Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 50-1 (2017) 6110–6115

Modelling and Control of Big Data Frameworks ⋆

Alberto Leva * **Alessandro Vittorio Papadopoulos** **

* *DEIB, Politecnico di Milano, Italy (e-mail: alberto.leva@polimi.it)*
** *IDT, Mälardalen University, Västerås, Sweden (e-mail: alessandro.papadopoulos@mdh.se)*

Abstract We present a model library conceived to design and assess critical components of big data frameworks, with a control-centric approach. The library adopts the object-oriented paradigm, using the Modelica language. Continuous-time and algorithmic models can be mixed, allowing to represent control code with high fidelity, and to reduce the simulation effort to the minimum required. We discuss the used modelling principles, describe the library, and show some design examples.

Keywords: Big data; Computing systems; Control-oriented modelling; Object-oriented modelling and simulation.

1. INTRODUCTION

The advent of cloud computing, and of new generation communication technologies opened the possibility to new applications in different fields. Especially, the promise of the "infinite" capacity of the cloud, provided the basis for Big Data Applications (BDAs), that are devoted to the elaboration of massive quantities of data, even in real-time when combined with data streaming. There is a number of different fields where BDAs are vital, ranging from cloud robotics, autonomous vehicles, to energy management in smart grids.

Independently of the specific application, a correct provisioning of computing resources is important to achieve the required efficiency. In fact, if too few computational resources are allocated to a BDA with respect to the current workload, i.e., the BDA is *under-provisioned*, the provided service results in poor performance. On the other hand, if too many computational resources are allocated to a BDA, i.e., the BDA is *over-provisioned*, the provided service results in good performance, but there is a waste of computational resources.

Identifying what is the right amount of resources at runtime, in face of varying environmental conditions is, in general, not an easy task (Lorido-Botrán et al., 2014; Papadopoulos et al., 2016). Feedback control solutions can provide a viable approach to tackle such a problem, and have been considered in some applications (Ali-Eldin et al., 2012; Lorido-Botrán et al., 2014). Indeed, several critical components of Big Data Frameworks (BDFs) are controllers in nature.

For a control-centric design and assessment of such components, models are however required. These have to represent the dynamic behaviour of the application, and the possible disturbances acting on it. Models also need to expose inputs and outputs suitable for the implementation of the devised solutions in the BDF at hand. Moreover, they must suit a variety of needs, from the optimisation of a single component, where a correct representation of the boundary conditions presented by the rest of the framework is necessary, up possibly to the design of an entire framework. Finally, as the assessment of a solution may require a huge number of simulation runs, computational efficiency is a must.

In this paper, building on previous research (Arcelli et al., 2016; Baresi et al., 2016), we propose a modelling paradigm to fulfil the needs just sketched from a control theoretical perspective. In particular, we propose a fluid approximations of BDF components, and a modular approach for their combination. We finally present and discuss a couple of application examples, to illustrate how the adopted modelling choices lead to an approach that can be very beneficial to tackle the addressed problems.

The paper is organised as follows. Section 2 presents state-of-the-art Big Data frameworks and technologies, the modelling paradigm proposed in this paper. Section 3 illustrates representative models for BDAs. Section 4 shows two application examples, referring to an existing framework, and concentrating in particular on control aspects. Finally, Section 5 draws some conclusions, and outlines future research.

2. BIG DATA APPLICATIONS

The amount of data produced by mobile phones, wearables, sensors and computers brings novel challenges in data storage and analysis. A number of different big data technologies have been developed to cope with such an increasing demand, such as Hadoop, Hbase or CouchDB. Occasionally, big data technologies are used to implement data-mining techniques, but more often the they are used for data processing in support of the data-mining

⋆ This work was partially supported by the Swedish Foundation for Strategic Research under the project "Future factories in the cloud (FiC)" with grant number GMT14-0032.

10.1016/j.ifacol.2017.08.2017

techniques and other data-science activities. MapReduce is a popular programming model and execution environment for BDAs (Dean and Ghemawat, 2008), and there have been some pioneering works on performance modeling (Ganapathi et al., 2010; Verma et al., 2011) and control (Cardosa et al., 2011; Berekmeri et al., 2014; Cerf et al., 2016). Spark is an alternative to MapReduce that generalises the types of data flows supported, overcoming the limitation of acyclic data flow models, and extending it with a data-sharing abstraction called "resilient distributed datasets," or RDDs (Zaharia et al., 2010). From a control theoretical viewpoint, however, Spark, MapReduce and similar technologies can be described in analogous ways without loss of generality.

In fact, when designing a controller for such kind of systems, one is mostly interested in models able to capture the system dynamics, that might come, for example, from queuing theory (Harchol-Balter, 2013), or fluid approximations of discrete-time queuing models (Wang et al., 1996). More specifically, fluid approximations are particularly useful to capture the average behaviour of a steady-state queuing system.

In general, there are many ways to describe a BDA to define and structure a BDF, but some elements are invariantly present:

- a Direct Acyclic Graph (DAG), where nodes represent operations on the data, while the edges represent the data transfers, irrespectively of which and how hardware/software entities carry out those operations;
- a bounded set of available resources, from which the entities just mentioned are to be taken and allotted to the operations;
- one or more managing entities, taking care of allotting resources so as to attain certain objectives—most frequently, a deadline for the application to complete.

In the proposed paradigm the BDA is a control system, where the managing entities form the controller, and the rest is the "process". An important difference with respect to control systems in other domains, however, is that the controller does not only manipulate process inputs to govern process outputs, for example allotting more CPU time to have a task progress at a desired rate, but at certain instants modifies the structure of the process, for example deciding how many processors will be allotted to process a set of data in parallel.

To account for this, we notice that BDAs evolve following a well defined and physics-induced pattern. The operations for which data is ready are started and produce output data, which will make other operations ready to start, and so on until everything completes. No matter which is the management policy, thus, the execution of the application is substantially driven by the availability of data. As such, we compose the model of a BDA by assembling the elements described below in their general form. The detailed implementation of these elements is specific of the BDF to be described (or designed).

Operations, possibly grouped into *stages*, are modelled as a set of queues (for example an input, a processing and an output queue) plus a Finite State Machine (FSM). Queues can be described in the continuous time as integrators with a lower saturation to zero. The FSM accounts for the phases of the operation. For example, a certain FSM may fire up the processing tasks only when the input queue finished receiving data, while another may start a processing task as soon as there is a certain amount of data in the input queue, and so on. Observe that the difference just shown pertains more to control than to "process physics", as the latter is just the evolution of the integrators. This shows that in the domain addressed herein, the separation between process and control is often not obvious. In general, however, the FSM dictates how the derivatives of the queue occupations are computed, and this principle allows for a straightforward model structuring.

Resource allocators act before an operation is started, to decide the maximum amount of resources to be allotted, and set the objectives for the resource controllers. These are modelled as event-driven systems.

It is important to notice that the adopted view makes the BDF, the BDA and the management entities a unity from the modelling standpoint, thereby substantiating – and in the hope of the authors, fostering – the application of the process/control co-design paradigm.

3. MODELS OF BIG DATA COMPONENTS

In this section we first present a few models, so as to exemplify and explain the concepts introduced in Section 2. The presented models refer to the Spark BDF [1], whence the specific terminology adopted, but the ideas presented herein are general. We end the section with a sketch of implementation.

3.1 Stage and task

A stage can be viewed as a compound operation (in Spark operations are grouped into stages based on data locality). A stage is composed of parallel tasks, each one run by an executor, that can be allotted a time-varying number of cores. We can view the task as the cascade of three queues: an input, a processing, and an output one. The continuous-time part of the model can be described as:

$$\begin{cases} \dot{n}_I(t) = r_{in}(t) - r_{IP}(t) \\ \dot{n}_P(t) = r_{IP}(t) - r_{PO}(t) \\ \dot{n}_O(t) = r_{PO}(t) f_r(t) - r_{out}(t) \end{cases} \tag{1}$$

where n_I, n_P, and n_O are the input, processing and output queue lengths, r_{in} is the exogenous input rate of the incoming data flow, r_{IP} and r_{PO} are the transfer rates from the input to processing and from the processing to the output queues, and r_{out} is the transfer rate of the processed requests. Apparently, all the rates, apart from r_{in} can be controlled by allocating or deallocating resources to the different processing stages. Given the addressed context we assume that a deadline can be missed owing to incorrectly allocated resources, but no data can be lost due to not fitting in a finite-length queue. Accordingly, we just care about the *occupation* of the queue, not about individual jobs in it, hence for the control-oriented purpose of this work, the particular service policy (e.g., FIFO) is irrelevant. Finally, f_r is the

[1] `http://spark.apache.org/`

data reduction factor, i.e., how smaller the output data set is with respect to the input one; notice that $0 < f_r < 1$ when the data is reduced, but it can be even equal to or larger than 1 when the task is not able to filter any data or, when the data need to be replicated in order to be filtered. The data reduction factor can be considered a parameter, set to conservative minimum or maximum values to carry out worst-case studies, or taken as a random variable for long-term simulations to capture the system's behaviour in the face of statistically characterised loads.

Letting σ being a switching signal describing the status of the task (that can be in one of the states represented in Figure 1), the rates are computed as follows:

$$\begin{aligned} r_{IP}(t) &= \begin{cases} R_r, & \text{if } \sigma = \text{R } \&\ n_I(t) \geq 0 \ \&\ c(t) > 0 \\ 0, & \text{otherwise} \end{cases} \\ r_{PO}(t) &= \begin{cases} R_p \cdot c(t), & \text{if } \sigma = \text{P } \&\ n_P(t) \geq 0 \\ 0, & \text{otherwise} \end{cases} \\ r_{out}(t) &= \begin{cases} R_w, & \text{if } \sigma = \text{W } \&\ n_O(t) \geq 0 \ \&\ c(t) > 0 \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

where c is the number of cores allotted to the stage, R_r and R_w are parameters representing the read and write processing rates, and R_p represents the processing rate per core. Practically speaking, the processing rates depend on availability of computational capacity (whence no core are present the rates are zero), the availability of elements in the queue, and the state σ of the BDA.

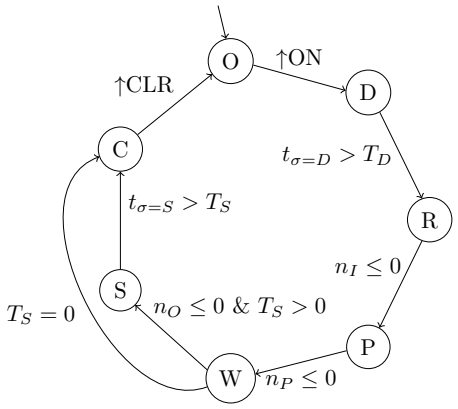

Figure 1. The FSM in the task model.

The status σ can take the values in $\Sigma = \{\text{O}, \text{D}, \text{R}, \text{P}, \text{W}, \text{S}, \text{C}\}$, i.e., **O**ff, **D**eserialise (basically, load the code from disk), **R**ead data, **P**rocess, **W**rite data, **S**erialise result (to disk, if required), and **C**leanup. The evolution of σ is governed by the FSM shown in Figure 1, where parameters T_D and T_S are the deserialising and serialising time (setting T_S to zero means that serialising data to disk is not required). The input ON is a logic signal used by a managing entity to trigger the task execution, while CLR indicates that cleanup is completed; CLR is an input as well, since the task may want to generate it autonomously, for example after a fixed time since the FSM entered state C, but the same signal may come form the exterior if more than one task needs terminating before the stage ends.

The FSM, coupled to (1)–(2), models the task as a switched linear system—notice that the way derivatives are computed allows to not introduce integrator saturation in the equations.

Based on this, a stage is readily modelled as a vector of task models, of dimension equal to the number n_E of executors allotted at the stage beginning by the managing entity. In this case the ON signal is distributed to all the tasks, and the CLR signal for all of them is obtained as the logical and operator of all the $t_{\sigma=S} < T_S$ conditions (i.e., the last task entering the C state triggers the cleanup and the end of the stage).

Finally, a vector input $\eta \in \mathbb{R}^{n_E}$ of "executor efficiencies" can be introduced. Its components lie in the $[0, 1]$ range, with a unity nominal value, and reducing them is a means to subject the system, for example, to a processing speed reduction due to a clock frequency decrease, in turn possibly caused – among a number of reasons – by thermal issues.

3.2 Stage controller

During the execution of a stage, a modulating controller can be used to ensure that some specifications are met. Specifically, we complemented our description of Spark with a stage-level controller composed of a discrete-time PI per executor, that decides how many cores to allot based on the measured progress in processing the data, and on the corresponding set point—see Section 3.3 later on for the set point generation. Since there is no inter-executor communication, we have to deal with a decentralised system, and thus we reason at the single executor level.

Denote by $\eta(t)$ the efficiency of the executor at hand, and observe that the controller has to operate only during the processing phase, i.e., when $\sigma = \text{P}$. In this case, recalling (1)–(2) and the FSM of Figure 1, the controlled system, with input c and output n_P, is

$$\dot{n}_P(t) = -R_p \eta(t) c(t). \tag{3}$$

This system is time-varying and has type 1. Hence a PI controller allows to asymptotically track a ramp set point with zero steady state error, which in the following we will show to be what is needed for this type of applications. However, we must guarantee stability accounting for the time-varying nature of (2). Coupling this system with the fixed-parameter PI

$$\begin{cases} \dot{x}_R(t) = k_I \left(n_P^\circ(t) - n_P(t) \right) \\ c(t) = x_R(t) + k_P \left(n_P^\circ(t) - n_P(t) \right) \end{cases} \tag{4}$$

where n_P° is the set point for n_P, produces the linear, time-varying closed-loop system with state $x(t) = [n_P(t)\ x_R(t)]^\top$ and state equation

$$\dot{x}(t) = A(t)x(t) + b(t)n_P^\circ(t) \tag{5}$$

where

$$A(t) = \begin{bmatrix} -R_p \eta(t) k_P & R_p \eta(t) \\ -k_I & 0 \end{bmatrix}, \quad b(t) = \begin{bmatrix} R_p \eta(t) k_P \\ k_I \end{bmatrix}. \tag{6}$$

Assuming that $\eta_{\min} \leq \eta(t) \leq 1$, where $\eta_{\min}$ is a minimum efficiency below which the executor simply has to be shut off together with its controller, it is possible to state the following theorem.

Theorem 1. A sufficient condition for guaranteeing that the autonomous system $\dot{x}(t) = A(t)x(t)$, with matrix

$A(t)$ as per (6), has the origin of the state space as a globally asymptotically stable equilibrium, is that the pair of positive parameters (k_P, k_I) satisfies the following inequality:

$$k_I \leq \frac{R_p \eta_{min}}{\eta_{min} + 1} k_P^2 \tag{7}$$

Proof. The matrix $A(t)$ can be written as $\alpha(t)A_1 + (1 - \alpha(t))A_2$ by setting

$$A_1 = \begin{bmatrix} -R_p \eta_{\min} k_P & R_1 \eta_{min} \\ -k_I & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} -R_p k_P & R_1 \\ -k_I & 0 \end{bmatrix} \tag{8}$$

where $0 \leq \alpha(t) \leq 1$. It is easy to prove that that both A_1 and A_2 are Hurwitz if $k_P > 0$ and $k_I > 0$, hence a sufficient condition for the origin to be a globally asympotically stable equilibrium is that A_1 and A_2 have a common Lyapunov function.

Theorem 3.1 in Shorten and Narendra (2002) states that A_1 and A_2 have a common (quadratic) Lyapunov function if and only if $A_1 A_2$ and $A_1 A_2^{-1}$ do not have real negative eigenvalues. The characteristic polynomials of $A_1 A_2$ and $A_1 A_2^{-1}$ are respectively

$$\begin{aligned} \pi_1(s) =& s^2 + R_p \left(k_I(\eta_{\min} + 1) - k_P^2 R_p \eta_{\min} \right) s + \\ &+ \eta_{\min} k_I^2 R_p^2, \\ \pi_2(s) =& s^2 - (1 + \eta_{\min}) s + \eta_{\min}. \end{aligned} \tag{9}$$

Polynomial $\pi_1(s)$ has no roots with negative real part if and only if

$$k_I(\eta_{min} + 1) - k_P^2 R1 \eta_{min} \leq 0, \tag{10}$$

while $\pi_2(s)$ evidently has two roots with positive real part.

Therefore, a sufficient condition for the hypothesis to hold true, is (7). □

Given this condition, the tuning of the PI is quite straightforward with a number of different methods, hence we do not delve here into further details if not for a short remark later on. We also omit considerations on the quantised nature of the control signal, see e.g. (Maggio et al., 2013, Section IV-C) for a possible way to address this problem in the case of core allocation.

3.3 Stage set point generator

Before starting a stage, the set points for the PI controllers (assumed here all equal for simplicity) need generating. The simple solution shown here is to start from a deadline T_{DL} for the stage, measure the initial occupation of each processing queue (n_{P0} to name it, as we refer to a single controller for the same reason as above) and build a trapezoid n_P° profile as

$$n_P^\circ(t) = \max \left(n_{P0} - \frac{n_{P0}}{T_{DL}(1 - \beta) - t_{P0}} t_P, 0 \right), \tag{11}$$

where t_P is the time spent in the processing phase, and $\beta \in [0, 1)$ dictates how earlier with respect to the deadline the controlled task is expected to terminate. In the first place β has to account for the expected duration of the writing phase, but the more disturbances are expected, the higher β has to be; t_{P0} and n_{P0} are measured at the beginning of the processing phase. The set point generation is illustrated graphically in Figure 2.

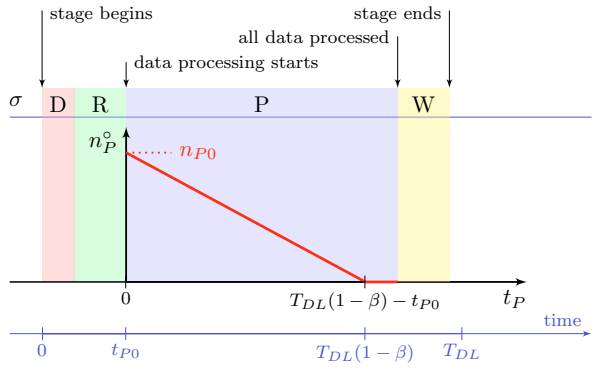

Figure 2. Set point generation as per (11).

The deadline for a stage, as well as possibly its β, is set by higher levels in the control hierarchy, not described in this paper. The only precaution to take is that the settling time of the PI loop, as computed with the tuned controller, be smaller then the expected duration of the processing phase. This is to guarantee that the controlled variable settles on the set point ramp well before the termination of that phase. The required duration estimates can come e.g. from profiling or historical data; one could also consider an adaptive controller, although we do not discuss the matter in this paper.

3.4 Implementation

We realised the presented models – and others – in the form of a Modelica library. We chose Modelica for two main reasons. First, the language allows to mix continuous-time, equation-based and event-driven, algorithm-based modelling. This permits to describe system and control in the continuous time, to exploit the efficiency of variable-step solvers, or to model the latter as code *replica*, to assess the devised control algorithms. Second, the equation-based, object-oriented paradigm behind Modelica is inherently multi-physics. To give just one example of the perspectives this opens, the models presented here could be seamlessly coupled to thermal ones, so that the available computational speed be realistically related to the heating of processors.

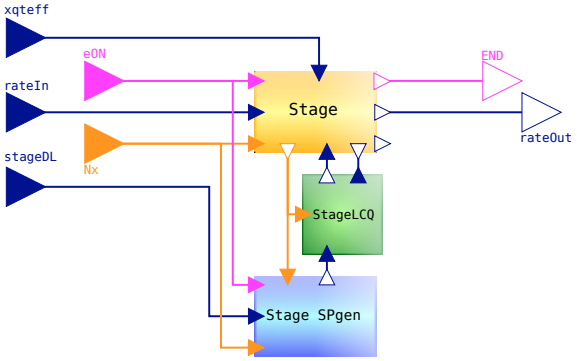

Figure 3. Stage with modulating control – Modelica diagram.

Modelica tools – some of which are free software – also allow to assemble models with a user-friendly visual interface. For example, combining the models just described,

the Modelica scheme for a stage endowed with its modulating controller takes the aspect shown in Figure 3.

4. SIMULATION EXAMPLES

4.1 Stage-level control

In this example we show a single stage endowed with its modulating control. The simple Modelica scheme is in Figure 4, where the Stage block is the icon for the internal representation in Figure 3.

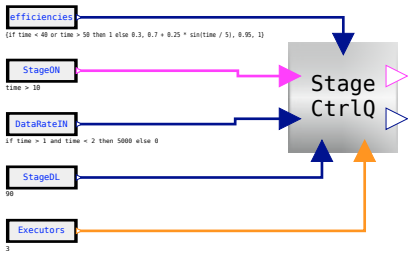

Figure 4. Simulation example 1 – Modelica diagram.

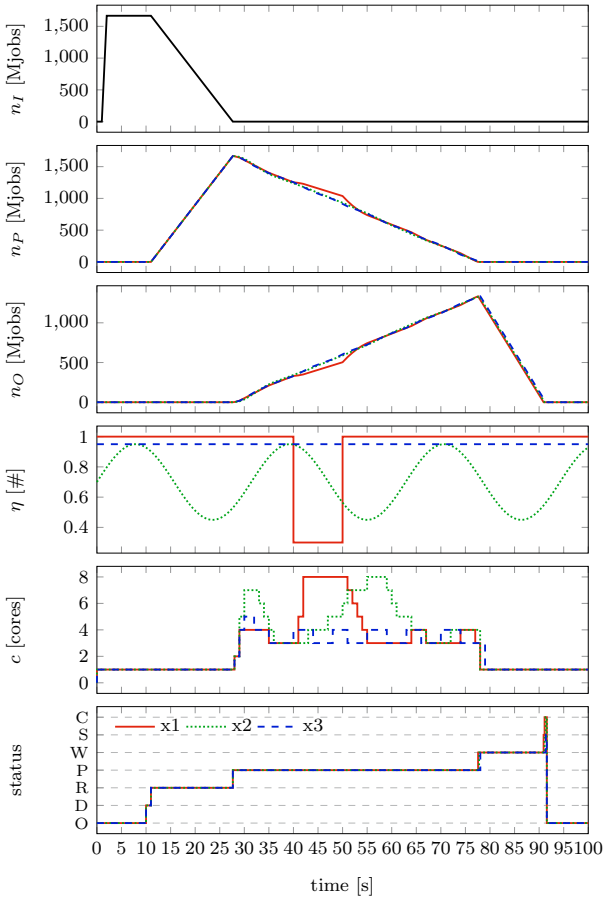

Figure 5. Simulation example 1 – results: queue lengths, executor efficiencies, allotted cores, and traversed status values.

Figure 5 shows the simulation results for a stage deadline of 90s and an activation instant at 10s, allotting the stage three parallel executors (x1 to x3). From top to bottom, the six plots report the length of the input, processing and output queues, the time-varying efficiencies of the three executors, the cores allotted to them, and the status values traversed by them till the completion of the stage.

4.2 A complete application

We now show a second example, in which a larger case is treated. The diagram of Figure 6 depicts the application. The scheme is composed of Stage blocks (already described) plus Shuffle ones, that distribute – possibly with duplications if needed – the output of upstream stages to the input of downstream ones.

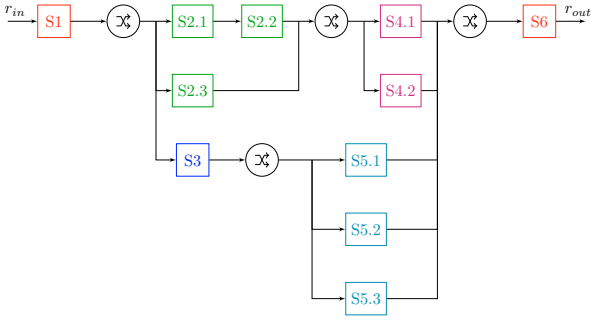

Figure 6. Simulation example 2 – scheme: the coloured blocks indicate the stages, the circles with the crossed arrows indicate the shuffle operations.

Figure 7 shows a synthesis of the results, reporting the states traversed by each stage along the evolution of the application; the colours of plots match those of the Stage blocks in Figure 6.

It is worth reporting that for 500 s of simulated time, and with a timestep of 1s for all the Stage PI controllers, only 450 ms of CPU time were used, which is about 1100× real time. This is a good argument in the favour of the adopted modelling framework.

5. CONCLUSIONS AND FUTURE WORK

We presented a modelling paradigm for big data frameworks, the first *nucleus* of a simulation library built along the said paradigm, and a couple of simulation examples to support the proposal.

The paradigm allows to describe existing frameworks by evidencing and abstracting their common elements, and is keen to produce efficient simulation models. Also, thanks to the adoption of an object-oriented modelling language, models are inherently open to multi-physic environments. This allows, in perspective, to couple models of Big Data applications to models of the physical environment (e.g., from the thermal behaviour of a CPU to potentially an entire data centre) for more comprehensive and effective evaluations of sizing and management strategies.

The paradigm also allows to introduce controllers in a straightforward manner. This paves the way to a control-based *design* – not just management – of Big Data applications and frameworks—an approach that already proven successful in other domains like operating systems.

Future work will be directed at completing the simulation library, exploiting the further possibilities just sketched, and apply the presented ideas – from both the control and the design viewpoint – to real-world applications.

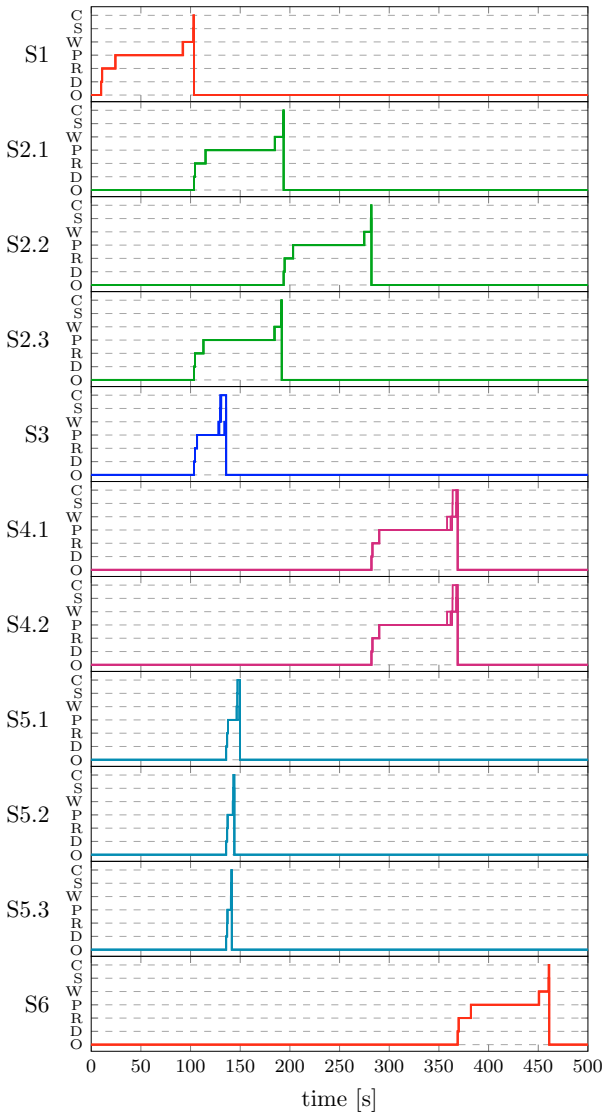

Figure 7. Simulation example 2 – results: status values (O through C) traversed by the stages from the beginning till the end of the application.

REFERENCES

Ali-Eldin, A., Tordsson, J., and Elmroth, E. (2012). An adaptive hybrid elasticity controller for cloud infrastructures. In *IEEE Network Operations and Management Symposium*, NOMS 12, 204–212. doi:10.1109/NOMS.2012.6211900.

Arcelli, D., Cortellessa, V., and Leva, A. (2016). A library of modeling components for adaptive queuing networks. In *Proc. 13th European Workshop on Performance Engineering*. Chios, Greece.

Baresi, L., Guinea, S., Leva, A., and Quattrocchi, G. (2016). A discrete-time feedback controller for containerized cloud applications. In *Proc. 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (to appear)*. Seattle, WA, USA.

Berekmeri, M., Serrano, D., Bouchenak, S., Marchand, N., and Robu, B. (2014). A control approach for performance of big data systems. *IFAC Proceedings Volumes*, 47(3), 152–157. doi:http://dx.doi.org/10.3182/20140824-6-ZA-1003.01319. 19th IFAC World Congress.

Cardosa, M., Narang, P., Chandra, A., Pucha, H., and Singh, A. (2011). STEAMEngine: Driving mapreduce provisioning in the cloud. In *2011 18th International Conference on High Performance Computing*, 1–10. doi:10.1109/HiPC.2011.6152649.

Cerf, S., Berekmeri, M., Robu, B., Marchand, N., and Bouchenak, S. (2016). Adaptive optimal control of mapreduce performance, availability and costs. In *Feedback Computing*. doi:10.1145/1235.

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1), 107–113. doi:10.1145/1327452.1327492. URL `http://doi.acm.org/10.1145/1327452.1327492`.

Ganapathi, A., Chen, Y., Fox, A., Katz, R., and Patterson, D. (2010). Statistics-driven workload modeling for the cloud. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, 87–92. doi:10.1109/ICDEW.2010.5452742.

Harchol-Balter, M. (2013). *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press.

Lorido-Botrán, T., Miguel-Alonso, J., and Lozano, J.A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 1–34. doi:10.1007/s10723-014-9314-7.

Maggio, M., Hoffmann, H., Santambrogio, M., Agarwal, A., and Leva, A. (2013). Power optimization in embedded systems via feedback control of resource allocation. *IEEE Transactions on Control Systems Technology*, 21(1), 239–246.

Papadopoulos, A.V., Ali-Eldin, A., Årzén, K.E., Tordsson, J., and Elmroth, E. (2016). PEAS: A performance evaluation framework for auto-scaling strategies in cloud applications. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 1(4), 15:1–15:31. doi:10.1145/2930659.

Shorten, R. and Narendra, K. (2002). Necessary and sufficient conditions for the existence of a common quadratic Lyapunov function for a finite number of stable second order linear time-invariant systems. *International Journal of Adaptive Control and Signal Processing*, 16, 709–728.

Verma, A., Cherkasova, L., and Campbell, R.H. (2011). *Resource Provisioning Framework for MapReduce Jobs with Performance Goals*, 165–186. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-642-25821-3_9.

Wang, W.P., Tipper, D., and Banerjee, S. (1996). A simple approximation for modeling nonstationary queues. In *Proceedings of the Fifteenth Annual Joint Conference of the IEEE Computer and Communications Societies Conference on The Conference on Computer Communications - Volume 1*, INFOCOM'96, 255–262. IEEE Computer Society, Washington, DC, USA. URL `http://dl.acm.org/citation.cfm?id=1895807.1895846`.

Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, 10–10. USENIX Association, Berkeley, CA, USA. URL `http://dl.acm.org/citation.cfm?id=1863103.1863113`.