Contextual Contracts for Component-Oriented Resource Abstraction in a Cloud of HPC Services

Wagner Guimarães Al-Alam[1], **Francisco Heron de Carvalho Junior**[1]

[1]Pós-Graduação em Ciência da Computação (MDCC)
Universidade Federal do Ceará (UFC)
Campus Universitário do Pici, Bloco 912 – Fortaleza – CE – Brazil

{wgalalam,heron}@lia.ufc.br

Abstract. *The efforts to make cloud computing suitable for the requirements of HPC applications have motivated us to design HPC Shelf, a cloud computing platform of services for building and deploying parallel computing systems for large-scale parallel processing. We introduce Alite, the system of contextual contracts of HPC Shelf, aimed at selecting component implementations according to requirements of applications, features of targeting parallel computing platforms (e.g. clusters), QoS (Quality-of-Service) properties and cost restrictions. It is evaluated through a small-scale case study employing a component-based framework for matrix-multiplication based on the BLAS library.*

1. Introduction

Cloud computing provides a new execution model based on the pay-as-you-go paradigm. It allows seamless access to virtualized hardware and software [Antonopoulos and Gillam 2011, Mell and Grance 2011], enabling a number of emerging applications. Among them, the relevance of applications with High-Performance Computing (HPC) is increasing, mainly due to the massively parallel processing requirements of new applications of deep learning and Big Data technologies [Vecchiola et al. 2009, Parashar et al. 2013, Ben Nun and Hoefler 2019]. However, despite the research initiatives addressing the problem of leveraging HPC services through clouds, surveyed by Netto et al. [Netto et al. 2018], many developers are still reluctant to move their applications to the clouds. They prefer the full control of available resources offered by on-premise clusters, since they believe that virtualization of processors and interconnections may degrade performance. Furthermore, some computational scientists and engineers are concerned with the loss of control over the place where their applications will run due to legal restrictions imposed by funding agencies, corporations, and governments.

HPC Shelf is a proposal of a cloud services platform for deploying component-based *parallel computing systems* aimed at large-scale parallel processing [de Carvalho Junior et al. 2019]. Through HPC Shelf, *application providers* develop problem solving environments, so-called *applications*, for attending a community of end users, so-called *specialists*. The computational requirements of the problems addressed by applications justify the use of large-scale parallel computing systems, i.e. comprising a team of two or more clusters or MPPs. HPC Shelf is component-oriented [Wang and Qian 2005], based on the Hash Component Model [de Carvalho Junior and Rezende 2013], which makes it possible to expose resources employed by parallel computing systems, including both hardware and software, in the form of components, addressing both functional and non-functional concerns.

Parallel computing systems refer to components indirectly, through *contextual contracts*, delegating the responsibility of selecting compliant components to a resolution system. Contextual contracts define a set of requirements that selected components must satisfy, including *application requirements,* concerning the functionality required by the application, and *platform requirements*, specifying characteristics of the parallel computing platforms where the component will be instantiated for execution. Also, QoS (Quality-of-Service) requirements and cost restrictions may be imposed in contracts.

This paper introduces Alite, a system of contextual contracts for HPC Shelf. It is a framework for the selection and classification of components that comply to contextual contracts. This paper reports an evaluation of Alite through a small-scale study, using a component-based framework for matrix-multiplication based on the BLAS (Basic Linear Algebra Subprograms) library [Dongarra 2002].

The rest of the paper is structured as follows. Section 2 introduces the main foundations and concepts behind HPC Shelf. Section 3 presents Alite. Then, Section 4 evaluates the performance of its resolution strategy. Section 5 discuss other research works related to Alite. Finally, Section 6 summarizes our conclusions and outlines further work.

2. HPC Shelf

HPC Shelf is a cloud computing platform aimed at offering HPC services. Applications use this services to build solutions for problems described by their users through high-level interfaces, in the form of component-oriented *parallel computing systems* based on Hash [de Carvalho Junior and Rezende 2013], a component model of parallel components. Such problems require large-scale parallel processing, engaging of one or more parallel computing platforms, such as clusters and MPPs.

The parallel computing systems of HPC Shelf comprises a workflow component, an application component, and a set of *solution components* of the following component kinds: ***virtual platforms***, representing distributed-memory parallel computing platforms; ***data sources***, from which data that interest to applications may be retrieved; ***computations***, implementing parallel algorithms for exploiting the features of a class of virtual platforms; ***connectors***, which couple computations and data sources placed in distinct virtual platforms; and ***bindings*** of two kinds: ***service bindings***, which connect pairs of a *user* and a *provider port* belonging to components of any kind; ***action bindings***, which bind *action ports* of a set of computations, connectors and the workflow component. Parallel computing systems refer to their components through *contextual contracts*, specifying a set of requirements to guide the selection of proper implementations. Alite is the system of contextual contracts herein proposed. It is presented in Section 3.

Through a service binding, a user component consumes a service offered by a provider component. In turn, action bindings are used for orchestration of computations and connectors by the workflow component. For that, the ports of an action binding carry a common set of *action names*. Components activate actions in their action ports by referring to action names. An activation for an action n in a given port p remain blocked until a pending activation of n exists in each port connected to p through an action binding.

Components have a life-cycle action port connected to workflow with the following action names: `resolve`, for selecting a component implementation based on a

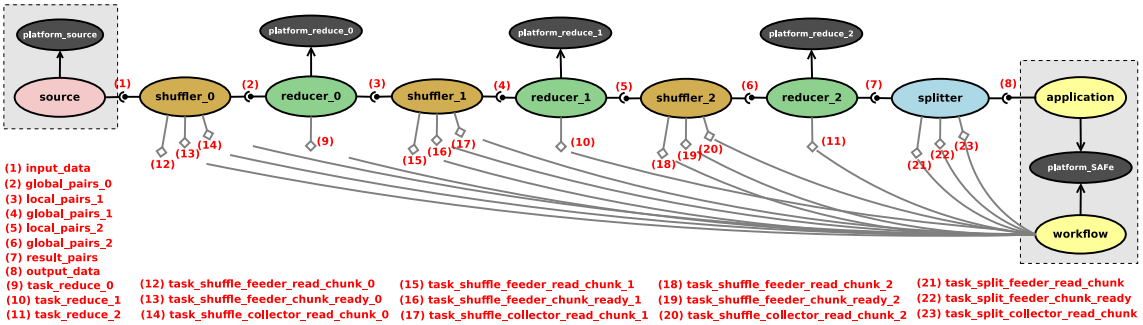

Figure 1. A MapReduce Parallel Computing System

contextual contract; `deploy`, for deploying a selected component in a virtual platform; `instantiate`, for creating an instance of the component; `run`, for executing a component instance; and `release`, for freeing the resources used by a component instance.

A connector may play two roles in a parallel computing system. They may *orchestrate* computations and connectors, through action binding synchronization, or they may give the support for choreographs among computations, connectors and data sources through service bindings. For that, they comprise a set of *facets*, each one placed in a virtual platform where a component it couples is placed, allowing direct communication.

Computation components are associated with a virtual platform where they will execute, forming a *system component*. The notion of system component is central in HPC Shelf, where a computation component may be developed under strong assumptions about the characteristics of the virtual platform where it will be instantiated.

The application component intermediates the communication between the solution components and the application itself, through service ports aimed at providing inputs, receiving outputs, managing intermediary data, monitoring components, etc.

Figure 1 depicts the architecture of a parallel computing system of a component framework we have developed for MapReduce computations [Rezende and de Carvalho Junior 2018], representing a three-stage MapReduce computation applied for enumerating triangles in a graph. The graph is obtained from a data source component (**source**) and the output is sent to application. The reducer components are computations, whereas the shuffler ones are connectors. Each reducer is placed on a distinct virtual platform. The figure emphasizes action and service bindings.

The architecture of HPC Shelf is based on the following three elements: Frontend, Core and Backend. The Frontend is SAFe (*Shelf Application Framework*) [de Carvalho Junior et al. 2019], a framework for building and running parallel computing systems, using an API (currently, only in C#) or SAFeSWL (SAFe Scientific Workflow Language). The Core manages the catalog where developers and maintainers register components and their life-cycle. Also, it implements Alite, the system of contextual contracts introduced in Section 3. Applications access the Core services for resolving contextual contracts and control the life-cycle of the selected components. Once instantiated, components are directly orchestrated by applications. The Backend is a service that each *maintainer* offer to the Core for instantiating virtual platforms. Once instantiated, virtual platforms may communicate directly with the Core for instantiating components.

⟨*contextual_signature*⟩ ::= '**[**' ⟨*constraint_list*⟩ '**]**' | ε
⟨*constraint_list*⟩ ::= ⟨*constraint*⟩ '**,**' ⟨*constraint_list*⟩ | ⟨*constraint*⟩
⟨*constraint*⟩ ::= ⟨*contextual_parameter*⟩ | ⟨*parameter_binding*⟩
⟨*parameter_binding*⟩ ::= ⟨*parameter_id*⟩ '**==**' ⟨*parameter_id*⟩
⟨*parameter_id*⟩ ::= PARAMETER_ID ('**-**' PARAMETER_ID)*
⟨*contextual_parameter*⟩ ::= PARAMETER_ID ⟨*subtyping_direction*⟩ COMPONENT_ID
⟨*subtyping_direction*⟩ ::= '$:>$' | '$<:$' | '$=$'
⟨*contextual_contract*⟩ ::= '**[**' ⟨*argument_list*⟩ '**]**'
⟨*argument_list*⟩ ::= ⟨*argument*⟩ ('**,**' ⟨*argument*⟩)*
⟨*argument*⟩ ::= ⟨*parameter_id*⟩ '**=**' COMPONENT_ID| ⟨*parameter_id*⟩ '**=**' PARAMETER_ID

Figure 2. An abstract syntax for contextual signatures and context contracts.

3. Alite: A Contextual Contract System for Resource Abstraction

Alite is based on HTS (Hash Type System) [de Carvalho Junior et al. 2016], the component type of system HPE (Hash Programming Environment) , a component-based platform for parallel programming for clusters [de Carvalho Junior and Rezende 2013]. From HTS, Alite inherits the concepts of *abstract components*, *contextual signatures* and *contextual contracts*. An abstract component represents a set of components that implement the same software concern under different assumptions about application requirements and execution environments. Such assumptions are represented by a contextual signature, defined by a set of *context parameters*. A *contextual contract* is a partial assignment of context arguments to context parameters of an abstract component. A context parameter is defined by a *name* and a *variable*, for referring to it, a *bound*, represented by a contextual contract, and a *direction*. The bound of a context parameter defines a restriction on the set of context arguments that may be assigned to it in a contextual contract. In turn, the direction may be either *covariant* ($<:$) or *contravariant* ($:>$). It is covariant if the parameter denotes an assumption that the component makes about the *contextual environment*, where the contextual environment is defined as the application to which it serves and the virtual platform where it executes. In turn, it is contravariant when it denotes an assumption that the contextual environment makes about the component.

The specification of an abstract component also includes a set of *units* and a set of *inner components*, since they are required by the Hash component model. Each inner component is typed by a contextual contract, possibly making reference to context parameters of the host component through their variables. However, for the purposes of this paper, we concentrate only on contextual signatures, ignoring units and inner components. Figure 2 presents an abstract syntax for contextual signatures, where we have attempted to use self-explained names for grammar variable, to make it easier to understand.

Alite extends the *instantiation types* of HTS to define contextual contracts, adding classes of context parameters and component kinds for representing numerical domains.

3.1. Classes of Context Parameters

Components are developed to attend application requirements in regard to the concern they address, by exploiting the features of a class of virtual platforms for optimizing performance and satisfying QoS (Quality-of-Service) requirements imposed by the application, subject to cost restrictions. For that, each context parameter is classified as an

application, *platform*, *QoS* or *cost* parameter. Such a characterization of context parameters will be useful for the resolution strategy introduced in Section 3.3.

Applications may impose statically, through contextual contracts of parallel computing systems, QoS requirements and cost restrictions that must be satisfied by the selected components. In turn, the components reached by the resolution procedure may calculate QoS and cost arguments dynamically, according to the arguments of the contextual contract under resolution, using performance and cost models.

3.2. Quantifier Components: Numerical Valuations for Context Parameters

For supporting numeric valuations in context parameters, Alite introduces a new kind of *quantifier components*, with four predefined domains, covering integer and real numbers: INT $\downarrow$ and REAL $\downarrow$, for direct subtype relation, i. e. $n <: m \Leftrightarrow n \leq m$, where n and m are quantifiers, and INT $\uparrow$ and REAL $\uparrow$, for inverse subtype relation, i. e. $n <: m \Leftrightarrow m \leq n$. Thus, let N and N' be two quantifiers of type INT $\downarrow$ (REAL $\downarrow$). N' may only be used when a quantifier N is required ($N' <: N$) if $N' \leq N$ (direct relation). In an analogous way, if they are quantifiers of type INT $\uparrow$ (REAL $\uparrow$), it is possible only if $N' \geq N$ (inverse relation). It is assumed that INT $\downarrow \subset$ REAL $\downarrow$ and INT $\uparrow \subset$ REAL $\uparrow$. Also, $+\infty$ and $-\infty$ are the top quantifiers of INT $\downarrow$ (REAL $\downarrow$) and INT $\uparrow$ (REAL $\uparrow$), respectively, i.e. they are supertypes of any quantifier in their respective domains. In what follos, we present some examples to provide more intuition about numerically qualified context parameters.

Firstly, let a covariant context parameter representing the efficiency of a parallel algorithm, measured in the interval $[0.0, 1.0]$. A computation component that ensures efficiency of 0.7 (70%) may be used in a context where it is required an efficiency of 0.6 (60%), since $0.7 <: 0.6$ by convariance. So, its bound is 0.0 and its domain is REAL $\uparrow$.

Now, let ***max_processing_nodes*** be a contravariant parameter representing the maximum number of processing nodes recommended for a computation component in the virtual platform where it is placed. If ***max_processing_nodes*** $= N$, it could be selected in a context where it is required a computation with ***max_processing_nodes*** $= M$, where $M \leq N$, so that ***max_processing_nodes*** of the selected component could not be violated. Thus, the contextual bound is $+\infty$ and the domain is INT $\downarrow$. Analogously, for the minimum number of nodes, the bound and domain would be 1 and INT $\uparrow$, respectively.

In an analogous way, a virtual platform whose processing nodes have N cores must be selected in a context where it is required a virtual platform with less than N cores, so that it will have enough cores to attend the contractual requirements. Thus, for a context parameter that represents the number of cores offered by the processing nodes of a virtual platform, the limit would be 1 and the domain would be INT $\uparrow$ too.

In a virtual platform component, a covariant context parameter for the *interconnection latency* between the nodes of the virtual platform has also INT as its domain and $+\infty$ as its bound, so that a virtual platform with a lower latency (better) than the required by the contract may be selected. It is worth note that if cost restrictions are applied through context parameters, they cannot be violated despite the use of a more efficient platform.

Finally, let a parameter for the *computing capability* of the GPUs offered by the nodes of a virtual platform, restricting the CUDA version supported by accelerated computation components. Its domain is INT $\uparrow$, since a virtual platform with GPUs of com-

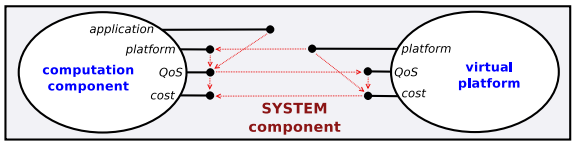

Figure 3. Dependencies between Classes of Context Arguments

puting capability 5 (higher value) could be used where computing capability 4 is required (lower value). A computation component that is selected to execute on such a virtual platform is implemented under the assumption of a computing capability less or equal than 4, so that it is safe to select a virtual platform with a computing capability of 5.

3.3. Resolution Strategy

For a given a contextual contract $\mathcal{A}$, Alite applies a resolution procedure comprising two main steps: *selection* and *classification*. In selection, a list of compliant components is generated, in such a way that their contextual contracts are subtypes of the inquired contextual contract. For computation components, selection comprises the following steps:

1. Using the resolution algorithm of HTS [de Carvalho Junior et al. 2016] restricted to application and platform context parameters, Alite finds the computation components that comply to $\mathcal{A}$. In what follows, they are $\mathcal{A}_1, \mathcal{A}_2, \ldots,$ and $\mathcal{A}_m$.
2. Each selected computation component with contract $\mathcal{A}_i$ is associated to a platform component with contract $\mathcal{P}_i$, for $i \in \{1, \ldots, m\}$, receiving arguments (contextual constraints) from both the application and the component.
3. For each $\mathcal{P}_i$, for $i \in \{1, \ldots, m\}$, use the resolution algorithm of HTS to calculate $\left\{\mathcal{P}_i^{j\, j\in\{1,\ldots,n\}}\right\}$. They are the sets contracts of compatible platforms for each $\mathcal{A}_i$.
4. Consider a set of pairs $\langle c, p\rangle$ representing *candidate system components*, where $c = \mathcal{A}_i$ and $p \in \left\{\mathcal{P}_i^{j\ j\in\{1,\ldots,n\}}\right\}$, for some $i \in \{1, \ldots, m\}$. Each pair attends the constraints imposed by the application, the virtual platform required by the computation, and the virtual platform assigned to it. Now, the pairs that satisfy the restrictions imposed by QoS and cost context arguments must be filtered.
5. Let $\langle c, p\rangle$ be a candidate system component. The context arguments of p are applied to the platform context parameters of c so that it is possible to take arguments for all the QoS parameters of c. Then, the QoS arguments of c determine the QoS arguments of p. In turn, the QoS arguments of p determine the cost arguments of p. Finally, the QoS and cost arguments of p determines the cost arguments of c. Figure 3 illustrate the dependency among context parameter classes of c and p.
6. Eliminate all pairs $\langle c, p\rangle$ such that some QoS and cost arguments do not satisfy the restrictions imposed in $\mathcal{A}$, using the subtyping relation.
7. The remaining $\langle c, p\rangle$ pairs form the set of compliant system components with respect to $\mathcal{A}$, which forms the input for the classification process.

In classification, the selected system components are ranked according to a resource allocation policy imposed by HPC Shelf. For example, it may prefer energy-efficient components, but the application may try to minimize execution costs subject to an execution time threshold. A classification criteria may vary over time and depends on the nature of the parallel computing system, involving QoS and cost context parameters. Thus, HPC Shelf must have its own global resource allocation policies, which may

matrix_type	*domain*	*float_point_precision*	*subroutine name*
GENERALMATRIX	REAL	SINGLE	**SGEMM**
	REAL	DOUBLE	**DGEMM**
	COMPLEX	SINGLE	**CGEMM**
	COMPLEX	DOUBLE	**ZGEMM**
SYMMETRICMATRIX	REAL	SINGLE	**SSYMM**
	REAL	DOUBLE	**DSYMM**
	COMPLEX	SINGLE	**CSYMM**
	COMPLEX	DOUBLE	**ZSYMM**
TRIANGULARMATRIX	REAL	SINGLE	**STRMM**
	COMPLEX	SINGLE	**CTRMM**
	COMPLEX	DOUBLE	**ZTRMM**
HERMITIANMATRIX	COMPLEX	SINGLE	**CHEMM**
	Complex	DOUBLE	**ZHEMM**

Table 1. Matrix-matrix multiplication subroutines of BLAS

.

evolve over time, but it must allow applications to influence the ranking of system components. We propose a classification framework supporting different resource allocation policies, which will be exemplified, in Section 4, with a particular ranking method.

Contextual Parameters for Ranking A new class of context parameter is now introduced, so-called *ranking context parameters*. They are predefined in HPC Shelf, and calculated dynamically according to a ranking function applied over QoS and cost parameters. Like any context parameter, their values could be quantifiers or qualifiers. In the last case, the values may define ranking levels. An application provider, possibly based on requirements imposed by specialist users, may control the ranking method by choosing one or more ranking context parameters to provide arguments. For that reason, HPC Shelf must expose clearly to providers the meaning of each ranking context parameter.

4. Case Studies

The efficiency (resolution time) and effectiveness (quality of classification) of the Alite's resolution strategy have been evaluated through a small-scale case study using a component framework based on the BLAS subset of matrix multiplication subroutines. Also, we have specified a component tuned to a specific virtual platform contract and registered platforms for three maintainers, CENAPAD-UFC, CENAPAD-RJ, and a local cluster, with various combinations of platform parameters, according to the presence of processing accelerators (MIC or GPU), number of processing nodes, processor types, and memory amount, leading to 16 different profiles.

The resolution time is measured by forcing Alite to compute over all possible system components. Also, a set of multi-criteria decision-making (MCDM) strategies available in the MCDM library of R language have been selected to classify the list of candidate system components: Weighted Product Model (WPM), Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) and Multi-criteria Optimization and Compromise Solution (VIKOR) [Tzeng and Huang 2011]. They are compared with manual classification performed by three HPC specialists. We have chosen MCDM methods based on rank calculation due to the higher overhead of strategies based on pairwise comparisons, such as AHP and PROMETHEE. However, Alite does not impose a specific method. So, other alternatives may be used in a particular implementation of HPC Shelf.

$$\begin{bmatrix} \textbf{\textit{name}} :> \text{BLAS3\_MM}, \\ \textbf{\textit{k\_size}} <: \text{INT} \downarrow, \textbf{\textit{n\_size}} <: \text{INT} \downarrow, \textbf{\textit{m\_size}} <: \text{INT} \downarrow, \\ \textbf{\textit{matrix\_type}} <: \text{MATRIXTYPE}, \textbf{\textit{domain}} <: \text{REALORCOMPLEX}, \textbf{\textit{precision}} <: \text{SINGLEORDOUBLE}, \\ \textbf{\textit{platform}} <: \text{CLUSTER}, \\ \textbf{\textit{flop\_count}} <: \text{INT} \downarrow, \textbf{\textit{estimated\_time}} <: 0{:}\text{INT} \uparrow, \textbf{\textit{power}} <: 0{:}\text{INT} \uparrow, \textbf{\textit{cost}} <: 0{:}\text{INT} \uparrow \end{bmatrix}$$

Figure 4. BLAS3_MM Component Signature

$$\text{BLAS3\_MM} \begin{bmatrix} \textbf{\textit{k\_size}} = 100, \textbf{\textit{n\_size}} = 100, \textbf{\textit{m\_size}} = 100, \\ \textbf{\textit{matrix\_pattern}} = \text{GENERICMATRIX}, \textbf{\textit{domain}} = \text{REAL}, \textbf{\textit{precision}} = \text{DOUBLE}, \\ \textbf{\textit{platform}} = \text{CLUSTER}, \textbf{\textit{node\_accelerator}} = \text{NO-ACELERATOR} \end{bmatrix}$$

Figure 5. Contextual Contract of BLAS3_MM

4.1. A Framework for Matrix Multiplication

Matrix multiplication is a common operation in linear algebra computations. In this evaluation, we employ the matrix-matrix multiplication subroutines of **Sca**lable **LAPACK** (ScaLAPACK) [Blackford et al. 1997], a well-known parallel implementation of BLAS.

Table 1 lists the subroutines which correspond to component implementations of BLAS3_MM, the abstract component whose contextual signature is presented in Figure 4. Table 1 shows how three context parameters of BLAS3_MM are combined to select a component implementation (subroutine) based on the *matrix type* (general, symmetric, triangular or hermitian), *domain type* (real or complex) and *float point precision* (single or double), respectively. Indeed, Figure 5 exemplifies a contextual contract for selecting an implementation of BLAS3_MM to compute over $100 \times 100 \times 100$ double-precision float-point matrices of real numbers in a cluster without acceleration in nodes. **DGEMM** could be a selection, according to Table 1. Also, suppose that CENAPAD-UFC-MICRO is the virtual platform contract selected to host **DGEMM**. Figure 6 shows the contract that results by combining the required contract, the contract of **DGEMM**, and the contract of CENAPAD-UFC-MICRO, the selected virtual platform, where the values of QoS and cost parameters are calculated through a performance model, as explained below.

DGEMM calculates $C = \alpha A * B + \beta C$, where α and β are scalar, $A_{m \times k}$ and $B_{k \times n}$ are input matrices and $C_{m \times n}$ is a matrix of input/output. For our purposes, it is implemented through a block-based approach [Grama et al. 2003]. To calculate the QoS context arguments of **DGEMM**, we present two equations for communication overhead

$$\begin{bmatrix} \textbf{\textit{k\_size}} = 100, \textbf{\textit{n\_size}} = 100, \textbf{\textit{m\_size}} = 100, \\ \textbf{\textit{matrix\_type}} = \text{GENERICMATRIX}, \textbf{\textit{domain}} = \text{REAL}, \textbf{\textit{precision}} = \text{DOUBLE}, \\ \textbf{\textit{platform}} = \text{CENAPAD-UFC-MICRO}, \textbf{\textit{totalGFlops}} = 1000000.0, \\ \textbf{\textit{network\_word\_rate}} = 1.0\text{E-}10, \textbf{\textit{tcomm}} = 9.587386437626905\text{E-}6, \\ \textbf{\textit{parallel\_time}} = 0.040015829583691064, \\ \textbf{\textit{power}} = 15.206015241802604, \textbf{\textit{cost}} = 2.184544168312736 \end{bmatrix}$$

Figure 6. A Contextual Contract of BLAS3_MM

Load	A	B	C
1	$0.12s$ (0)	$3.80s$ (35)	$3.75s$ (35)
2	$0.12s$ (0)	$22.03s$ (203)	$22.19s$ (201)
3	$0.12s$ (0)	$216.43s$ (2007)	$217.45s$ (2005)

Table 2. Execution time and Number of Selected Candidate Systems

and parallel execution time. In Equation 1, p is the number of processors, D is the size of double-precision values (in bytes), t_w is the per-word transfer time, calculated by the inverse of the rate of words per second, and t_s is the communication startup time.

$$\mathrm{T}_{comm} = 2log(p)(D * t_w + t_s) + \frac{(km(\sqrt{p}-1)t_w)}{\sqrt{p}} + \frac{(kn(\sqrt{p}-1)t_w)}{\sqrt{p}} + 2t_s log(\sqrt{p}) \quad (1)$$

In Equation 2, C_i is the processing capacity of the platform.

$$\mathrm{T}_p = \frac{(\frac{m\times n\times k}{p} * 4)}{C_i} + (\frac{m\times n\times k}{p} \times 8 \times t_m) + T_{comm} \quad (2)$$

The power consumption and execution cost is calculated by dividing the execution time by the total power consumption per time and the running cost per hour, respectively.

4.2. Experimental Results

The computer used in the experiments has a Intel Core i5-2500 processor with 3.30GHz clock, 4 cores, and 8GB of memory. The operating system is Ubuntu $14.04.4$ LTS.

In order to evaluate the resolution time, we define three BLAS3_MM contracts: A, B, and C. While A has no compatible platform, avoiding most of the computational work of the resolution algorithm, B and C restrict some attributes and no attribute, respectively. For each contract, we define a resolution load by varying platforms profiles registered in the Core to 15 profiles in load A, 100 in load B and 1000 in load C.

Table 2 shows a resolution time of 3.62 minutes for contracts B and C over 1000 platforms. Between parenthesis, it is shown the size of the yielded lists of system candidates. These results evidence that Alite may be suitable for a big set of platform profiles even running in a non-parallel non-optimized prototype implementation. So, by exploring the parallelism of the resolution algorithm, as well as optimizing algorithm and data structure implementations, ALITE could perform multiple resolutions in a reasonable time.

For evaluating the classification strategy, we have used three MCDM methods: WPM, Topsis and VIKOR. They are compared with classifications performed by three HPC specialists, identified by P_1, P_2 and P_3. The weights used with classification methods were 0.14, 0.29 and 0.57 for execution cost, power consumption and execution time as well all arrangements of its weights, generating 6 different classifications. Besides these variations of weights, we also made a classification with weight 0.33 for each parameter and another three combinations where one parameter has weight 1, summing 10 cases.

Table 3 shows the values calculated by *Alite* for execution cost, power consumption and execution time of DGEMM_CPU. It also shows the classifications performed by the three HPC specialists (P_1, P_2 and P_3) and automatic methods (TOPSis, Vikor and

System	Execution Cost	Power Consumption	Execution Time	P_1	P_2	P_3	TOPSIS	VIKOR	WPM
0	2.18	15.21	0.04	4	4	8	7	6	4
1	2.09	15.22	0.02	3	3	4	4	4	3
2	2.04	15.24	0.01	2	2	2	2	2	2
3	2.03	15.28	0.01	1	1	1	1	1	1
4	7.97	15.20	0.04	15	14	7	9	8	15
5	5.45	18.41	0.04	8	8	10	10	10	8
6	5.51	18.41	0.04	9	9	11	11	11	9
7	5.51	18.41	0.04	10	10	12	12	12	10
8	5.59	16.80	0.08	11	11	13	13	13	11
9	5.59	16.80	0.08	12	12	14	14	14	12
10	5.65	16.80	0.08	13	13	15	15	15	13
11	5.45	16.81	0.04	7	7	9	8	9	7
12	5.36	16.83	0.02	6	6	5	5	5	6
13	5.32	16.88	0.01	5	5	3	3	3	5
14	15.25	48.81	0.00	16	15	6	6	7	14

Table 3. Values of Execution Cost, Energy Consumption and Execution Time

WPM). One may notice a correlation between P_1 and P_2 classifications with the classification using WPM. In turn, the classification made by P_3 has found a more evident correlation with TOPsis and Vikor.

Most of the methods manually used by the specialists used the lexicographic ordering (P_1 and P_2), ignoring the effect of compensation between alternatives. This compensation occurs when one or more values are bigger than the difference in weight, influencing the ranking. In the case of P_3, he normalizes the values, not suffering the effects of compensation. This is more visible when comparing the criteria with equal weights, where P_3 got the the same classification than TOPsis and Vikor. When comparing only one parameter, the resultant classification is similar through all cases and methods.

By the analysis of the correlation matrix, we note that the MCDM methods replicate specialists choices, evidencing that WPM, Topsis, and Vikor may be used to classify candidate systems. Furthermore, other ranking methods could be used by the inclusion of an R language connector, providing a set of implementations of ranking methods.

5. Related Works and Contributions

The problem of selecting component implementations in component-based software systems has been studied since the 1990's. Among the proposed methods and tools addressing this problem, Alite has the matching of components representing parallel computations and parallel computing platforms as a distinguished feature.

Pande, Garcia and Pant have defined Pliability, a new metric to deal with component selection in a CBSE environment by involving a cost-benefit analysis and a set of qualitative parameters in order to apply the proposed metric, using Integer Programming for obtaining solutions [Pande et al. 2013]. In the work of Yazir et al. the PROMETHEE method of MCDA is used to dynamically allocate resources in a cloud. Using a distributed strategy, they use MCDA to evaluate the benefits of migrating a VM or not. They conclude that their strategy was promising regarding scalability, feasibility, and flexibility [Yazir et al. 2010]. The work of Vralsen, Mendes and Reed has introduced the terms application signature model and performance contracts to define and monitor applications to execute on a grid computer that has a dynamic nature [Vraalsen et al. 2001]. Lee, Meredith, and Vetter have developed COMPASS (Code Originated Models of Performance via

Automatic Source Scanning). It is aimed at assisting the design of performance models for analysis of parallel programs. It uses OpenARC (Open Accelerator Research Compiler) to generate the application performance model. Also, it uses the Aspen language, which has a set of tools to deal with performance equations [Lee et al. 2015]. COMPASS is an example of tool that demonstrates the feasibility of creating precise performance models of parallel programs for calculating QoS and cost parameters of computation and platform components of HPC Shelf. Mariani et al. employ machine learning to predict the performance for HPC applications running in clouds [Mariani et al. 2018]. Also, Cunha et al. have used machine learning to help the placement of jobs in clouds [Cunha et al. 2017]. In this context, Amalarethinam and Beena have published a survey in cloud scheduling, comparing many job schedule strategies [Amalarethinam and Beena 2014].

6. Conclusions and Further Work

In this paper, we introduce Alite, the contextual contract system of HPC Shelf. It defines a resolution strategy comprising selection and classification of components representing resources of parallel computing systems, taking into account application requirements, architectural features of parallel computing platforms, QoS requirements and cost restrictions. We evaluate Alite using a case study with matrix-multiplication components based on BLAS. The evaluation evidences that it is *efficient* for a reasonable set of contracts regarding resolution time. With respect to classification, the MCDM methods have been *effective*, leading to results that do not present significant discrepancies compared to a classification made by three human HPC specialists.

This work is an innovative contribution to context-sensitive allocation of system resources in cloud-based applications. Contextual contracts are able to represent the context as environmental constraints that guide allocation. In particular, the approach introduced here applies, in an unprecedented way, notions of component orientation and type safety to define a specific notion of contextual abstraction about system components (software+hardware) designed to meet specific requirements of HPC systems.

Further works will evaluate Alite for real scenarios, using a large-scale case study. For instance, we are working on component frameworks for virtual platforms of existing IaaS infrastructures, such as Amazon EC2, Google Compute Platform, and Microsoft Azure, which will be used with application frameworks we have developed, such as the MapReduce framework and Gust [Rezende and de Carvalho Junior 2018]. We are also interested in evaluating other classification strategies, particularly that ones based on execution histories of components, maybe using machine learning techniques.

References

Amalarethinam, D. G. and Beena, T. L. A. (2014). Cloud scheduling-a survey. *International Journal of Computer Applications*, 97(13).

Antonopoulos, N. and Gillam, L. (2011). *Cloud Computing: Principles, Systems and Applications*. Computer Commmunications and Networks. Springer.

Ben Nun, T. and Hoefler, T. (2019). Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis. *ACM Computung Surveys*, 52(4):65:1–65:43.

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. (1997). *ScaLAPACK User's Guide*. Society for Industrial and Applied Mathematics (SIAM).

Cunha, R. L., Rodrigues, E. R., Tizzei, L. P., and Netto, M. A. (2017). Job placement advisor based on turnaround predictions for hpc hybrid clouds. *Future Generation Computer Systems*, 67:35 – 46.

de Carvalho Junior, F. H. and Rezende, C. A. (2013). A Case Study on Expressiveness and Performance of Component-Oriented Parallel Programming. *J. of Parallel and Distributed Computing*, 73(5):557–569.

de Carvalho Junior, F. H., Rezende, C. A., Silva, J. C., Al Alam, W. G., and de Alencar, J. M. U. (2016). Contextual Abstraction in a Type System for Component-Based High Performance Computing Platforms. *Science of Computer Programming*, 132:96–128.

de Carvalho Junior, F. H., Silva, J. C., and Dantas, A. B. O. (2019). A Scientific Workflow Management System for Orchestration of Parallel Components in a Cloud of Large-Scale Parallel Processing Services. *Science of Computer Programming*, 173:95–127.

Dongarra, J. (2002). Basic Linear Algebra Subprograms Technical Forum Standard I. *International Journal of High Performance Applications and Supercomputing*, 16(2):115–199.

Grama, A., Gupta, A., Karypis, J., and Kumar, V. (2003). *Introduction to Parallel Computing*. Addison-Wesley.

Lee, S., Meredith, J. S., and Vetter, J. S. (2015). COMPASS: A Framework for Automated Performance Modeling and Prediction. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, ICS, pages 405–414.

Mariani, G., Anghel, A., Jongerius, R., and Dittmann, G. (2018). Predicting cloud performance for hpc applications before deployment. *Future Generation Computer Systems*, 87:618 – 628.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. Technical Report 800-145, Computer Security Division, National Institute of Standards and Technology, U. S. Depart. of Commerce.

Netto, M. A. S., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L. F., and Buyya, R. (2018). HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges. *ACM Computing Surveys*, 51(1):1–29.

Pande, J., Garcia, C. J., and Pant, D. (2013). Optimal component selection for component based software development using pliability metric. *SIGSOFT Softw. Eng. Notes*, 38(1):1–6.

Parashar, M., AbdelBaky, M., Rodero, I., and Devarakonda, A. (2013). Cloud Paradigms and Practices for Computational and Data-Enabled Science and Engineering. *Computing in Science Engineering*, 15(4):10–18.

Rezende, C. A. and de Carvalho Junior, F. H. (2018). MapReduce with Components for Processing Big Graphs. In *XIX Simpósio de Sistemas Computacionais de Alto Desempenho (WSCAD'2018)*.

Tzeng, G.-H. and Huang, J.-J. (2011). *Multiple attribute decision making: methods and applications*. Chapman and Hall/CRC.

Vecchiola, C., Pandey, S., and Buyya, R. (2009). High-Performance Cloud Computing: A View of Scientific Applications. In *10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN'09)*, pages 4–16. IEEE.

Vraalsen, F., Aydt, R. A., Mendes, C. L., and Reed, D. A. (2001). Performance Contracts: Predicting and Monitoring Grid Application Behavior. In Lee, C. A., editor, *Grid Computing — GRID 2001*, pages 154–165, Berlin, Heidelberg. Springer Berlin Heidelberg.

Wang, A. J. A. and Qian, K. (2005). *Component-Oriented Programming*. Wiley-Interscience.

Yazir, Y. O., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., and Coady, Y. (2010). Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 91–98.