Computational Resource Management for Data-Driven Applications with Deadline Constraints

Rafael Tolosana-Calasanz[1], Javier Diaz-Montes[2], Omer Rana[3] and Manish Parashar[2], Erotokritos Xydas[4], Charalampos Marmaras[4], Panagiotis Papadopoulos[5], Liana Cipcigan[4]
[1] Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain
[2] Rutgers Discovery Informatics Institute, Rutgers University, USA
[3] School of Computer Science & Informatics, Cardiff University, UK
[4] School of Engineering, Cardiff University, UK
[5] UK Power Networks, UK
contact author: rafaelt@unizar.es

Abstract—Recent advances in the type and variety of sensing technologies have led to an extraordinary growth in the volume of data being produced, and led to a number of streaming applications that make use of this data. Sensors typically monitor environmental or physical phenomenon at pre-defined time intervals or triggered by user defined events. Understanding how such streaming content (the raw data or events) can be processed within a time threshold remains an important research challenge. We investigate how a cloud-based computational infrastructure can autonomically respond to such streaming content, offering Quality of Service guarantees. In particular, we contextualise our approach using an Electric Vehicles (EVs) charging scenario, where such vehicles need to connect to the electrical grid to charge their batteries. There has been an emerging interest in EV aggregators (primarily intermediate brokers able to estimate aggregate charging demand for a collection of EVs) to coordinate the charging process. We consider predicting EV charging demand as a potential workload with execution time constraints. We assume that an EV aggregator manages a number of geographic areas, and a pool of computational resources of a cloud computing cluster to support scheduling of EV charging. The objective is to ensure that there is enough computational capacity to satisfy the requirements for managing EV battery charging requests within specific time constraints.

Index Terms—Elastic resource provisioning, autonomic systems, feedback control.

1 Introduction

With the technological development in sensing technologies, there has been an increase in the volume (and velocity) of data becoming available and in the number of industrial and commercial applications utilizing such data. These sensors continuously monitor environmental or physical phenomenon, such as humidity, pressure, temperature, and energy consumption, and they stream their raw data measurements to a main location through the network for computation and analysis. Depending on the complexity of the sensors or instruments involved, data rates and generation timelines can vary significantly across these different types of infrastructures. Often, applications need to analyze such data in a streaming fashion within a time threshold (deadline). Based on their latency requirements, these streaming applications can be classified in two main types, namely low latency and medium to high latency: (i) *Low latency* applications require response times in the order of milliseconds and typically handle hundreds or thousands of events per second, where each event is processing using a simple function – e.g. in financial streams, intrusion detection, fraud detection. In contrast, (ii) *medium to high latency* applications have response times in the order of seconds, minutes, or even hours, their workloads are typically coarse-grained and more complex, hence requiring more computational resources, often executed as *batch* processes. We can find examples of these in areas such as surveillance and monitoring [1], *smart*-traffic management, energy management in smart building [2], geo-spatial imaging processing [3], and data analysis of electricity meter data to support demand estimation & prediction [4]. When a provider manages a number of such applications with a shared, elastic computing infrastructure, understanding how such streaming content can be processed within some time threshold, so that the number of computational resources can be minimised remains an important challenge. More specifically, a *slack* can be identified, when the (estimated) time required for processing is less than the processing time required in a Service Level Agreement (SLA) – and may be defined as the difference between the deadline (established in the Service Level Agreement (SLA)) and the actual processing time. Data elements from different applications can be buffered by the provider for the given time slack, before starting the processing to reduce resource usage overheads (such as reducing storage space at the destination) Our focus in this paper is on medium-to-high latency applications. Specifically, we focus on battery charging of Electric Vehicles (EVs) as our driving leitmotiv scenario. Nevertheless, we believe that our findings can be generalized and applied to many

contexts that share similar requirements and characteristics.

Nowadays, there is a growing market share of EVs over conventional Internal Combustion Engine (ICE) powered vehicles, due to political, socio-economical and ecological factors. According to the UK Committee on Climate Change, the British Government should aim for 1.7 million EVs on British roads by 2025 [5]. In such a context, EVs will have to be connected to power networks in order to re-charge their batteries. In the future development of Smart Grids, some of these applications will be managed by new market players along with the utility companies and Distributed Network Operators (DNOs). One example is the EV aggregator, which will be acting as a broker between utilities and EV customers. Aggregators are anticipated to purchase electricity from the power market, and to sell it to the EV customers for re-charging their batteries. This activity involves a number of energy management activities such as planning & scheduling of charging, day-ahead demand forecast, estimating potential "triad" periods, or real-time monitoring and control of energy consumption. KiwiPower[1] is an example of an aggregator company currently operating in the UK.

According to recent studies [6], by 2030, in case the charging of EV batteries is left uncontrolled, a significant increase in the electricity demand peaks is to be expected [6]. A controlled charging process of EVs involves gathering all the charging requests from a particular area of the electricity network, and establishing a charging schedule such that system power losses are minimized and electrical and user constraints are met. The existing proposals for controlled EV battery charging can be classified into two main approaches: (i) *centralized approaches* [7], [8], [9], which accomplish all the computations required for a given region at a central node. However, due to the size of data from smart meters and subsequent processing required, these approaches are not feasible for real-time control; (ii) *distributed approaches* [10], [11], [12], that accomplish computations at different granularities (e.g. at the EV level, or at a geographic area level) in order to alleviate the workload of the coordinator node. As reported in [13], centralised approaches are not effective for real-time control at large-scales, due to the large volume of smart metering data and processing required. On the other hand, to the best of our knowledge, the existing distributed approaches have ignored *computational resource management*, i.e., the dynamic (de-) allocation of computational resources as charging demand varies to ensure that estimates can be calculated within a particular time interval. In a dynamically chaging context, one-off, statically generated estimates are likely to be inaccurate and unuseful.

In this paper, we propose an approach that combines cloud computing and queueing theory to manage the provisioning of computational resources to meet the requirements of EV aggregators in an efficient and automated manner. We achieve this by predicting demand when the density of EVs and the data they generate grow (in an unpredictable manner). Specifically, we consider the scenario where an EV aggregator is in charge of a large number of areas and can access a pool of computational resources. In general terms, charging requests for each area arrive at the EV aggregator which, in turn, has to compute charging schedules for each area periodically. The overall time threshold (deadline) for each schedule lasts for a whole electrical control period (until the next requests for an area arrive) and, typically, the processing time for undertaking scheduling is less than this overall control period. Our objective is to optimize the number of computational resources (e.g., number of Virtual Machines (VMs) within a cloud system) allocated, so that data elements are processed, *on average*, within the pre-specified time threshold, using a minimal number of VMs. For such an objective, the data elements of a *given data stream are buffered at a queue* for the maximum time (*time slack*) that, together with the processing time, does not violate the deadline on average.

Our approach uses reactive and proactive/predictive controllers to achieve dynamic and elastic management of computational resources to enable efficient and timely calculation of such a schedule. Based on feedback control and making use of system properties derived from queueing theory, the reactive controller dynamically adapts the number of VMs to meet the SLA for an aggregator. Since, in practice, information about waiting time of data elements may not always be available or may be very difficult to obtain, we decided to make use of Little's law (LL) for deriving waiting time from arrival rate and queue (buffer) size. In other words, the controller reacts by (de-)allocating resources on changes of the queue size, leading to an indirect control of the time slack. It should also be noted that although LL has been widely studied for infinite times and stationary conditions, recent studies show that it also holds under finite times and non-stationary conditions [14].

A purely reactive controller can be self-adaptive to variations in workload, but it assumes no knowledge of the workload itself. Hence, due to resource management constraints in cloud systems, e.g. provisioning overheads of VMs, provisioning actions are not instantaneous and, in consequence, a reactive controller may have limited benefit when workload fluctuates. This may lead to poor performance, i.e. oscillations over the target and *over reactions*, even predisposing it to sub-optimal use of resources. Thus, the reactive controller is improved by incorporating predictive capabilities that would help to establish a baseline for the number of computational resources required, using historical data. However, the two controllers acting independently need to coordinate their actions, in the context of provisioning actions with some time delay: the action of the predictive controller may need a time interval to take effect, but it might also be inaccurate (due to an incorrect estimate of workload). Although there are a number of existing hybrid (reactive–predictive) autonomic controllers [15], [16], to the best of our knowledge, they do not consider coordination policies among the reactive and predictive functionalities, which are mandatory for accurate performance in real practice. We propose a coordination policy to guarantee that both controllers (reactive and predictive) cooperate towards the objective. We implemented our controller on top of the CometCloud system in a federated Cloud scenario and validated it using data from the ECOTality Electric Vehicle project [17]. The key contributions of this paper include:

1) A novel approach, to the best of our knowledge, that

1. http://kiwipower.co.uk

can elastically adapt the computational resources allocated by an EV aggregator to follow the fluctuating EV charging demand.
2) Design and implementation of the proposed approach using queueing theory and autonomic techniques, realised using a combination of both predictive and reactive controllers.
3) A coordination policy that influences the subsequent resource management, ensuring that the predictive and reactive controllers are aware of each other and work together towards the same goal.
4) Experimental validation of our approach using the federated CometCloud system (able to dynamically add additional computation resources on-demand, across multiple sites), using (real) data for EV charging demands obtained from the ECOTality project.

In Section 2, an overview of Little's Law is given as well as its use in practice. The controlled EV charging process is described in Section 3. Its computational requirements are described in Section 4. The reactive and predictive controllers are described in Sections 5.2 & 5.1, respectively. In Section 6, the performance of the approach is validated experimentally using the CometCloud system Related work is compared to this proposal in Section 7. Conclusions drawn are discussed in Section 9.

2 Background: Little's Law

A queueing system comprises [14] a number of discrete objects often called items, arriving at some rate within a system. The stream of arrivals enters the system, joins one or more queues, eventually receives a service, and exits in a departure stream. In general terms, services perform operations over items, which involve an amount of time (e.g., for computing – we typically call such an action, processing, and to the amount of time it takes, processing time). Little's Law (LL) is a mathematical relationship between three variables: the average number of items in a queueing system (L) equals the average arrival rate of items (λ), multiplied by the average waiting time of an item (W). Thus, $L = \lambda * W$.

Although Little's Law was originally proposed for infinite intervals and stationary conditions for the distribution of the system variables, it has been shown to hold under a number of conditions, including finite intervals of time. This provides significant value for engineering design and operational problem solving [14]. Next, we summarize LL theorems [14].

Theorem 1. (from [14]) Little's Law Over $[0, T]$. LL.1. For a queueing system observed over $[0, T]$ that is empty at 0 and at T and has $0 < T < \infty$, $L = \lambda * W$ holds.

A number of important observations for *practical consideration* can be obtained from the previous theorem. First, LL holds for finite intervals of time and under *non-stationary* conditions. This is key in practice, as the probability distribution of arriving items may be non-stationary. In a computing system, for instance, the arrival of data elements to a computing system can be subject to sudden data bursts or spikes. Analogously, performance of computational resources can also be subject to sudden variations – e.g., unexpected performance degradation or failures. Second, LL also holds independent of the queue discipline. Furthermore, a generalisation of LL.1 is also proposed in [14] by eliminating the restriction of zero starting and ending queues in the interval $[0, T]$.

Theorem 2. (from [14]) Little's Law over $[0, T]$. LL.2. For a queueing system observed over $[0, T]$ that has $0 < T < \infty$, $L = \lambda * W$ holds.

In addition to the remarks derived from LL.1, LL.2 also ensures that LL holds when the queueing system is not empty at the beginning or at the end of the interval of time. Nevertheless, the conservation of items still needs to be guaranteed for LL to be held – i.e. there are no lost items. Using LL.2, we can estimate the average waiting times, but as it is pointed out in [18], this simple indirect estimator tends to be significantly biased when arrival rates are time-varying and service processing times are relatively long. Surrogate estimators are also proposed in [18] for such cases.

2.1 Traffic Intensity and Queueing Time

An important concept in Queueing Theory is that of *traffic intensity*, which is the ratio of the incoming and outgoing rates in a queueing system.

Definition 1. Given a multi-server queueing system, its *traffic intensity*, denoted as ϱ, is:

$$\varrho = \frac{\lambda}{c\mu_u} \tag{1}$$

where λ is the average arrival rate of items, c is the number of server instances, and μ_u is the average throughput per server. In order for a system with a finite number of servers to be stable, its traffic intensity must be $0 \leq \varrho < 1$. Nevertheless, this is not the case when considering an infinite number of servers. In that case, the system will always have enough servers, and we are then more interested in the number of busy servers and their utilization.

Our approach assumes that we always have enough resources, which is analogous to having an infinite number of resources. We make use of the traffic intensity for deriving the number of computational resources (server instances in the definition) required during the execution. Additionally, we also characterize the average time that each data element of a data stream spends on the queue. The focus here is primarily to consider the total number of active resources (and how these could be activated/de-activated over time).

Definition 2. The average time T items spent within a queueing system can be characterised by

$$T = W + S \tag{2}$$

where W is the average time a data item spends waiting for a resource in the queue, and S the average processing time for a data item.

3 The EV Charging Process

The distributed controlled charging process that we are considering in this paper is described in [12]. It consists of two activities enacted concurrently: (i) the scheduling (initialization) activity and (ii) the operational activity enacted periodically with an established control frequency – expected to be at 15 to 30 minute intervals for most SmartGrid scenarios. The overall time required for charging an EV battery is a multiple of these intervals (e.g. in residential areas, the whole charging process an EV can require over 6 hours).

3.1 The Scheduling Period

We consider the optimization problem proposed in [12], which focuses on deriving the optimal solution for scheduling EV charging requests within a geographic area. At the beginning of each scheduling activity, an EV connected to a charging point sends to the aggregator the following information: (i) the actual state of charge (SoC), (ii) the desired SoC at the end of the charging session, (iii) the connection duration, (iv) the on-board battery charger efficiency, (v) the charging power rate and (vi) the EV battery charging efficiency.

On the other hand, the aggregator also receives the loading capacity limits (i.e. maximum electrical load at a particular point) contained in the energy network as a limits matrix, and the electricity prices for each scheduling activity from the Distribution System Operator (DSO). Then, for each area, the aggregator needs to compute the following *mixed integer optimization* objective function:

$$min.\ Z = \sum_{t=T}^{T_f} \sum_{n=1}^{N} E_{v_{n_t}} * p_t \tag{3}$$

where T is the time interval associated with a particular operational activity, T_f is the final operational activity, N is the total number of EVs managed within an area, $E_{V_{n_t}}$ is the energy in kWh supplied to EV n during activity t, and p_t is the electricity price at hour t in £/kWh. This equation is subject to a number of constraints (see [12] for details), which enforce that the electricity distribution constraints are not violated and the owners' demands are satisfied. With such a scheduling, the aggregator can monitor the evolution of EVs' SoC and deal with possible changes in user preferences (for example the change in departure time or in the final SoC requirements). Finally, a schedule for an area is sent along with curtailment factors of the following operational activity to the DSO. The curtailment factors are used by the DSO in order to decide which EVs charging should be curtailed in case of an emergency.

3.2 The ECOTality dataset

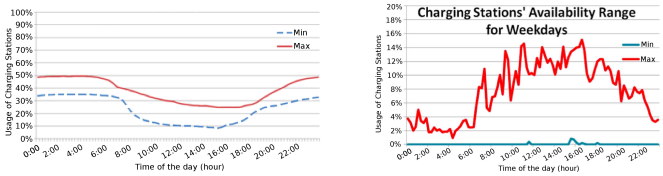
ECOtality, Inc., is an electric transportation and storage technologies company, and it is the parent company of *ECOtality North America* (formerly eTec), Innergy Power Corporation, Fuel Cell Store, and ECOtality Australia Pty Ltd. ECOtality North America manages *The EV Project* [17], which is one of the largest electric vehicle infrastructure demonstration projects, with a budget of over $230 million, equally funded by the US Department of Energy, and ECOtality and its partners. This project focuses on examining the various activities and situations involving EV drivers' behavior and charging infrastructure use [17], [19]. For all these reasons, it can be considered as one of the most realistic deployments available.

Data gathered by the EV Project is reported on a quarterly basis, and accessible online [2]. A charging event is defined in the EV Project as the period when a vehicle is connected to a charging unit, during which period some power is transferred [17], [19]. For a Smart Grid management purpose, smart meters at charging points transfer data packets continuously during the power transfer at the beginning of each scheduling period (i.e. hourly, subject to specific regulations). Data gathered from charging events at the EV Project include probability distributions for the hourly average usage of charging points, the aggregated hourly charging demand, the average energy requested per charging event, the average recharging time, etc. By the 2nd Quarter of 2013, over 2.9 million charging events had been recorded by the EV Project from project participants in the US driving vehicles enrolled in it: approximately 8,300 Nissan Leaf, Chevrolet Volts, and Smart ForTwo EVs [17], [19]. These EVs made use of an infrastructure that consisted of nearly 8,200 Residential electric vehicle supply equipment (EVSE) charging stations, over 3,750 Commercial (publicly available, workplace, and fleet) EVSE charging stations, and 87 DC Fast Chargers (DCFC). For data privacy reasons, each dataset provided *aggregate* values for an entire participating electricity network area in the US, namely residential, private non-residential, and public.

Based on this data, we are interested in the statistics that provide us evidence on *when* and *where* EV drivers are likely to request charging, and the duration of this process. Before The EV Project began collecting data, common wisdom had been that 80% of charge events for a typical driver would be at home [17]. Data collected at the EV Project seems to validate this [17]: The percentage of home charging for all regions appeared to stabilize at about 74% of all events for the Leaf and 80% for the Volt. Another important statistic given by the EV project is that EVs averaged 1.1 charging events (or requests) per day. For the Volt driver, the average was 1.5 charging events per day. Although Volt drivers charge their vehicles more often, they tend to charge at home.

These statistics from the ECOTality project are also correlated to the use of the charging points at the infrastructure. In this sense, considering only residential areas, Figures 1a the availability distribution of charging points for weekdays. In the y-axis, it shows the maximum and minimum percentage of charging units connected (for charging) across time. The behavior is highly associated with the arrival and departure times of the EVs at their homes, but it is also influenced by the electricity tariff, in this case, the minimum electricity demand is observed at 06:00 approximately. This is directly related to the hours that many EV owners are going to their work and therefore stop the charging process. The opposite is happening at night because the EV owners prefer to charge their EVs at the hours with a lower tariff. On the other hand, the case of commercial areas can be seen in

2. http://avt.inl.gov/evproject.shtml



(a) Weekdays in Residential Areas (b) Weekdays in Commercial Areas

Fig. 1: Aggregated Charging Availability

Figure 1b, which depicts charging availability distribution at the charging points for weekdays, respectively for all regions.

From Figure 1a, it can be observed that data flows generated for the EV charging processes in a residential area are quite regular, i.e. min. and max. values are close to each other. EV charging at commercial areas is very irregular and bursty, as can be seen from Figure 1b, which can make any prediction of the expected computational resource demand to support such prediction difficult. Based on the assumptions in [17] for Smart metering needs, and for an anticipated number of c.1.7M million EVs in the UK by 2025 [5], overall data flow volumes between 30TB to 40TB will be generated on a "total EV population per annum" basis. Similar figures are expected for the use of smart meters within the EV scenario described in [20].

4 Computational Capacity Strategy

With the purpose of distributing energy, the electric power networks are arranged into different areas. An aggregator will be in charge of *multiple* electrical areas, having each area a potential number of connected EVs (each area supports up to a fixed maximum number of points for EVs to charge their batteries). Periodically (i.e. expected to be at 15 to 30 minute intervals), an aggregator will need to process a charging schedule for each area, prior to enacting the operational activity (i.e. actual EV battery charging operation), as described in Section 3.1: We define an input data item to be all the charging requests received for a given area. Our focus in this paper is on managing computational resources for the scheduling stage and to improve efficiency of use of such resources. For such a purpose, the aggregator will need a shared pool of computational resources, as depicted in Figure 2.

The objective for the EV scenario is to guarantee certain Quality of Service (QoS) response time or deadline on *average*, while minimizing the operational costs derived from using computational resources. The established QoS means that there will be some areas whose scheduling will be under / above the deadline. Ideally, one may want to reduce as much as possible the dispersion of completion time, so that most of areas have their schedules as close as possible to their deadline.On the other hand, a direct way of reducing costs is by minimizing the number of resources used for processing each charging request or job. For such a purpose, we propose to take advantage of the *slack time* of a job – the period for which the computation of such a job can be delayed without causing a QoS deadline violation. In this problem, the slack time of a job is the deadline minus the estimated execution time and its associated overheads (e.g., data transfer time). Using the slack time to delay the processing of a job requires being able to measure the actual time a job is waiting to be processed. Direct monitoring of the waiting time can be challenging in distributed computing systems, and as a result, we use LL to derive the waiting time from the queue size of pending jobs. Since slack can vary depending on the incoming charging requests and/or their computational complexity, we need a solution that can elastically adapt to these changes. In this way, we can autonomously minimize costs while still satisfying the required QoS levels.

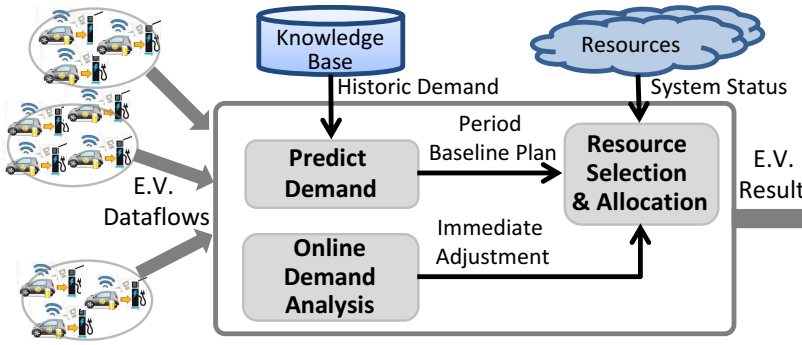


Fig. 2: System Model

5 Elastic Resource Management

To achieve the previously described capacity management strategy, we propose a reactive-predictive approach that can elastically manage the computational resources allocated at EV charging stations, as shown in Figure 2. On one hand, a predictive controller allows us to anticipate the future demand based on historical knowledge of the workload – thereby establishing a baseline plan for an entire electrical scheduling activity. With such a behavior, we alleviate current (systems) implementation overheads when setting up a new VM. On the other hand, a reactive controller allows us to perform online monitoring and analysis of the workload to dynamically adapt to (*unexpected*) changes in both the workload and the resources. Next, we model the two controllers that implement the strategies.

5.1 Predictive Controller

Various techniques have been developed to predict EV charging demand [21], [22] and their results show significantly high success prediction rates. For each scheduling

period T, our controller takes as inputs: (i) the predicted EV demand (any of the cited techniques can be used for this), (ii) computational cost of the scheduling algorithm, and (iii) historical information of VM performance, as input – and outputs the baseline number of VMs required for period T.

Analysis and Capacity Planning: for deriving the number of computational resources, the controller makes use of the traffic intensity, Eq. 1. By setting traffic intensity to 1 and solving for the number of computational resources, c, we obtain the number of VMs that can maintain the size of the queue without variation:

$$\hat{c_T} = \frac{\hat{\lambda_T}}{\hat{\mu_{u_T}}} \tag{4}$$

where $\hat{\lambda_T}$ is the estimated average input rate of data items, $\hat{\mu_{u_T}}$ is the *estimated* average output rate (throughput) per machine, and $\hat{c_T}$ is the estimated number of computational resources (VMs) for period T. It should be highlighted that setting traffic intensity to 1 does not lead to system instability, because our approach assumes that we always have enough resources available. We are just interested in the number of *active* resources and in switching them on and off depending on demand. An accurate prediction of computational resources, therefore, depends on $\hat{\lambda_T}$ and $\hat{\mu_{u_T}}$.

In the particular scenario of EVs, the input rate $\hat{\lambda_T}$ is typically constant, as it depends on the charging control periods (T can be typically fixed periods of 15 minutes). The value $\hat{\mu_{u_T}}$ depends on (i) the predicted EV demand, (ii) the scheduling algorithm, and (iii) the actual VM performance. In this case, as the predicted EV demand, we make use of the Support Vector Machine based EV demand technique described in [22]. As the scheduling algorithm, we make use of the optimization problem from Section 3.1. Finally, the values for $\hat{\mu_{u_T}}$ can be derived by building a performance knowledge for the type of VM. Such a mapping requires a previous experimentation that can be accomplished by running the algorithm with the historical records from ECO-Tality workload.

5.2 Reactive Controller

The reactive controller implements an autonomic MAPE (monitoring, analysis, planning and execution) loop [23].

Monitoring: the following system variables are monitored: (i) queue size L, which is the control output; (ii) the arrival rate for a data stream λ; and (iii) the processing time of each data element, which allows us to calculate S. These variables are recorded and computed every time a job enters or leaves the system to ensure our knowledge base is always up to date. Additionally, we have variables that are calculated periodically, which allow the reactive controller to take operational decisions. This includes the process of applying LL (based on average estimates), and allows the controller to act on L (average number of jobs in the queue) rather than on W (average wait time of a job in the queue). Average values may increase the response time of the reactive controller to changes in the environment. In order to mitigate the effects of long running averages and allow the system to rapidly react upon changes (i.e. bursty conditions of λ and unexpected performance of VMs), we propose moving averages (based on a time window of data).

In a moving average only a subsequence, a window, of a time series is considered for computing the average: older values of the series are being discarded with the arrival of new values. Thereby, the effect of long running averages is limited. Nevertheless, it may be challenging to determine the window size: if the window is too small, the derived values for W may tend to be biased, especially in cases where the processing time is too long. In contrast, if the window is too big, then the same problem of arithmetic averages appears. In our case, this issue is mitigated by the use of a predictive controller that always keep a long term view of the demand observed in the past as well as the predicted one.

It is also worth highlighting that the performance variation of computational resources is considered implicitly in our model by assuming the application can take an arbitrary time to process a data element.

Analysis: In this phase, we analyze demand and utilization data to establish the setpoint, i.e., the objective queue size $L*$ that allows us to minimize resource utilization while ensuring QoS requirements. We calculate the difference between the objective queue size ($L*$) and the monitored one (L*) to determine if an action is required. We have two possible actions: (i) remain idle, (ii) increase/decrease computational resources.

The average time any data element spends in the system is given by: $T = W + S$, where W is the average queueing time, and S is the average processing delay. We assume that the Quality of Service (QoS) for a data stream involves the processing of data elements within a deadline, δ, on average. Therefore, in order to enforce QoS, the following must be fulfilled: $T = W + S \leq \delta$.

As we are utilizing an elastic pay-as-you-go infrastructure, as a load balancing policy, we want to meet QoS while minimizing the number of computational resources. This policy is fully satisfied when on average $T_{max} = W_{max} + S = \delta$. Therefore, when $W_{max} = \delta - S$, we obtain the maximum time slack, i.e. the maximum time on average that a data element can spend in the queue without violating the QoS. Alternatively, with W_{max} and by applying LL, we can obtain a reference setpoint, the maximum average number of data elements in the queue, $L*$:

$$L* = \lambda W_{max} = \lambda(\delta - S) \tag{5}$$

It can be seen that $L*$ depends on λ, the arrival rate, and S, the average system processing time. Under variable and unpredictable workload and performance times, $L*$ is likely to vary unpredictably. Thus, the challenge for the controller is to make use of IaaS elastic actions, in order to maintain $L*$ within its objective value. After the reference point is set, the controller establishes upper and lower thresholds around it, incorporating hysteresis in the reference setpoint to avoid oscillatory behavior.

Capacity Planning: currently, we only consider horizontal scaling, i.e. the controller can provision and de-provision a discrete number of computational resources (VMs). We assume that the infrastructure is constrained by the number of concurrent VMs that can be provisioned – this number is set by the infrastructure provider and influenced by the number of physical machines and the number of VMs per physical machine. However, we assume that we always have enough resources. We also assume homogeneity in

the computational resources and consideration of multiple VM types, to achieve finer grain actions, is future work. In our approach, we adaptively regulate the number of provisioned VMs, c, such that the QoS of the data stream is enforced. In order to find the number of active VMs (c), we make use of the traffic intensity, ϱ, (which defines the relationship between input and output, as defined in Section 2). Setting traffic intensity to 1, and solving for c, we obtain the number of computational resources that enable output rate to match the input, i.e. $\lceil c \rceil = \frac{\lambda}{\mu}$, with $c \in \mathbb{N}$. As we have a discrete number of resources, we round up to the nearest integer.

Therefore, the actions for decreasing/ increasing the waiting time corresponds to $c + \Delta c$ and $c - \Delta c$, respectively, where Δc represents the increment in the number of VMs. The lower the value of Δc, the slower the reaction of the controller. Ideally, the quickest reaction is desired, but if the control action is too large, it may lead to controller oscillations and instability. Instability provoked by switching rapidly and violently between different configurations (namely in this case, over-provisioning and under-provisioning states). Typically, the value of Δc is experimentally determined.

Execution: as with the predictive controller, this phase interacts with the middleware to make changes in the resource allocation.

5.3 Coordination of Controllers

The coordination policy we propose here aims at blending current knowledge (short-term view) and historical knowledge (long-term view) to achieve our objective while increasing the stability of our system. The proposed strategy is summarized in Table 1 and it consist of two steps:

1) *Establishing Baseline:* prior to the beginning of a scheduling period T, the predictive controller analyzes historical data and predicts the computational demands for period T. Once the predictive controller plans and executes an action, the reactive controller is disabled for a time period p, so that the action can influence system behavior (throughput). For instance, in case the action involves the launching of new VMs, there exist associated set up overheads and subsequent processing delays. The disable period p must be greater than or equal to delays introduced by such overheads. The advantage of using the predictive controller is that it can provision resources just-in-time, if planned conveniently in advance, thereby mitigating VM set up overheads.

2) *Adjusting to the Unexpected:* between period T and $T + 1$, the reactive controller monitors the computational demand as well as the throughput of the system to correct potential deviations from the plan implemented by the predictive controller. The reactive controller assumes no knowledge of the workload and it bases all its decisions on the observed data.

Whenever the reactive controller wants to take an action, we consider two situations: a) said action is within the boundaries of the predictive controller and is aimed a correcting the observed behavior to ensure the predicted baseline. In this case, we take said actions; and b) said action contradicts the predictive controller. In this case, it is better to ignore the reactive action and follow the predicted one, as we consider that the predictions can be taken with a higher degree of confidence, according to the prediction techniques on the workload developed at [21], [22], or at least the missed predictions will not have a big error.

6 Experimental Validation

The controllers described in this paper were integrated within the CometCloud system [24]. CometCloud is an autonomic framework for enabling real-world applications on software-defined federated cyberinfrastructure, including hybrid infrastructures integrating public & private Clouds and data-centers. The overarching goal of CometCloud is to realize a software-defined federation with cloud abstractions that offer resources in an elastic and on-demand way, supporting the batch execution model. The role of the queues described in Section 5.2 is provided by CometCloud's tuplespace – more details in [25].

6.1 Experiment Methodology

A set of experiments were used to validate our approach. We consider a constant incoming data rate (λ) over time (representing each data element all the charging requests of a given area), and a variable workload based on the ECOTality data [17]. We assume that the EV aggregator manages a higher number of residential areas than commercial areas. Therefore, the variation of demand can be mostly explained by the behavior of EV owners and their habits as observed in the ECOTality project: the arrival and departure times of the EVs at their homes, also influenced by the electric tariff. In that case, the minimum electricity demand is observed at 06:00 approximately. This is directly related to the hours that many EV owners leave for work and results in stopping the charging process. The opposite is happening at night because the EV owners prefer to charge their EVs at the hours with a lower tariff. Based on such postulates, the processing time S (i.e. $S = \frac{1}{\mu}$) for the scheduling times vary accordingly as shown in Figure 3. In order to reduce the execution time of experiments, rather than considering periods T of 15 minutes, we also consider that the QoS established by the aggregator is such that each scheduled task stays for 33.3 seconds in the system. This means that the time a task can spend in the queue (W) varies as S changes. Moreover, the data produced in commercial charging areas is bursty and unpredictable, generating deviations on the expected demand. We analyze its influence by considering different degree of missed prediction in our experiments.

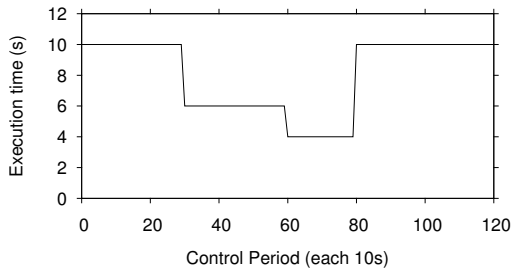


Fig. 3: Variation of the Processing Time for Scheduling Periods

TABLE 1: Coordination Policy for the Controllers

Event	Condition	Action
Predictive contr. takes action *a*	True	Reactive contr. disabled for period *s*
Reactive contr. takes action *a*	Predictive contr.'s baseline *outside* the boundaries of action *a*	Predictive contr. has priority
Reactive contr. takes action *a*	Predictive contr.'s baseline *within* the boundaries of action *a*	Reactive contr. has priority

We have deployed a testbed between the Universidad de Zaragoza (Spain) and Rutgers University (USA). In this testbed, we have the autonomic control layer at Zaragoza and the resource management layer (CometCloud) at Rutgers. The computational resource layer in Zaragoza consists of a single dedicated machine 2.6 GHz Intel Core i5 with 16 GB 1600 MHz DDR3. The computational resources located at Rutgers include 32 dedicated machines within a cluster. Each node has 8 cores, 6 GB memory, 146 GB storage and a Gigabit Ethernet connection. The resources of our cluster are offered using a cloud abstraction, enabling their provision/de-provision on-demand. Since we have an HPC cloud, the overhead of provisioning machines in this infrastructure is not significant and therefore not considered. The measured latency on the internal network is 0.227ms on average. The interconnection network overhead between Zaragoza and Rutgers is 130 ms on average.

6.2 Experiments

We want to validate the performance of our hybrid controller in comparison with a purely reactive controller in different scenarios: (i) 100% accuracy of prediction and (ii) different degrees of missed predictions. We have performed a number of of experiments using a sliding window of 20 data values to calculate moving averages. In order to validate our metrics, we have computed a baseline case where the number of VMs required to maintain the queue size (L) to zero are provisioned. In such a baseline, all of the jobs meet the deadline (except at the initial time as the system is empty), but the number of VM hours is maximized. We compared such a baseline with the baseline of our proposal, and we also measured the number of jobs in our proposal that are not meeting the deadline.

6.2.1 Purely Reactive Controller vs Hybrid Controller

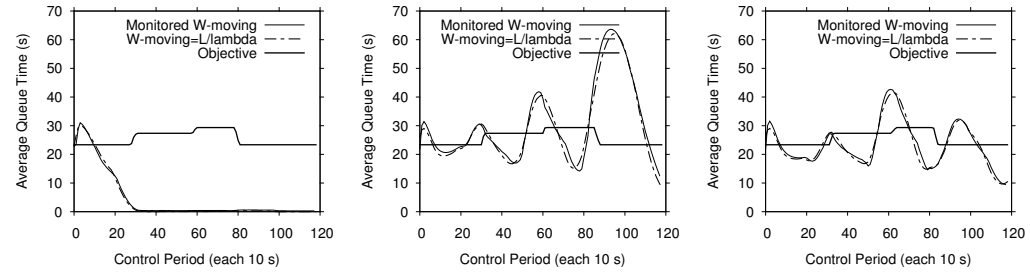
In the first experiment, we compare the performance of our hybrid controller with a purely reactive controller. We assume a perfect prediction of the required computational resources (i.e. 100% hit). Since measuring the queue time for each data element in our system can be non-trivial, we use the number of elements in the queue and the arrival rate to estimate the queue waiting time W (see Equation 5). By using this information, the reactive controller can determine the number of machines to allocate or release, see Section 5. In these experiments, we allocate or release two VMs over or under the equilibrium point (Δc) to increase or reduce the queue size. Moreover, we consider that the system is not empty at the beginning and has 12 VM instances, i.e. 2 VMs over the equilibrium point. Additionally, we use a baseline strategy that, unlike previous strategies, does not make use of the slack. We call this strategy *no slack*. Figure 4 collects the experimental results.

Figure 4a & 4d presents the results of our baseline experiment. As before, we observe that the number of elements in the queue tends to zero and the amount of allocated machines is maximized. Figures 4b & 4e present the results of the experiment for the purely reactive controller, whereas Figures 4c & 4f present the results of the experiment for the hybrid controller. Both graphs (Figures 4b & 4c) depict how the queue waiting time evolves over time. We can observe that the "W=L/λ" (W calculated using LL) and the "Monitored W" (the actual measured W) oscillate around the "objective waiting time", as expected by LL. It should also be noticed that the waiting time W also evolves in accordance with the processing time S from Figure 3.

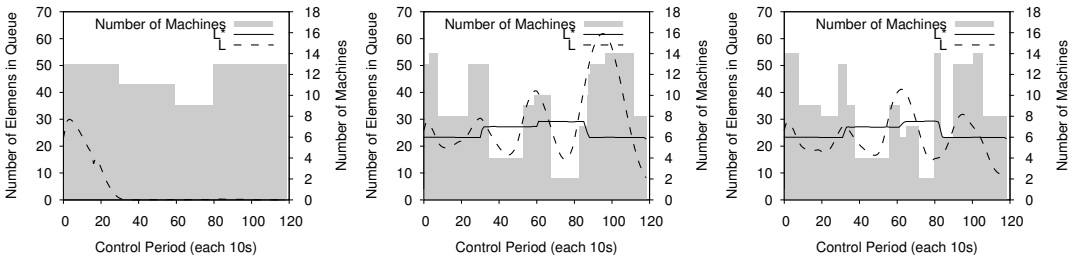
We can also observe the evolution of the queue size over time and the number of VMs involved. At the beginning, the system needs to process the peak workload, then it decreases at control period 25 (time 250s) and it gets its minimum workload by control period 60 (time 600s). Finally, at control period 80, there is a huge peak in the workload again to the peak at the end. Both controllers allocate and de-allocate machines to achieve the objective waiting time. Thus, when the workload decreases both de-allocate VMs and react similarly. This is due to the fact that the action of de-allocating VMs has no significant overhead and can take place almost instantaneously. However, the most significant difference between both controllers happens at control period 80, when the workload goes from low to the peak. The increase in the workload is poorly managed by the purely reactive controller. In spite of the fact that the peak starts at control period 80, it triggers action at control period of 90. As we are utilizing average values, the consequence of this is that in order to overcome high values of L for a period of time, the reactive controller has to enforce low values of L for a period of time to compensate, and this generates significant oscillation. In contrast, the hybrid controller can anticipate the event and lead to 14 VMs being launched at control period 80, which are then maintained for some time. As a consequence, the oscillation around the objective L is much less compared to the hybrid controller. This arises due to an overhead associated with allocating VMs and the processing delay – thereby limiting instantaneous action. Therefore, the behavior of the hybrid controller outperforms that of the reactive controller because the hybrid controller can anticipate future variations in the workload.

6.2.2 Hybrid Controller with Missed Predictions

We repeat similar experiments with the hybrid controller, but introduce an error in predicting the number of required



(a) Queue Waiting Time - No Slack (b) Queue Waiting Time - Purely Reactive (c) Queue Time - Hybrid 100% Prediction Hit



(d) Queue Length - No Slack (e) Queue Length - Purely Reactive (f) Queue Length - Hybrid 100% Prediction Hit

Fig. 4: Summary of experiment - Fixed Execution Time. Action is $\pm$ 2 VMs over the optimum.

VMs: a deviation of 20% over the resources required (Figures 5a & 5c) and a deviation of 20% under the resources required [3] (Figures 5b & 5d). Analogously, we can observe how "W=L/λ" (W calculated using LL) and the "Monitored W" (the actual measured W) oscillates around the objective waiting time, as expected by LL. The waiting time W also evolves in accordance with the processing time S from Figure 3, and the hybrid controller allocates and de-allocates VMs in order to achieve the set objective.

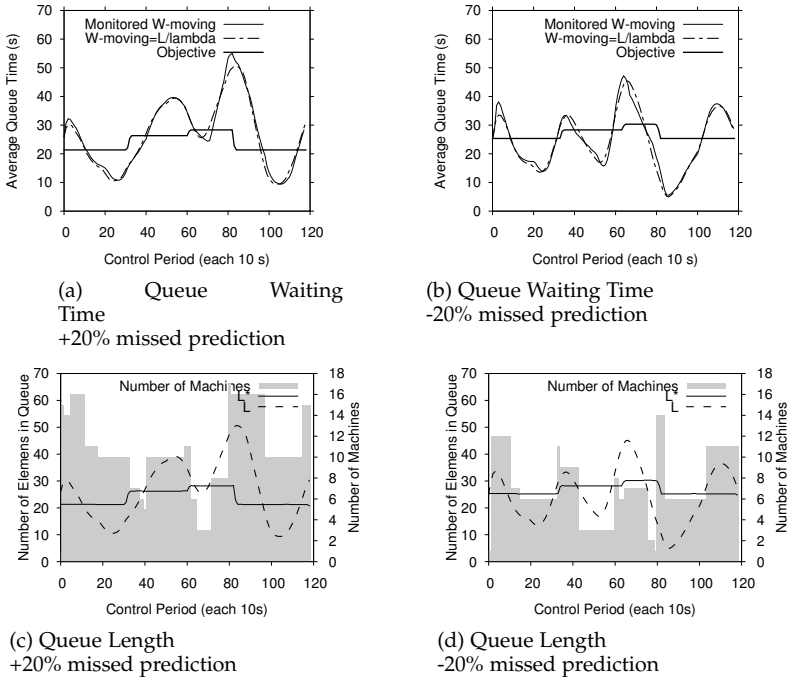
In these experiments, as before, we can also observe the evolution of the queue size and the number of VMs involved. It can be observed that the hybrid controller with a perfect prediction performs better than with deviations in the prediction of the required resources: The oscillations around the target are smaller and few VMs are involved. In contrast, in comparison with the purely reactive controller, the hybrid controller still oscillates less even with missed predictions. There are two potential reasons for such behavior: (i) if the number of allocated VMs exceeds the required number, then eventually they will be removed (this can be seen, for instance, in Figure 5d between control periods 20 to 40 – time 200s to 400s); (ii) if the number of allocated VMs does not reach the required number, then the performance (i.e. accuracy of prediction) will decrease. The hybrid controller with 20% of missed prediction under the actual required VMs performs reasonable well, as it is setting up most of the required VMs beforehand. The worst case scenario is that of the reactive controller, which delays allocation of VMs in these instances.

3. It should be noted that the SVN-based energy demand prediction technique presented in [22] shows a 4.6% Mean Absolute Percentage Error. Based on our estimation of resources described in Section 5.1, we are here, therefore, assuming a worst case scenario of 20% of deviation in the number of VMs

6.3 Completion Time Analysis

In this Section, we analyze the completion time of the data elements processed in each one of the experiments executed before. In this paper, we proposed an approach to leverage the slack of jobs (i.e. remaining of deadline minus execution time and overheads), aiming at minimizing the amount of resources provisioned to satisfy the workload given a specific Service Level Agreement (SLA). In our case, we chose that our SLA was to meet the data elements' deadline on average. Next, we evaluate two specific metrics: a) the amount of resources used, and b) the SLA assurance. Fig. 6 collect these results.

In previous experiments, we observed oscillations around the target value. We can observe now how the median completion time for the baseline approaches is typically very far from the deadline. However, in the rest of the cases, all our strategies show how the median completion time is very close to the deadline. This means that when the slack is not used, we are wasting a significant amount of resources. Specifically, in our experiments, in comparison with the baseline (*no slack*), our only reactive approach saved up to a 31 % of machine hours in the experiments (around 1.3 machine hours), whereas the hybrid controller saved between up to 34 % in the experiments (up to 1.3 machine hours). Additionally, the median of the data elements' completion time also tells us that the proposed SLA is satisfied. The dispersion of the completion time is smaller for the hybrid controller than for the purely reactive controller, which is due to the predictive anticipation behavior that overcomes the existing overheads. This means that the oscillations around the target are reduced: The interquartile range for the reactive controller is 30.41, whereas for the hybrid controller ranges from 20.38 to 29.85. The hybrid controller with a 100% of prediction success achieves the smallest in-



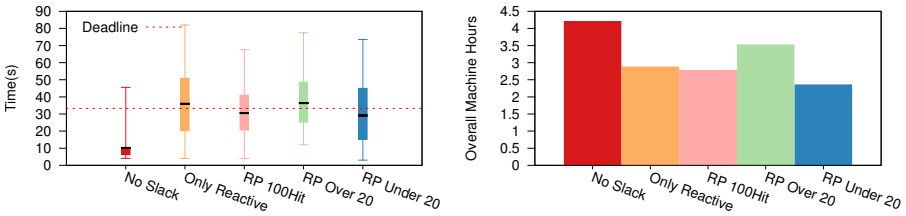
(a) Queue Waiting Time +20% missed prediction

(b) Queue Waiting Time -20% missed prediction

(c) Queue Length +20% missed prediction

(d) Queue Length -20% missed prediction

Fig. 5: Hybrid Controller with missed predictions - Action is $\pm$ 2 machines over the optimum.



(a) Job Completion Time

(b) Overall Machine Hours

Fig. 6: Completion Time Analysis Summary.

terquartile range, whereas the hybrid controller with a 20% of under deviation performs the worst. This is due to the fact that the predictions lead to allocate fewer machines than required, and therefore the anticipation is less successful in this scenario. Finally, it is also worth highlighting that our approach managed to complete around 50% of the jobs within the deadline, and to enforce a small interquartile range. Therefore, the values that are above the deadline are delayed only a few seconds, which is acceptable for the subsequent operational stage.

7 Related Work

The two biggest fields in which LL has been applied are operations management (OM), and computer science and engineering (CSI) [14]. The approach has also been made use of in various case studies, ranging from computer architecture to computer networks and distributed systems. The most significant case studies from OM and CSI are described in [14].

Significant literature also exists related to work in deadline based scheduling algorithms for multimedia applications in communication networks. In general terms, the function of these scheduling algorithms is to select the session whose head-of-line (HOL) packet is to be transmitted next through the network. This process is based on the QoS requirements. A survey that provides an overview can be found in [26], [27]. In other words, these proposals aim to evenly share the workload (packets) onto a shared resource (the network). In contrast, in our approach, we adapt the available computational capacity (resources) to the workload. We achieve this by provisioning the number of computational resources to a data stream in accordance to predictions, and workload variations and resource performance.

A number of studies developed autonomic policies and mechanisms for elasticity in clouds. In [16], the AGILE system provides medium-term resource demand predictions for achieving enough time to scale up the computational resources in advance, minimizing VM launching overheads. In comparison to our approach, AGILE is application agnostic and does not consider the characteristics of streaming applications. For the NoSQL cluster scenario, TIRAMOLA was presented in [28]. It can self-resize a NoSQL cluster according to user-defined policies. Decisions on (de-

)allocating VMs from a cluster are modeled as a Markov Decision Process and taken in real-time. Autoflex [15] is a service agnostic system for autonomic scaling of VMs that combines both reactive and proactive approaches. A purely reactive resource provisioning approach was proposed in [29] under the YinzCam system, which provides cloud-hosted service for real-time Web scores, news, etc. The workload considered exhibits significant spikes. Hence, the controller is designed so that the scaling up action is done much faster than the scaling down. Again, this approach is designed to enable service operations to be on-line and responding without any time-slack. More recently, vertical scalability was also studied in [30] by means of different performance models that enable mapping performance to capacity. Autonomic computing was studied to provide opportunistic in-transit processing by taking advantage of the estimated "slack" that is available at different stages of a data-intensive workflow [31].

The increasing deployment of sensor network infrastructures has led to large volumes of data becoming available, leading to new challenges in storing, processing and transmitting such data [32]. For that reason, stream processing frameworks such as Yahoo's S4 [33], or IBM InfoSphere Streams [34] provide streaming programming abstractions to build and deploy jobs as distributed applications at scale for commodity clusters and clouds. Nevertheless, even that these systems support high input data rates, they do not consider variability in workload and unexpected performance of resources, which is our focus in this paper. In some other approaches, the parallelism is extracted from the data stream query operators they provide, Aurora [35], Borealis [36] and Stream Cloud [37], which differs that in our case, we explicitly exploit the parallelism by having multiple data elements in execution. In this area, Spark has popularized the idea of discretized streams to process streams as a sequence of discrete micro-batches, which improves fault recovery [38]. These micro-batches are dynamically allocated across workers based on data locality and availability. While Spark assumes a ready-to-use cluster of workers, our autonomic approach intends to elastically provision and deprovision machines to minimize the operational costs of processing the streams.

Our work is closely related to three approaches. In [39], the goal is to allocate resources dynamically from a cloud, so that the processing rate can match the rate of data arrival. They also consider variable transient input rates. Our approach is more general, as such a case corresponds in our approach to a scenario where the time slack is zero. Moreover, we make use of a federation of heterogeneous resources and we propose autonomic based mechanisms and policies for the selection of resources. In [40], the authors propose a workflow specification where each job consists of one or more alternate implementations with different non-functional properties, so that the system can choose any of them dynamically at runtime. In this paper, we have not considered dynamism at workflow-level, but our dynamic provisioning of resources is accomplished in a federation of heterogeneous resources. Finally, the work in [41], [42], [43] consists of a sequence of nodes, where each node has multiple data buffers and computational resources – whose numbers can be adjusted in an elastic way. They utilize the token bucket model for regulating data injection rates into such nodes. As before, they do not consider time slacks. Another important difference to our approach is that instead of utilizing multiple nodes, we assume CometCloud system as a coordination mechanism that can outsource the computation when required.

Finally, the problem of detecting spikes in cloud workloads can be beneficial not only for the purpose in this paper of resetting monitoring average values and starting a new monitored period, but also for making proactive resource management decisions. Indeed, unanticipated changes in workload characteristics can potentially lead to service slowdown and might end in service-failure due to insufficient resource allocation. In [44], the authors investigate methods for detecting spikes in cloud workloads. In particular, they developed methods that make use of signal processing techniques. Previous efforts have also been made on modeling and characterizing workloads and spikes. The work in [45] presents a detailed workload characterization study of the 1998 World Cup Web site. In [46], the authors analyze a number of real workload and data spikes and from the results they propose and validate a model of stateful spikes that allow them to synthesize volume and data spikes, and that can be used for cloud providers.

8 Discussion

The results presented in this paper show how our controller is able to react to changes in the EV charging demand. We have observed that when using our resource management approach, the waiting time experienced by jobs oscillates around the QoS target, which could lead to an inaccurate or delayed estimate of EV demand. This could be because (i) the number of VMs needed to achieve a desired waiting time cannot be estimated accurately a priori, thus requiring dynamic (de-) provisioning VMs, or (ii) when charging demand changes, the controller has to deal with "inertia" that delays the effect of our control actions. Such an "inertia" is influenced by the effect that average values can have and the actual processing time, which, once an action is taken, delays the effect of the action on the queue size.

In order to eliminate or mitigate such effects, we propose the following strategies. If the waiting times over the target are not acceptable, a higher number of VMs can be provisioned based on the available *slack*, as explained in Section 5. There is, therefore, a trade-off between operational cost (number of VMs) and performance. In our experiments we demonstrate how the maximum allowed slack can be used to minimize the number of resources used, whereas in [25] we show how a system would react when not using slack at all. Another factor to take into account is the effect of long running averages, which does not allow the system to rapidly react. To address this issue, we propose to reduce the data used to take operational decisions. Two alternative approaches can be taken, viz., to reset average values periodically or to compute moving averages. In this work, we show how a moving average allows us to react in a timely manner to changes in the workload while keeping the system stable. Finally, the overheads involved in the control actions can only be avoided by adding a predictive controller. In this work, we were based our predictions on

historical information. However, we also highlighted that the coexistence of two autonomic controllers acting on the same variable (number of VMs) leads to the establishment of coordination policies, which if not defined carefully could hinder performance.

9 Conclusions

In this paper, we propose an approach that optimizes the computational resource management for distributed data-driven applications. We validate our approach by managing the computational demands of a distributed controlled EV battery charging process within a single geographical level. This scenario considers that an EV aggregator is in charge of managing electricity demand over a large number of areas, therefore requiring a pool of computational resources to estimate charging demand. Resource management in the context of this work involves dynamic allocation of VMs and queueing theory. When the charging requests for each area arrive at the EV aggregator, it computes a charging scheduling for each area. This is achieved by means of a controller that automatically allocates/deallocates VMs in accordance with the number of EVs. The controller monitors input rates and execution times, periodically computes the target waiting times (queue sizes), and reactively regulates allocated VMs accordingly to enforce computations in a timely manner. Nevertheless, due to current maturity of technologies, the cloud paradigm has to face significant overheads in the VM (de-)provisioning process. Our controller is also designed to leverage demand prediction models (e.g., based on historical records), so that it can anticipate variations in the charging demand, deriving the number of VMs required and, thus, minimizing the provisioning overheads. A coordination policy has been proposed in order to guarantee that both autonomic behaviors cooperate towards achieving the desired objective. We implemented our approach on top of the CometCloud system to support Cloud federation and validated it using trace data of EV charging demands from the ECOTality project.

Although we focused our computational findings on the EV charging challenge, our approach can also be generalized to other *medium to high latency* applications in areas such as surveillance and monitoring [1], *smart*-traffic management, or energy management in smart building [2]. In all these domains, similar applications that share requirements can be found: data is received from sensors periodically, response times need to be in the order of seconds, minutes, or even hours, and their workloads are typically coarse-grained and more complex, hence involving more computational resources, often executed as *batch* processes. Currently, we are working on a more general predictor model, based on time series analysis, suitable for a broader range of applications. Moreover, we are exploring how to consider other non-functional objectives such as energy consumption.

Acknowledgment

This work was supported in part by: the Spanish Ministry of Education, (Framework Program CEI Iberus – Universidad de Zaragoza, for faculty staff, mobility call 2014), the Spanish Ministry of Economy (program "Programa de I+D+i Estatal de Investigación, Desarrollo e innovación Orientada a los Retos de la Sociedad" –TIN2013-40809-R), NSF via grants numbers ACI 1339036, ACI 1441376. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI2).

References

[1] A. Anjum, T. Abdullah, M. Tariq, Y. Baltaci, and N. Antonopoulos, "Video stream analysis in clouds: An object detection and classification framework for high performance video analytics," *IEEE Transaction on Cloud Computing*, 2016 - to appear.

[2] I. Petri, O. Rana, Y. Rezgui, H. Li, T. Beach, M. Zou, J. Diaz-Montes, and M. Parashar, "Cloud supported building data analytics," in *CCGrid*, 2014, pp. 641–650.

[3] C. Herath and B. Plale, "Streamflow programming model for data streaming in scientific workflows," in *CCGrid*, 2010, pp. 302–311.

[4] Y. Simmhan, B. Cao, M. Giakkoupis, and V. K. Prasanna, "Adaptive rate stream processing for smart grid applications on clouds," in *Intl. workshop on Scientific cloud computing*, 2011, pp. 33–38.

[5] F. McMorrin, R. Anderson, I. Featherstone, and C. Watson, "Plugged-in fleets: A guide to deploying electric vehicles (EVs) in fleets," The Climate Group, Tech. Rep., February 2012.

[6] P. Papadopoulos, O. Akizu, L. Cipcigan, N. Jenkins, and E. Zabala, "Electricity demand with electric cars in 2030: comparing Great Britain and Spain," *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, vol. 225, no. 5, pp. 551–566, 2011.

[7] E. L. Karfopoulos, C. E. Marmaras, and N. Hatziargyriou, "Charging control model for electric vehicle supplier aggregator," in *3rd IEEE ISGT Europe*, 2012.

[8] E. Sortomme, M. Hindi, S. MacPherson, and S. Venkata, "Coordinated charging of plug-in hybrid electric vehicles to minimize distribution system losses," *Smart Grid, IEEE Transactions on*, vol. 2, no. 1, pp. 198–205, 2011.

[9] S. Deilami, A. Masoum, P. Moses, and M. A. S. Masoum, "Real-Time Coordination of Plug-In Electric Vehicle Charging in Smart Grids to Minimize Power Losses and Improve Voltage Profile," *Smart Grid, IEEE Transactions on*, vol. 2, no. 3, pp. 456–467, 2011.

[10] E. Karfopoulos and N. Hatziargyriou, "A multi-agent system for controlled charging of a large population of electric vehicles," *Power Systems, IEEE Transactions on*, vol. 28, no. 2, pp. 1196–1204, 2013.

[11] P. Papadopoulos, "Integration of electric vehicles intro distribution networks," Ph.D. dissertation, Cardiff University, 2012.

[12] I. Grau, P. Papadopoulos, S. Skarvelis-Kazakos, L. M. Cipcigan, N. Jenkins, and E. Zabala, "Management of electric vehicle battery charging in distribution networks with multi-agent systems," *Electric Power Systems Research*, 2014.

[13] E. L. Karfopoulos and N. D. Hatziargyriou, "A multi-agent system for controlled charging of a large population of electric vehicles," *Power Systems, IEEE Transactions on*, vol. 28, no. 2, pp. 1196–1204, 2013.

[14] J. D. C. Little, "OR FORUM - little's law as viewed on its 50th anniversary," *Operations Research*, vol. 59, no. 3, pp. 536–549, 2011.

[15] F. J. A. Morais, F. V. Brasileiro, R. V. Lopes, R. A. Santos, W. Satterfield, and L. Rosa, "Autoflex: Service agnostic auto-scaling framework for iaas deployment models," in *CCGrid, Delft, Netherlands*, 2013.

[16] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "AGILE: elastic distributed resource scaling for infrastructure-as-a-service," in *ICAC'13, San Jose, CA, USA*, 2013, pp. 69–82.

[17] S. Schey, "Q2 2013 report – the ev project," ECOTality North America, Tech. Rep., 2013.

[18] S.-H. Kim and W. Whitt, "Statistical analysis with little's law," *Operations Research*, vol. 61, no. 4, pp. 1030–1045, 2013.

[19] R. Tolosana-Calasanz, J. Á. Bañares, O. F. Rana, C. Pham, E. Xydas, C. E. Marmaras, P. Papadopoulos, and L. Cipcigan, "Enforcing quality of service on opennebula-based shared clouds," in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014*, 2014, pp. 651–659.

[20] J. Gao, Y. Xiao, J. Liu, W. Liang, and C. P. Chen, "A Survey of Communication/Networking in Smart Grids," *Future Gener. Comput. Syst.*, vol. 28, no. 2, pp. 391–404, Feb. 2012.

[21] E. Xydas, C. Marmaras, L. M. Cipcigan, N. Jenkins, S. Carroll, and M. Barker, "A data-driven approach for characterising the charging demand of electric vehicles: A uk case study," *Applied Energy*, vol. 162, pp. 763–771, 2016.

[22] S. Xydas, C. E. Marmaras, L. M. Cipcigan, A. Hassan, and N. Jenkins, "Electric vehicle load forecasting using data mining methods," in *Hybrid and Electric Vehicles Conference, IET*. IET, 2013, pp. 1–6.

[23] M. Parashar and S. Hariri, "Autonomic computing: An overview," in *Unconventional Programming Paradigms*. Springer Berlin Heidelberg, 2005, pp. 257–269.

[24] J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar, "Comet-cloud: Enabling software-defined federations for end-to-end application workflows," *IEEE Internet Computing*, vol. 19, no. 1, pp. 69–73, 2015.

[25] R. Tolosana-Calasanz, J. Diaz-Montes, O. F. Rana, and M. Parashar, "Extending cometcloud to process dynamic data streams on heterogeneous infrastructures," in *Intl. Conf. on Cloud and Autonomic Computing (ICCAC)*, 2014.

[26] H. Fattah and C. Leung, "An overview of scheduling algorithms in wireless multimedia networks," *Wireless Communications, IEEE*, vol. 9, no. 5, pp. 76–83, 2002.

[27] R. Guérin and V. Peris, "Quality-of-service in packet networks: basic mechanisms and directions," *Computer Networks*, vol. 31, no. 3, pp. 169–189, 1999.

[28] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris, "Automated, elastic resource provisioning for nosql clusters using TIRAMOLA," in *CCGrid, Delft, Netherlands*, 2013.

[29] N. D. Mickulicz, P. Narasimhan, and R. Gandhi, "To auto scale or not to auto scale," in *ICAC, San Jose, CA*, 2013, pp. 145–151.

[30] E. B. Lakew, K. Cristian, H.-R. Francisco, and E. Erik, "Towards faster response time models for vertical elasticity," in *IEEE/ACM UCC, London, UK*, 2014, pp. 560–565.

[31] V. Bhat, "Autonomic management of data streaming and in-transit processing for data intensive scientific workflows," Ph.D. dissertation, Rutgers University, 2008.

[32] L. Golab and M. T. Özsu, "Issues in data stream management," *SIGMOD Rec.*, vol. 32, no. 2, pp. 5–14, 2003.

[33] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *IEEE Intl. Conf. on Data Mining Workshops (ICDMW)*, 2010, pp. 170–177.

[34] A. Biem, E. Bouillet, H. Feng *et al.*, "Ibm infosphere streams for scalable, real-time, intelligent transportation services," in *ACM SIGMOD Intl. Conf. on Management of Data*, 2010, pp. 1093–1104.

[35] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik, "Scalable Distributed Stream Processing," in *1st Conf. on Innovative Data Systems Research (CIDR)*, Asilomar, CA, 2003.

[36] D. J. Abadi, Y. Ahmad, M. Balazinska *et al.*, "The Design of the Borealis Stream Processing Engine," in *2nd Conf. on Innovative Data Systems Research (CIDR)*, Asilomar, CA, 2005.

[37] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, and P. Valduriez, "Streamcloud: A large scale data streaming system," in *IEEE Intl. Conf. on Distributed Computing Systems*, 2010, pp. 126 –137.

[38] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *ACM Symp. on Operating Systems Principles*, 2013, pp. 423–438.

[39] S. Vijayakumar, Q. Zhu, and G. Agrawal, "Dynamic resource provisioning for data streaming applications in a cloud environment," in *IEEE CloudCom*, 2010, pp. 441–448.

[40] A. G. Kumbhare, Y. Simmhan, and V. K. Prasanna, "Exploiting application dynamism and cloud elasticity for continuous dataflows," in *SC'13, Denver, CO, USA*, 2013.

[41] R. Tolosana-Calasanz, J. A. Bañares, and O. F. Rana, "Autonomic streaming pipeline for scientific workflows," *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 16, pp. 1868–1892, 2011.

[42] J. Á. Bañares, O. F. Rana, R. Tolosana-Calasanz, and C. Pham, "Revenue creation for rate adaptive stream management in multi-tenancy environments," in *GECON*, 2013, pp. 122–137.

[43] R. Tolosana-Calasanz, J. Á. Bañares, C. Pham, and O. F. Rana, "Enforcing qos in scientific workflow systems enacted over cloud infrastructures," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1300–1315, 2012.

[44] A. Mehta, J. Durango, J. Tordsson, and E. Elmroth, "Online spike detection in cloud workloads," in *IEEE Intl. Conf. on Cloud Engineering, IC2E, Tempe, AZ, USA*, 2015, pp. 446–451.

[45] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE Network*, vol. 14, no. 3, pp. 30–37, 2000.

[46] P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *1st ACM Symposium on Cloud Computing (SoCC), Indianapolis, Indiana, USA*, 2010, pp. 241–252.