Kehrer and Wolfgang Blochinger
Parallel and Distributed Computing Group, Reutlingen University,
Alteburgstr. 150, 72762 Reutlingen, Germany
firstname.lastname@reutlingen-university.de

Received (received date)
Revised (revised date)

ABSTRACT

With on-demand access to compute resources, pay-per-use, and elasticity, the cloud evolved into an attractive execution environment for High Performance Computing (HPC). Whereas elasticity, which is often referred to as the most beneficial cloud-specific property, has been heavily used in the context of interactive (multi-tier) applications, elasticity-related research in the HPC domain is still in its infancy. Existing parallel computing theory as well as traditional metrics to analytically evaluate parallel systems do not comprehensively consider elasticity, i.e., the ability to control the number of processing units at runtime. To address these issues, we introduce a conceptual framework to understand elasticity in the context of parallel systems, define the term elastic parallel system, and discuss novel metrics for both elasticity control at runtime as well as the ex-post performance evaluation of elastic parallel systems. Based on the conceptual framework, we provide an in-depth analysis of existing research in the field to describe the state-of-the-art and compile our findings into a research agenda for future research on elastic parallel systems.

Keywords: Cloud computing; high performance computing; parallel computing; elastic parallel system; elasticity; workload.

1. Introduction

Many scientific disciplines deliver sustained progress based on massive amounts of compute resources that are employed to process highly compute-intensive problems in parallel to speed up their computation. The discipline concerned with constructing parallel applications for these workloads is referred to as High Performance Computing (HPC).

Traditionally, parallel applications have been designed and developed for on-premise HPC clusters and supercomputers. However, more recently, the cloud evolved into an attractive execution environment for parallel applications [1, 2, 3]. As of today, many cloud providers, including Amazon Web Services (AWS)[a] and

[a]https://aws.amazon.com.

Microsoft Azure[b], offer cloud environments optimized for HPC [4, 5]. One cloud-based parallel system built on top of AWS even achieved rank 136 in the TOP500 list[c].

Compared to traditional HPC clusters, cloud environments come with attractive characteristics such as on-demand access to compute resources, pay-per-use, and elasticity [2, 6, 1, 7]. Specifically, elasticity, which is often considered to be the most beneficial cloud-specific property [8, 9], gives rise to a fundamentally new concept in HPC: The ability to control the physical parallelism of an application at runtime, which allows versatile optimizations [10, 1, 11, 12]. Therefore, an elasticity controller, which dynamically adapts the physical parallelism, has to reason about the scaling behavior of a parallel system at runtime to make decisions on the optimal number of processing units. Also, the monetary costs of a parallel computation have to be considered due to the pay-per-use billing model.

In this paper, we argue that existing parallel computing theory as well as traditional metrics to analytically evaluate parallel systems do not comprehensively consider this novel ability to control the number of processing units at runtime. To address these issues, we introduce a conceptual framework to understand elasticity in the context of parallel systems, define the term *elastic parallel system*, and discuss a novel metric called *workload efficiency* to analytically reason about the scaling behavior of a parallel system at runtime. Moreover, we define the performance metrics *elastic speedup* and *elastic efficiency* to enable the ex-post performance evaluation of elastic parallel systems. Based on the conceptual framework, we provide an in-depth analysis of existing research on elasticity in HPC, discuss the key findings, and describe the state-of-the-art with respect to the benefits obtained and the elasticity control mechanisms employed in related work. Finally, we derive future research directions based on our findings and present general research questions that have to be answered for all existing classes of parallel applications.

This paper is structured as follows. Section 2 summarizes the fundamentals required in this work. In Section 3, we present our conceptual framework to understand elasticity in the context of parallel systems. In Section 4, we classify existing research on elastic parallel systems based on the conceptual framework to determine the state-of-the-art. We discuss future research directions in Section 5. In Section 6, we conclude our work.

2. Fundamentals

To lay the ground for our work, we describe existing approaches to elasticity control and two different types of cloud environments to operate parallel applications.

[b]https://azure.microsoft.com.
[c]TOP500 list (June 2019): https://www.top500.org/lists/2019/06/.

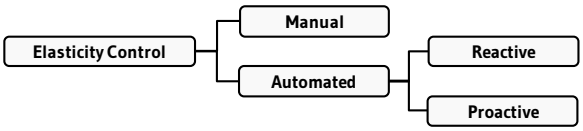

Fig. 1. Approaches to elasticity control adapted from [8]

2.1. *Elasticity & Elasticity Control in the Cloud*

With the emergence of the first cloud offerings, elasticity has been recognized as the most important cloud-specific property because it enables applications to scale dynamically as their workload increases/decreases at runtime [8, 9]. Since then, elasticity has been mainly employed in the context of interactive (multi-tier) applications that are designed to process (independent) user requests [13, 14, 15]. As the arrival rate of user requests increases over time, more compute resources can be provisioned. Similarly, compute resources are decommissioned if the arrival rate decreases. The entity that controls the number of compute resources, e.g., virtual machines (VM), is called *elasticity controller*. Elasticity control can be implemented as a manual or automated process, whereas automation is typically preferred. Automated elasticity control approaches derive scaling actions either by means of reactive or proactive mechanisms [8]. Reactive mechanisms use specified conditions (e.g., thresholds on monitored metrics) to control elasticity. Proactive mechanisms rely on forecasting to adapt a system according to its predicted behavior in the future. Typically, it depends on the characteristics of the application if reactive or proactive mechanisms yield better results. Both approaches might also be combined into a hybrid approach. These different approaches to elasticity control are depicted in Fig. 1. A detailed classification of elasticity mechanisms is discussed in [8, 9].

2.2. *Cloud Environments for HPC*

In principle, two types of cloud environments suitable for parallel computing can be distinguished: Standard and HPC-aware cloud environments [16].

Standard cloud environments often suffer from CPU time sharing leading to heterogeneous processing speeds as well as low network throughput and high network latency, which are well-known effects of resource pooling and virtualization [1, 17]. Whereas interactive applications can deal with these effects, many parallel applications require the use of synchronous communication and/or barrier synchronization, which negatively affects HPC performance in standard cloud environments [18].

As a result of previous research efforts, *HPC-aware cloud environments* have been built, which limit the negative side-effects of standard cloud environments by means of (1) CPU pinning, which ensure CPU affinity both at the application and at the hypervisor level leading to improved cache locality and higher cache hit rates [19], (2) HPC-aware VM placement policies [16], (3) guaranteed network performance [20], and (4) disabled memory overcommitment [21].

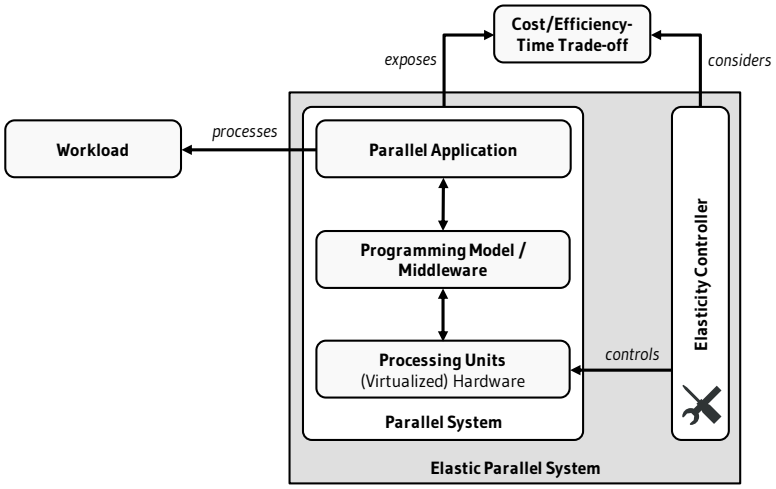

Fig. 2. Conceptual framework for understanding elasticity in the context of parallel systems

With the introduction of HPC-aware cloud environments, more parallel application classes can benefit from on-demand access to compute resources and a pay-per-use billing model. Moreover, elasticity can be employed. However, note that HPC-aware cloud environments are typically much more expensive compared to standard cloud environments. This mainly results from the specialized hardware employed and the limited opportunities of the cloud provider to operate multiple, diverse customer workloads across the physical infrastructure (multitenancy) [19, 22]. Moreover, VM placement policies limit the provider's VM placement options.

3. Elastic Parallel Systems

In this section, we introduce a conceptual framework to understand elasticity in the context of parallel systems, which we employ to review and discuss related research in Section 4 and to derive future research directions in Section 5. The conceptual framework is visualized in Fig. 2. It shows an *elastic parallel system*, which we define as a parallel system[d] accompanied by an elasticity controller that controls the number of processing units at runtime. Therefore, the elasticity controller reasons about the scaling behavior of the parallel system to make decisions on the optimal number of processing units. Existing parallel computing theory does not comprehensively consider this novel ability to control the number of processing units at runtime. For instance, traditional metrics to analytically evaluate parallel systems such as parallel efficiency assess the performance averaged over the entire execu-

[d] A parallel system is a combination of a parallel algorithm (parallel application, programming model / middleware) and a parallel architecture (hardware) [23].

tion time [23, 24, 25]. By definition, parallel efficiency can only be calculated, when the computation has been completed as it depends on the parallel execution time. This also holds for the speedup. Additionally, the definitions of parallel efficiency and speedup are based on a fixed number of processing units p. Elastic parallel systems, on the other hand, are characterized by a dynamic number of processing units, which can be adapted at runtime: Thus, the number of processing units is a function of time $p(t)$ that is defined by the elasticity controller. As a result, existing metrics to analytically evaluate parallel systems cannot be used by the elasticity controller to reason about the scaling behavior of a parallel system at runtime. The conceptual framework described in the following provides the fundamental concepts to discuss the opportunities and challenges that arise in the context of elasticity. On this basis, we discuss a novel metric called *workload efficiency* to analytically reason about the scaling behavior of a parallel system at runtime. Moreover, we define the performance metrics *elastic speedup* and *elastic efficiency* to enable the ex-post evaluation of elastic parallel systems.

Traditionally, elasticity has been considered in the context of interactive (multi-tier) applications, i.e., applications that process independent user requests. Thus, the workload has been described by the user requests processed by an application [14]. Whenever the arrival rate of user requests varies over time, these applications benefit from elasticity by dynamically adapting the number of processing units to control the response time [14]. On the other hand, the workload of parallel systems cannot be described as independent user requests. Instead, it depends on a single problem, which requires a specific number of *(basic) computational steps* to be solved [23]. Moreover, parallel overhead occurs in the manifestation of idling, communication, and excess computation. To model parallel overhead in our framework, we distinguish between three different types of steps that can be executed by a processing unit: (1) *essential* computational steps, (2) *non-essential* computational steps, and (3) *idling* steps. All sources of overhead can be modeled as one (or more) of these types. For instance, communication and synchronization of data across distributed processing units leads to idling steps and additional computational steps that are not executed by a sequential algorithm and thus can be classified as non-essential.

Under this consideration, we define the workload of a parallel system as follows:

Definition 3.1. Workload
The workload $W(I)$ of a parallel system is given by the total number of *essential* computational steps executed based on a defined input I.

The input I describes a specific problem that has to be solved and the workload is given by the total number of essential computational steps required to solve the problem. Note that, under the assumption that it takes unit time to perform a single computational step, the workload is equivalent to the sequential runtime T_{seq}. Note also that our definition of workload is equal to the definition of the *problem size* as typically considered in parallel computing theory [23].

With elasticity, one has the ability to adapt the number of processing units, which are employed to process a specific problem, at runtime. This adaptation changes the physical parallelism of a parallel system, which we model as the processing rate.

Definition 3.2. Processing Rate
The processing rate $R(p(t), t)$ of a parallel system describes the total number of steps that can be executed (by all employed processing units) in time interval t.

The elasticity controller can dynamically adapt the processing rate of a parallel system by adding or removing processing units (cf. Fig. 2). Additionally, the processing rate is affected by every change of an individual processing rate (of a single processing unit): Individual processing rates might change over time due to virtualization and resource pooling.

The major motivation for parallel processing is to shorten the application's execution time by adding more processing units to the computation. However, due to the inherent overhead, one cannot assume linear scalability. Instead, the elasticity controller has to consider a *cost/efficiency-time trade-off*: Whereas adding more processing units effectively reduces the execution time, a higher number of processing units also leads to a lower efficiency and thus results in higher monetary costs [10, 12]. This leads to two conflicting optimization goals: (1) Reduce monetary costs & maximize efficiency vs. (2) shorten execution time. In the following, we discuss the cost/efficiency-time trade-off inherent to parallel systems in more detail by comparing them to an (ideal) perfectly scalable parallel system.

For a perfectly scalable parallel system, the monetary costs to process the total number of essential computational steps are independent of the number of processing units employed[e]. Thus, in theory, an unbounded number of processing units can be employed to speed up the computation. Realistic parallel systems, on the other hand, are not perfectly scalable. Thus, whereas employing more processing units reduces the execution time, it also leads to higher overhead (and thus lower efficiency) [26]. In cloud environments, one has to pay processing units (or compute resources in general) per time unit. As a consequence, one does not only pay for the essential computational steps executed but also for the overhead that occurs (in form of non-essential computational steps and idling steps).

The monetary costs of a parallel computation in the cloud C_{par} can be modeled as follows [18]:

$$C_{par}(I, p(t)) = T_{par}(I, p(t)) \cdot \bar{p} \cdot c_{\pi}, \tag{1}$$

where T_{par} is the execution time, $\bar{p}$ is the time-averaged number of processing units employed, and c_{π} the price of one processing unit per time unit.

[e]This only holds under the assumption that there is also a perfect mapping of the processing demand (in form of essential computational steps) to processing units and zero provisioning costs.

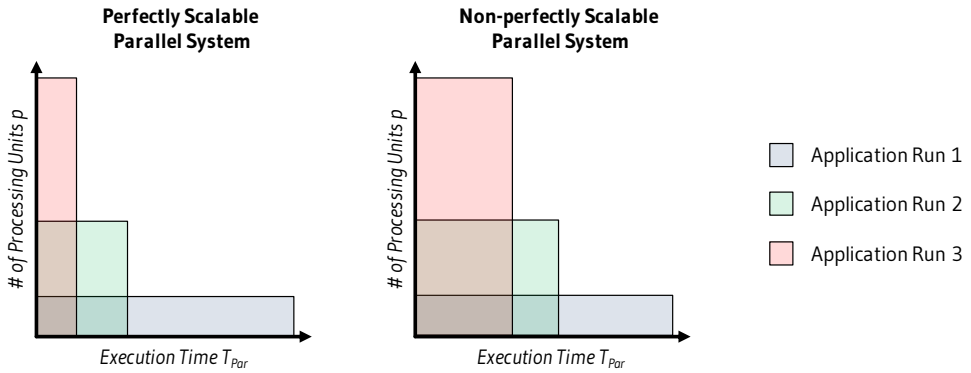

Fig. 3. Whereas the monetary costs of a perfectly scalable parallel system are independent of the number of processing units employed, for parallel systems that are not perfectly scalable, the monetary costs increase with the number of processing units. Here, the monetary costs are expressed as the sizes of the areas shown.

Fig. 3 depicts three different application runs with different numbers of processing units for a perfectly and a non-perfectly scalable parallel system. The areas shown for each application run correspond to the monetary costs of a parallel computation in the cloud C_{par} (cf. Equation 1). As we can easily see, the areas shown for the perfectly scalable parallel system all have the same size, whereas the sizes of the areas shown for the non-perfectly scalable parallel system increase with an increasing number of processing units leading to higher monetary costs. This results from the overhead that increases with the number of processing units [23]. How much overhead occurs for which number of processing units depends on the scaling behavior of the parallel system considered.

To dynamically adapt the number of processing units, the elasticity controller has to reason about the cost/efficiency-time trade-off at runtime. Therefore, we introduce the *workload efficiency* - a novel metric to analytically reason about the scaling behavior of a parallel system at runtime. In the following, we introduce the *workload intensity* based on which the workload efficiency can be defined.

Definition 3.3. Workload Intensity
The workload intensity $WI(I, p(t), t)$ of a parallel system describes the total number of essential computational steps executed in time interval t.

The workload intensity quantifies the amount of useful work executed by all employed processing units as a function of time. We adopt the term *workload intensity* from [14], where it has been used in the context of interactive (multi-tier) applications. Based on the processing rate and the workload intensity, we define the workload efficiency as follows:

Definition 3.4. Workload Efficiency
The workload efficiency $WE(I, p(t), t)$ of a parallel system describes the ratio of the workload intensity to the processing rate: $WE(I, p(t), t) = \frac{WI(I,p(t),t)}{R(p(t),t)}$

Note that the definition of workload efficiency differs from the common notion of utilization, which we define in our terms as follows:

Definition 3.5. Utilization
The utilization $U(I, p(t), t)$ of a parallel system describes the ratio of the total number of essential and non-essential computational steps executed in time interval t to the processing rate.

Utilization does not distinguish between essential and non-essential computational steps, with the latter stemming from overhead, e.g., in form of excess computation.

Whenever the workload efficiency changes at runtime, the elasticity controller has to evaluate the cost/efficiency-time trade-off. Such changes result from (1) a changing workload intensity, (2) a changing processing rate, (3) a combination of both. Under the assumption that the processing rate is constant, a changing workload intensity is typically related to the parallel algorithm, the input data considered (problem to be solved), and the current state of the computation: Excess computation and communication overhead are considered in form of non-essential computational steps and idling steps, which both affect the workload intensity. Moreover, changing characteristics of the execution environment (e.g., network latency and network bandwidth) affect the workload intensity by introducing additional overhead. A changing processing rate is typically related to a changing number of processing units or changing processing rates of individual processing units (i.e., varying processing speed due to virtualization and resource pooling).

Fig. 4 visualizes examples of the three aforementioned scenarios. Fig. 4 (a) shows a scenario, where the processing rate is constant, but the workload intensity decreases as the execution time increases. This is related to growing overhead, which makes it harder to efficiently exploit all available processing units. Fig. 4 (b) shows a scenario, where the processing rate doubles because the elasticity controller provisions additional processing units, but the workload intensity does not change. This could be the result of not sending work to the newly provisioned processing units, which consequently do not contribute to the computation. Fig. 4 (c) shows a scenario, where the processing rate doubles because the elasticity controller provisions additional processing units. Because more processing units contribute to the computation, the workload intensity also increases. However, it does not double because the parallel system is not perfectly scalable. In all three cases, the workload efficiency changes at runtime and thus the elasticity controller has to dynamically evaluate the cost/efficiency-time trade-off.

To evaluate the cost/efficiency-time trade-off, the elasticity controller combines knowledge about efficiency, monetary costs, and time. The workload efficiency metric enables the elasticity controller to analytically reason about the scaling behavior of the parallel system at runtime. Time can be considered as wall-clock time. The current monetary costs per time unit can be calculated by multiplying the current number of processing units with c_π (cf. Equation 1). Whether the execution time

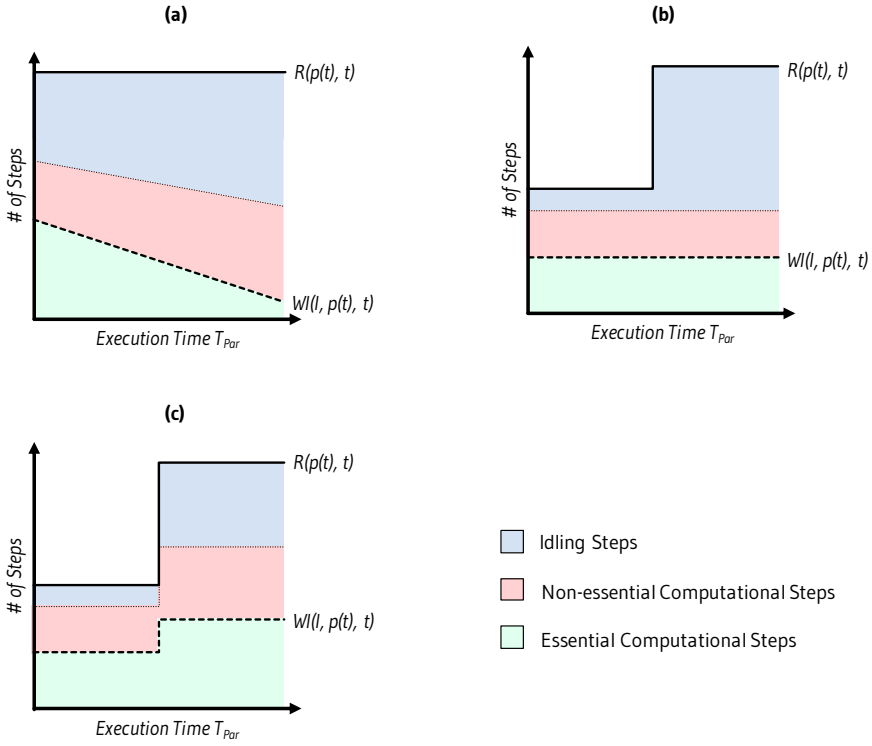

Fig. 4. Different scenarios of workload efficiency changes at runtime.

T_{par} and thus also the monetary costs C_{par} can be predicted or not largely depends on the application class considered. We discuss several examples how to deal with the cost/efficiency-time trade-off in case of a changing changing processing rate and / or a changing workload intensity based on existing research in Section 4.

As discussed initially, traditional performance metrics such as speedup S and efficiency E are functions of a specific input I and a fixed number of processing units p. As a result, these metrics cannot be used for ex-post evaluation of elastic parallel systems. To deal with this issue, the authors of [27] define elastic speedup and elastic efficiency as functions of (1) an initial number of VMs, (2) a lower bound for the quantity of VMs, and (3) an upper bound for the quantity of VMs. They use these metrics to compare application runs with and without elastic scaling. However, these definitions are tailored to their implementation of a threshold-based elasticity controller and not generally applicable. The authors of [28] also provide a definition of elastic speedup, but focus on High Throughput Computing (HTC) and performance in terms of response time. In the following, we provide general definitions of elastic speedup and elastic efficiency based on our former discussions.

We define **elastic speedup** $S_{elastic}$ and **elastic efficiency** $E_{elastic}$ analogously to speedup S and parallel efficiency E. However, whereas speedup and parallel

efficiency are functions of the number of processing units p, elastic speedup and elastic efficiency are functions of $p(t)$:

$$S_{elastic}(I, p(t)) = \frac{T_{seq}(I)}{T_{par}(I, p(t))} \tag{2}$$

$$E_{elastic}(I, p(t)) = \frac{S_{elastic}(I, p(t))}{\bar{p}} = \frac{T_{seq}(I)}{T_{par}(I, p(t)) \cdot \bar{p}}, \tag{3}$$

where $\bar{p}$ is the time-averaged number of processing units employed.

Note that, for a constant function $p(t) = p = \bar{p}$, elastic speedup $S_{elastic}$ and elastic efficiency $E_{elastic}$ are equal to speedup S and efficiency E. We can also say that speedup and efficiency do only consider a single time interval $[0, T_{par}]$, where $\bar{p} = p$.

4. State-of-the-Art

In this section, we review existing research on elastic parallel systems to determine the state-of-the-art. To identify relevant research, we conducted a literature search by following a systematic procedure [29]. For the search process, we employed the ACM Digital Library[f], IEEE Xplore[g], and Google Scholar[h]. The following search query was used for the search: *cloud AND (elasticity OR elastic application OR auto-scaling OR auto scaling) AND (high performance computing OR high-performance computing OR hpc OR parallel application OR parallel system OR parallel computing)*. Subsequently, we manually selected relevant articles: Only articles that discuss in detail how to employ elasticity in the context of parallel systems were selected. Additional literature has been identified (1) by reviewing the references of the selected articles and (2) by analyzing articles that cite the already selected ones [30]. Table 1 summarizes the selected articles. We discuss the key findings in the following.

Da Rosa Righi et al. [11] describe a reactive elasticity control mechanism for iterative-parallel applications. The presented concept called AutoElastic supports the automated transformation (source-to-source translation) of existing MPMD[i]-based MPI-2 applications with a master/worker architecture into elastic parallel applications. MPI-2 features dynamic process management and thus supports a varying number of MPI processes [31]. Consequently, elasticity can be employed by adapting the number of worker processes for each iteration by means of an elasticity controller that derives the number of worker processes required based on monitoring data and defined thresholds. The presented concepts are evaluated by using a numerical integration application which simulates different workload patterns (e.g., ascending, descending, and wave workload). The authors state that their approach

[f] https://dl.acm.org.
[g] https://ieeexplore.ieee.org.
[h] https://scholar.google.com.
[i] Multiple Program Multiple Data.

is specifically designed for embarrassingly parallel applications. Scaling actions are based on upper and lower CPU utilization thresholds. Note that because the elasticity controller monitors only the utilization of the parallel system, it cannot distinguish between essential and non-essential computations. The monetary costs are not considered. Further research is required to determine which application classes can benefit from such an approach. Similar concepts are discussed by the authors in [32].

Table 1. Classification of existing research on elastic parallel systems.

Authors & Article	**Cloud Environment**	**Application Class**	**Optimization Objective**	**Elasticity Control**
Da Rosa Righi et al. [11]	Standard	Iterative Master-Slave	Utilization	Reactive
Da Rosa Righi et al. [32]	Standard	Iterative Master-Slave	Utilization	Reactive
Rodrigues et al. [33]	Standard	Iterative Master-Slave	Utilization	Hybrid
Da Rosa Righi et al. [34]	Standard	Iterative Master-Slave	Utilization	Hybrid
Shankar et al. [35]	Standard	Linear Algebra	Costs & Time (Scaling Policy)	Reactive
Raveendran et al. [36]	Standard	Iterative MPI	Time (Costs implicitly considered)	Reactive
Rajan et al. [10]	Standard	Split-Map-Merge	Costs & Time	Reactive
Rajan et al. [10]	HPC-aware	Split-Map-Merge	Costs & Time	None
Haussmann et al. [12]	Standard	Irregular Task-parallel	Costs (Time as Opportunity Costs)	Reactive

Rodrigues et al. [33] present a hybrid elasticity controller for iterative-parallel applications based on a technique called live thresholding, which has also been used in [34]. Live thresholding dynamically adapts the thresholds of a reactive elasticity controller, which is implemented as a closed feedback-loop architecture. Workload patterns are detected by comparing the last two average load values calculated based on monitored time series data and simple exponential smoothing. Similarly to the evaluation presented in [11], only the utilization of processing units is monitored and thus the elasticity controller cannot distinguished between essential and non-essential computations. The cost/efficiency-time trade-off is not considered.

Shankar et al. [35] discuss the opportunities related to elasticity in the context of a parallel application for distributed Cholesky decomposition. As the execution time increases it becomes harder to efficiently employ processing units thus overhead

increases and the workload intensity decreases. The authors argue that static resource allocation is inefficient in this case and present an architecture for non-trivial parallel processing in serverless (standard) cloud environments accomplished with a reactive elasticity control (auto-scaling) mechanism. The scaling policy underlying the elasticity control mechanism considers a trade-off between fast execution and low monetary costs and can be manually selected upfront. The resulting execution engine called *numpywren* specifically targets linear algebra workloads that should be operated on serverless computing platforms and has been shown to be more cost-efficient compared to other frameworks.

Raveendran et al. [36] propose a concept to transform MPI-based iterative-parallel applications into elastic applications. They describe how to adapt existing MPI-based applications to deal with a dynamically changing number of processing units. The presented approach basically terminates a running application (with check-pointing) and restarts the application with a different number of processing units. Termination can only be applied at the end of an iteration. The described elasticity controller is designed to optimize several user-defined constraints such as the desired execution time. The elasticity controller estimates the execution time of the application based on the number of iterations and the average iteration time. The underlying assumption is that the amount of work per iteration is constant. Scaling decisions are made by comparing the measured average iteration time with the required iteration time to complete within the user-defined execution time: If the average iteration time is below the required iteration time, processing units are added. Otherwise, processing units are removed. The presented approach is built upon the assumption that, in general, a lower number of processing units can be exploited more efficiently and thus also incur lower monetary costs to process a problem (as discussed in Section 3 and shown in Fig. 3). Consequently, overhead is not directly measured but the cost/efficiency-time trade-off is implicitly considered by scaling down when the user-defined execution time can also be met with less compute resources.

Rajan et al. [10] specifically address applications that can be developed and executed using the so-called *split-map-merge* paradigm. According to the authors this includes bag-of-tasks, bulk synchronous parallel, and Map-Reduce applications. The authors consider the cost-time product as objective function to evaluate the cost/efficiency-time trade-off. Therefore, the automated decision making process considers (1) information on the input problem, (2) an application model built for split-map-merge applications, (3) a user-defined objective function (in this case the cost-time product as a function of the number of processing units), and (4) information on the execution environment (e.g., processing speed, network bandwidth) obtained by measuring sample workloads. Note that this approach can be utilized in two different ways: First, one can statically select the optimal number of processing units per application run under the assumption that the characteristics of the environment do not change at runtime (e.g., by selecting an HPC-aware cloud environment). This approach is depicted in Fig. 5. Second, one can continuously evaluate

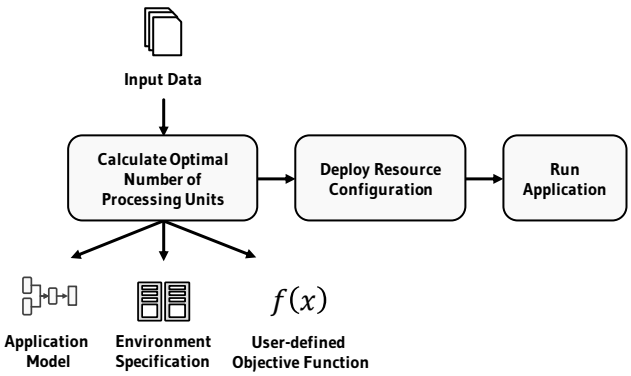

Fig. 5. The authors of [10] present an approach for fine-grained cost control per application run by determining the optimal number of processing units based on an application model, information on the execution environment, a user-defined objective function, and problem-specific input data.

the objective function. For instance, when the characteristics of the cloud environment (e.g., processing speed, network bandwidth) are expected to change at runtime, an elasticity controller monitors the environment and continuously evaluates the objective function based on monitoring data to consider the cost/efficiency-time trade-off. When the optimal (with respect to the user-defined objective function) number of processing units changes, the elasticity controller dynamically adapts the resource configuration by adding or removing processing units.

Haussmann et al. [12] discuss elasticity-related opportunities and challenges for irregular task-parallel applications. Their computation and communication patterns are input-dependent, unstructured, and evolving during the computation and thus their scaling behavior cannot be determined upfront [37, 38]. As a result, the total number of essential computational steps (workload) is unknown a priori and cannot be predicted. The authors discuss the two conflicting objectives of fast processing and low monetary costs finally leading to a multi-objective optimization problem and Pareto optimal solutions, which prevents automated decision making with respect to the number of processing units. To deal with this problem, the authors employ the concept of opportunity costs to convert the underlying objective functions into a single aggregated objective function, thus allowing cost-based selection of the number of processing units. Because one cannot reason about the effects on execution time, speedup, efficiency, and monetary costs in absolute terms, the authors present a reactive elasticity controller for heuristic cost optimization: The cost function is approximated based on metrics monitored at runtime. Therefore, the elasticity controller continuously monitors the application and evaluates the defined objective function (minimize the monetary costs based on the presented cost model). The authors empirically evaluate their elasticity controller by comparing the results of their heuristic cost optimization approach with the minimum monetary costs (which can be obtained by measuring the scalability of the application with exemplary input problems).

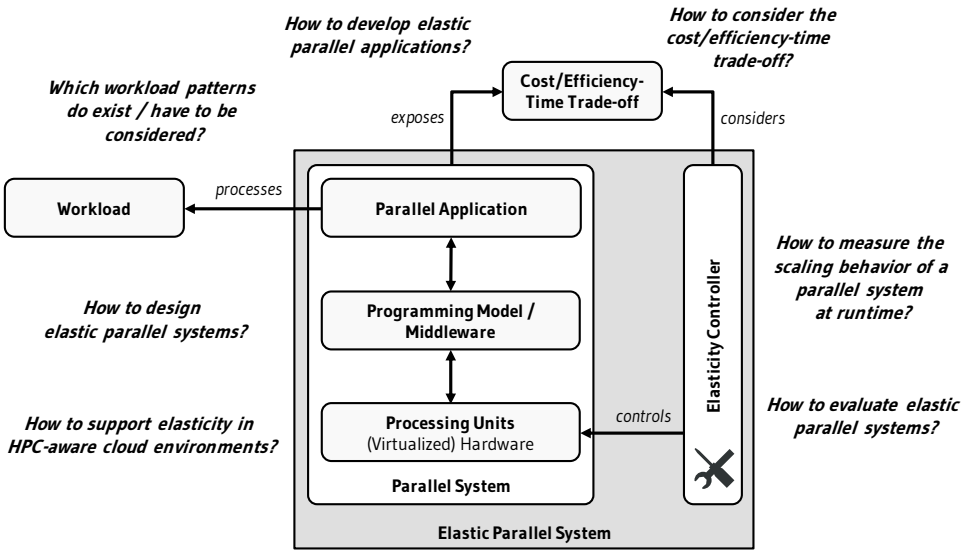

Fig. 6. Research questions for future research on elastic parallel systems

Whereas the scaling behavior of applications based on the split-map-merge paradigm can be predicted by using an application model [10], the scaling behavior of irregular task-parallel applications is unknown upfront and unpredictable by nature, which requires reactive elasticity control [12]. On the other hand, both approaches convert the underlying multi-objective optimization problem into a single-objective optimization problem to enable automated decision making with respect to the optimal number of processing units. The authors of [10] use the cost-time product to create a single-objective optimization problem. The authors of [12] employ the concept of opportunity costs, which can be used to express time in terms of costs, thus enabling a purely cost-driven optimization.

5. Future Research Directions

Whereas we could identify some approaches that show how to beneficially exploit elasticity for operating parallel applications in the cloud, more research is required to understand the full potential of this fundamentally new opportunity and to address the related challenges on all levels of parallel systems. To pave the way for future research activities, we compile our findings into a research agenda, which is presented in form of general research questions that have to be answered for all existing classes of parallel applications. Fig. 6 shows these research questions in the context of our conceptual framework.

How to consider the cost/efficiency-time trade-off? A major challenge for making parallel applications elastic is the cost/efficiency-time trade-off discussed in Section 3. We found different optimization approaches in existing work that

consider this trade-off in the context of specific application classes. However, as of today, a clear and generally applicable understanding of this trade-off does not exist. An important consideration in this context is the balancing of various sources of overhead: Whereas it might be reasonable to adapt the number of processing units at runtime, corresponding adaptation efforts (e.g., data transfer and repartitioning) finally lead to additional overhead that has to be considered. Another important issue is the overhead added by resource pooling and virtualization, which affects the processing rate of a parallel system and thus also influences the cost/efficiency-time trade-off. Note that such effects can lead to a changing workload efficiency even if the workload intensity and the number of processing units do not change.

How to measure the scaling behavior of a parallel system at runtime? To enable elasticity control for parallel systems, the scaling behavior has to be measured at runtime. Consequently, the elasticity controller requires up-to-date monitoring data based on runtime metrics that allow reasoning about all existing sources of overhead and their effects on execution time, efficiency, and monetary costs. We proposed the workload efficiency as a runtime metric to analytically reason about the scaling behavior and the cost/efficiency-time trade-off. Technically, code-level instrumentation is required to generate monitoring data, which includes an important trade-off: Whereas fine-granular instrumentation improves the accuracy of monitored metrics, it also introduces additional overhead that might outweigh the utility. On the other hand, coarse-gained instrumentation has a low overhead but might fail in reporting the actual system state leading to wrong decisions. Furthermore, for each metric, the monitoring interval has to be carefully defined. Choosing smaller monitoring intervals enables fast decision making and a high accuracy, but at the expense of additional overhead. Also note that the workload efficiency has to be interpreted with care. For instance, the currently measured workload efficiency might be extremely low because it has been measured during a global communication phase. Thus, simply removing processing units based on a low measured workload efficiency might not be appropriate. Rather, monitoring data has to be combined with knowledge about the parallel algorithm, the input data, the parallel system architecture, and the current state of the computation to reason about the optimal number of processing units.

Which workload patterns do exist / have to be considered? In the context of parallel systems, we understand workload patterns as common shapes of the workload intensity function (with a static number of processing units). Understanding the workload patterns of different application classes will be key to enable advanced elasticity control mechanisms. As of today (cf. Section 4), most often a decreasing workload pattern, which implies that overhead increases over time, has been exploited to benefit from elasticity. In such cases, elasticity can be employed to remove processing units as the execution time increases thus leading to a higher efficiency and lower monetary costs. Recent work also shows how parallel applications with an unpredictable scaling behavior can benefit from elasticity: The shape of their workload intensity function is unknown and cannot be predicted. As

a result, only two strategies to statically select the number of processing units can be applied: (1) Always use the same number of processing units for all processed problems or (2) randomly guess the number of processing units per application run. Both approaches are not satisfactory. These applications benefit from dynamically adapting the number of processing units by means of an elasticity controller that is able to reason about the cost/efficiency-time trade-off at runtime. Future research is required to determine the workload patterns of other parallel application classes.

How to design elastic parallel systems? From an architectural perspective, parallel systems (and applications) have been extensively researched and the design process is well understood [23]. Corresponding design knowledge is also captured in pattern languages for parallel computing that support the creative process of designing such systems. For instance, the OPL[j] pattern language [39, 40] supports pattern-based parallel design. However, to make use of elasticity, parallel systems are accompanied by an elasticity controller that dynamically adapts the number of processing units (cf. Fig. 6). Dealing with a dynamically changing number of processing units leads to new requirements for traditionally designed parallel systems [41, 42]. For instance, established techniques such as static task mapping have been shown to deteriorate the performance of parallel applications operated in standard cloud environments [43, 42]. More research is required to understand the impact of elasticity on established design processes and parallel architectures. Specifically, distributed memory architectures are of interest for horizontal scaling, which is the predominant scaling approach to make use of elasticity [8].

How to develop elastic parallel applications? Traditionally, runtime systems, programming models, and development frameworks for parallel applications have been designed for a static number of processing units per application run. An elastic parallel application, on the other hand, has to be able (1) to make use of newly provisioned processing units and (2) to release existing processing units at runtime. This requirement leads to major implications, e.g., for task generation and mapping. Whereas MPI-2, for instance, supports dynamic process management and thus a varying number of MPI processes [31], dynamic task generation and mapping have to be manually implemented. To develop elastic parallel applications with only minor effort at the programming level, a new breed of programming models and development frameworks is required. We envision cloud-aware middleware solutions on top of which elastic parallel applications can be implemented by employing higher-level programming models and development frameworks. Another important aspect is the consideration of the cost/efficiency-time trade-off, which typically requires user-specific configuration. To abstract from low-level technical details, solutions optimized for specific parallel application classes have to be investigated.

How to evaluate elastic parallel systems? In existing research, elastic parallel systems have been compared based on different dimensions such as monetary costs, energy consumption, or performance. On the other hand, existing work in the

[j] https://patterns.eecs.berkeley.edu.

context of interactive (multi-tier) applications proposes novel elasticity metrics to evaluate elasticity controllers [44, 45, 46]. These metrics are designed to evaluate and compare the elastic behavior of systems, i.e., how good resource demand and resource supply are mapped with respect to accuracy and timing [45]. The authors of [45] show how to aggregate elasticity metrics to compare a baseline platform with a benchmarked platform. We could not identify similar approaches applied to elasticity control mechanisms in the HPC domain.

How to support elasticity in HPC-aware cloud environments? HPC-aware cloud environments typically allow the specification of placement groups to co-locate virtual machines, which enables low latency network links (cf. Section 2.2). However, it is recommended to start all instances within a placement group at the same time. Whereas instances can be added at runtime (by leveraging elasticity), it depends on the current hardware capacity if these instances can be added to the existing placement group. Consequently, more research on elasticity support in HPC-aware cloud environments is required. In this context, also the economic aspects for cloud providers have to be considered.

6. Conclusion

In this paper, we extend existing parallel computing theory to consider elasticity, i.e., the ability to control the number of processing units of a parallel system at runtime. We introduce a conceptual framework to understand elasticity in the context of parallel systems, define the term elastic parallel system, and discuss novel metrics for both elasticity control at runtime as well as the ex-post performance evaluation of elastic parallel systems. Moreover, we provide a systematic analysis of existing research in the field and discuss existing approaches to elasticity control for parallel systems. We argue that a deep understanding of elasticity-related opportunities and related control mechanisms will be fundamentally required in the future. This is not only backed by HPC-aware cloud offerings. We can also see the adoption of cloud-specific properties and related operational principles in supercomputing, which seems to be a reasonable evolution of traditional environments.

Among the novel opportunities related to elastic parallel systems, also many research challenges have to be addressed. For instance, elasticity control mechanisms that consider the scaling behavior of parallel systems are required to dynamically adapt the number of processing units under consideration of scarce resources such as time and money. Whereas recent work addresses some of the challenges related to elasticity, we identified several research opportunities presented in form of general research questions that have to be answered for all existing classes of parallel applications. On the other hand, also interactive (multi-tier) applications might benefit from considering the cost/efficiency-time trade-off. As of today, because each user request can be processed independently, these applications are considered to be trivial parallel and the cost/efficiency-time trade-off is not considered.

References

[1] G. Galante, L. C. Erpen De Bona, A. R. Mury, B. Schulze and R. da Rosa Righi, An analysis of public clouds elasticity in the execution of scientific applications: a survey, *Journal of Grid Computing* **14** (Jun 2016) 193–216.

[2] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha and R. Buyya, Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges, *ACM Computing Surveys (CSUR)* **51** (January 2018) 8:1–8:29.

[3] S. Kehrer and W. Blochinger, Taskwork: A cloud-aware runtime system for elastic task-parallel hpc applications, in *Proceedings of the 9th International Conference on Cloud Computing and Services Science* (SciTePress, 2019) 198–209.

[4] J. Zhang, X. Lu and D. K. D. Panda, Designing locality and numa aware mpi runtime for nested virtualization based hpc cloud with sr-iov enabled infiniband, in *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments VEE '17*, (ACM, New York, NY, USA, 2017) 187–200.

[5] R. Aljamal, A. El-Mousa and F. Jubair, A comparative review of high-performance computing major cloud service providers, in *2018 9th International Conference on Information and Communication Systems (ICICS)* April 2018 181–186.

[6] B. Varghese and R. Buyya, Next generation cloud computing: New trends and research directions, *Future Generation Computer Systems* **79** (2018) 849–861.

[7] C. Vecchiola, S. Pandey and R. Buyya, High-performance cloud computing: A view of scientific applications, in *10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)* IEEE2009 4–16.

[8] G. Galante and L. C. E. d. Bona, A survey on cloud computing elasticity, in *2012 IEEE Fifth International Conference on Utility and Cloud Computing* Nov 2012 263–270.

[9] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah and P. Merle, Elasticity in cloud computing: State of the art and research challenges, *IEEE Transactions on Services Computing* **11** (March 2018) 430–447.

[10] D. Rajan and D. Thain, Designing self-tuning split-map-merge applications for high cost-efficiency in the cloud, *IEEE Transactions on Cloud Computing* **5** (April 2017) 303–316.

[11] R. da Rosa Righi, V. F. Rodrigues, C. A. da Costa, G. Galante, L. C. E. de Bona and T. Ferreto, Autoelastic: Automatic resource elasticity for high performance applications in the cloud, *IEEE Transactions on Cloud Computing* **4** (Jan 2016) 6–19.

[12] J. Haussmann, W. Blochinger and W. Kuechlin, Cost-efficient parallel processing of irregularly structured problems in cloud computing environments, *Cluster Computing* (Dec 2018).

[13] C. Fehling, F. Leymann, R. Retter, W. Schupeck and P. Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications* (Springer Publishing Company, Incorporated, 2014).

[14] N. R. Herbst, S. Kounev and R. Reussner, Elasticity in cloud computing: What it is, and what it is not, in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)* (USENIX, San Jose, CA, 2013) 23–27.

[15] D. Moldovan, G. Copil, H. Truong and S. Dustdar, Mela: Monitoring and analyzing elasticity of cloud services, in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science* **1**2013 80–87.

[16] A. Gupta, L. V. Kal, D. Milojicic, P. Faraboschi and S. M. Balle, Hpc-aware vm placement in infrastructure clouds, in *2013 IEEE International Conference on Cloud Engineering (IC2E)* March 2013 11–20.

[17] X. Yang, D. Wallom, S. Waddington, J. Wang, A. Shaon, B. Matthews, M. Wil-

son, Y. Guo, L. Guo, J. D. Blower, A. V. Vasilakos, K. Liu and P. Kershaw, Cloud computing in e-science: research challenges and opportunities, *The Journal of Supercomputing* **70** (Oct 2014) 408–464.

[18] A. Gupta and D. Milojicic, Evaluation of hpc applications on cloud, in *2011 Sixth Open Cirrus Summit* Oct 2011 22–26.

[19] A. Gupta, L. V. Kale, F. Gioachin, V. March, C. H. Suen, B. S. Lee, P. Faraboschi, R. Kaufmann and D. Milojicic, The who, what, why, and how of high performance computing in the cloud, in *IEEE 5th International Conference on Cloud Computing Technology and Science* **1**Dec 2013 306–314.

[20] V. Mauch, M. Kunze and M. Hillenbrand, High performance cloud computing, *Future Generation Computer Systems* **29**(6) (2013) 1408 – 1416.

[21] K. B. Ferreira, P. Bridges and R. Brightwell, Characterizing application sensitivity to os interference using kernel-level noise injection, in *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis* Nov 2008 1–12.

[22] A. Gupta, P. Faraboschi, F. Gioachin, L. V. Kale, R. Kaufmann, B. Lee, V. March, D. Milojicic and C. H. Suen, Evaluating and improving the performance and scheduling of hpc applications in cloud, *IEEE Transactions on Cloud Computing* **4** (July 2016) 307–321.

[23] A. Grama, A. Gupta, G. Karypis and V. Kumar, *Introduction to Parallel Computing*, second edn. (Pearson Education, 2003).

[24] L. E. Jordan and G. Alaghband, *Fundamentals of Parallel Processing* (Prentice Hall Professional Technical Reference, 2002).

[25] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods* (Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989).

[26] D. L. Eager, J. Zahorjan and E. D. Lazowska, Speedup versus efficiency in parallel systems, *IEEE Transactions on Computers* **38** (March 1989) 408–423.

[27] R. Rosa Righi, C. A. Costa, V. F. Rodrigues and G. Rostirolla, Joint-analysis of performance and energy consumption when enabling cloud elasticity for synchronous hpc applications, *Concurrency and Computation: Practice & Experience* **28** (April 2016) 1548–1571.

[28] D. Chandler, N. Coskun, S. Baset, E. Nahum, S. R. M. Khandker, T. Daly, N. W. I. Paul, L. Barton, M. Wagner, R. Hariharan *et al.*, Report on cloud computing to the osg steering committee, *Tech. Rep.* (2012).

[29] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner and M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, *Journal of Systems and Software* **80**(4) (2007) 571 – 583.

[30] J. Webster and R. T. Watson, Analyzing the past to prepare for the future: Writing a literature review, *MIS Quarterly* **26**(2) (2002) xiii–xxiii.

[31] W. Gropp, R. Thakur and E. Lusk, *Using MPI-2: Advanced features of the message passing interface* (MIT press, 1999).

[32] R. da Rosa Righi, V. F. Rodrigues, C. A. da Costa, D. Kreutz and H.-U. Heiss, Towards cloud-based asynchronous elasticity for iterative HPC applications, *Journal of Physics: Conference Series* **649** (oct 2015) p. 012006.

[33] V. F. Rodrigues, R. da Rosa Righi, C. A. da Costa, D. Singh, V. M. Munoz and V. Chang, Towards combining reactive and proactive cloud elasticity on running hpc applications, in *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS* INSTICC, (SciTePress, 2018) 261–268.

[34] R. da Rosa Righi, V. F. Rodrigues, G. Rostirolla, C. A. da Costa, E. Roloff and P. O. A. Navaux, A lightweight plug-and-play elasticity service for self-organizing

resource provisioning on parallel applications, *Future Generation Computer Systems* **78** (2018) 176 – 190.

[35] V. Shankar, K. Krauth, Q. Pu, E. Jonas, S. Venkataraman, I. Stoica, B. Recht and J. Ragan-Kelley, numpywren: serverless linear algebra, *CoRR* **abs/1810.09679** (2018).

[36] A. Raveendran, T. Bicer and G. Agrawal, A framework for elastic execution of existing mpi programs, in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum* May 2011 940–947.

[37] T. Gautier, J. L. Roch and G. Villard, Regular versus irregular problems and algorithms, in *Parallel Algorithms for Irregularly Structured Problems* eds. A. Ferreira and J. Rolim (Springer Berlin Heidelberg, Berlin, Heidelberg, 1995) 1–25.

[38] Y. Sun and C.-L. Wang, Solving irregularly structured problems based on distributed object model, *Parallel Computing* **29** (November 2003) 1539–1562.

[39] B. L. Massingill, T. G. Mattson and B. A. Sanders, Reengineering for parallelism: an entry point into plpp for legacy applications, *Concurrency and Computation: Practice and Experience* **19**(4) (2007) 503–529.

[40] K. Keutzer, B. L. Massingill, T. G. Mattson and B. A. Sanders, A design pattern language for engineering (parallel) software: merging the plpp and opl projects, in *Proceedings of the 2010 Workshop on Parallel Programming Patterns* ACM2010

[41] M. Parashar, M. AbdelBaky, I. Rodero and A. Devarakonda, Cloud paradigms and practices for computational and data-enabled science and engineering, *Computing in Science Engineering* **15** (July 2013) 10–18.

[42] S. Kehrer and W. Blochinger, Migrating parallel applications to the cloud: assessing cloud readiness based on parallel design decisions, *SICS Software-Intensive Cyber-Physical Systems* **34** (Jun 2019) 73–84.

[43] A. Gupta, O. Sarood, L. V. Kale and D. Milojicic, Improving hpc application performance in cloud through dynamic load balancing, in *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing* May 2013 402–409.

[44] M. Kuperberg, N. Herbst, J. v. Kistowski and R. Reussner, Defining and quantifying elasticity of resources in cloud computing and scalable platforms, Tech. Rep. 16, Karlsruher Institut für Technologie (KIT) (2011).

[45] N. R. Herbst, S. Kounev, A. Weber and H. Groenda, Bungee: An elasticity benchmark for self-adaptive iaas cloud environments, in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* May 2015 46–56.

[46] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin and S. Kounev, Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field, *IEEE Transactions on Parallel and Distributed Systems* **30** (April 2019) 800–813.