Modeling and Extracting Load Intensity Profiles

JÓAKIM VON KISTOWSKI, NIKOLAS HERBST and SAMUEL KOUNEV,
University of WÜRZBURG
HENNING GROENDA and CHRISTIAN STIER, FZI Forschungszentrum Informatik, Karlsruhe
SEBASTIAN LEHRIG, s-lab – Software Quality Lab, Paderborn University

Today's system developers and operators face the challenge of creating software systems that make efficient use of dynamically allocated resources under highly variable and dynamic load profiles, while at the same time delivering reliable performance. Autonomic controllers, for example, an advanced autoscaling mechanism in a cloud computing context, can benefit from an abstracted load model as knowledge to reconfigure on time and precisely. Existing workload characterization approaches have limited support to capture variations in the interarrival times of incoming work units over time (i.e., a variable load profile). For example, industrial and scientific benchmarks support constant or stepwise increasing load, or interarrival times defined by statistical distributions or recorded traces. These options show shortcomings either in representative character of load variation patterns or in abstraction and flexibility of their format.

In this article, we present the *Descartes Load Intensity Model* (DLIM) approach addressing these issues. DLIM provides a modeling formalism for describing load intensity variations over time. A DLIM instance is a compact formal description of a load intensity trace. DLIM-based tools provide features for benchmarking, performance, and recorded load intensity trace analysis. As manually obtaining and maintaining DLIM instances becomes time consuming, we contribute three automated extraction methods and devised metrics for comparison and method selection. We discuss how these features are used to enhance system management approaches for adaptations during runtime, and how they are integrated into simulation contexts and enable benchmarking of elastic or adaptive behavior.

We show that automatically extracted DLIM instances exhibit an average modeling error of 15.2% over 10 different real-world traces that cover between 2 weeks and 7 months. These results underline DLIM model expressiveness. In terms of accuracy and processing speed, our proposed extraction methods for the descriptive models are comparable to existing time series decomposition methods. Additionally, we illustrate DLIM applicability by outlining approaches of workload modeling in systems engineering that employ or rely on our proposed load intensity modeling formalism.

CCS Concepts: • **General and reference → Cross-computing tools and techniques**; • **Networks → Cloud computing**; • **Computer systems organization → Self-organizing autonomic computing**; • **Software and its engineering → Virtual machines;**

Additional Key Words and Phrases: Load intensity variation, load profile, open workloads, metamodeling, transformation, model extraction

This research has been supported by the Research Group of the Standard Performance Evaluation Corporation (SPEC). This work was funded by the German Research Foundation (DFG) under grant No. KO 3445/11-1 and parts (cf. Chapter 6) from the European Union's Seventh Framework Programme (FP7/2007- 2013) under grant agreement nos. 317704 (CloudScale) and 610711 (CACTOS).

Authors' addresses: J. von Kistowski, N. Herbst, and S. Kounev, Chair for Software Engineering, University of Würzburg, Am Hubland, 97074 Würzburg, Germany; emails: {joakim.kistowski, nikolas.herbst, samuel.kounev}@uni-wuerzburg.de; H. Groenda and C. Stier, FZI Forschungszentrum Informatik, Haid-und-Neu-Str. 10, 76131 Karlsruhe, Germany; emails: {groenda, stier}@fzi.de; S. Lehrig, Professur Softwaretechnik, Chemnitz University of Technology, Straße der Nationen 62, 09107 Chemnitz, Germany; email: sebastian.lehrig@uni-paderborn.de.

DOI: http://dx.doi.org/10.1145/3019596

ACM Reference Format:
Jóakim von Kistowski, Nikolas Herbst, Samuel Kounev, Henning Groenda, Christian Stier, and Sebastian Lehrig. 2017. Modeling and extracting load intensity profiles. ACM Trans. Auton. Adapt. Syst. 11, 4, Article 23 (January 2017), 28 pages.
DOI: http://dx.doi.org/10.1145/3019596

1. INTRODUCTION

Today's cloud and web-based IT services need to handle large numbers of concurrent users under highly variable and dynamic load intensities. Customers access services independently of each other and expect a stable Quality-of-Service (QoS). In this context, any knowledge about a service's load intensity profile and their variations becomes crucial information for managing the underlying IT resource landscape. Human behavior patterns due to human habits, trends, calendar effects, and events heavily influence load profiles. An autonomic controller, for example, an advanced autoscaling mechanism deployed in a cloud computing context, may implement the MAPE-K control loop [Kephart and Chess 2003]. When such a controller is constantly provided with abstracted knowledge about the observed and expected load profile, it could trigger the majority of adaptations more precisely and on time, as envisioned by the Models@Run-Time community [Blair et al. 2009].

Also, performance evaluation of systems under dynamic load conditions poses new challenges. Benchmarking frameworks such as Faban [2006], Rain [Beitch et al. 2010], and JMeter [Halili 2008] allow request injection rates to be configured either to constant values, or to stepwise increasing rates (e.g., for stress tests). The feature of generating variable rates based on a recorded or synthetic load trace is not fully supported by the mentioned frameworks. They usually work with a fixed number of load generating threads, whereas one of those corresponds to one virtual user with its routine and think time, as in a closed workload. From this, we see that open workloads with a possibly unlimited number of users and a representative load intensity variation independent of the actual system's performance are supported only to a limited extent by existing benchmarking frameworks.

In this article, we introduce the *Descartes Load Intensity Model* (DLIM) and the corresponding tools. DLIM describes load profiles by combining piecewise mathematical functions in a tree-like manner. Manual construction and maintenance of DLIM model instances becomes infeasible in complex scenarios or at runtime usage. We address this by proposing the *high-level Descartes Load Intensity Model* (hl-DLIM) to support the description of load variations using a small set of parameters to characterize seasonal patterns, trends, as well as bursts and noise. The initial ideas for these two models have been presented in von Kistowski et al. [2014]. A first sketch of the DLIM metamodel without details on several abstract model elements can be found there. The work-in-progress description does not include in-depth validation of the models, model extraction, and later improvements to the models.

DLIM can be used to define an arbitrary load intensity profile, which can than be leveraged for benchmarking purposes to evaluate the behavior of a system under different dynamic workload scenarios (e.g., bursty workloads, seasonal patterns). This is useful in several use cases, for example, for both online and offline evaluation of the quality of system adaptation mechanisms such as elastic resource provisioning techniques in modern cloud environments. In contrast to pure regression approaches, DLIM offers the advantage of classifying load intensity variations by type, as they are fitted to certain model elements. As a result, models include additional information on types, which is useful when analyzing or modifying load intensity variations.

A load intensity profile, represented as a DLIM model instance, can be created either manually by the user or it can be extracted from a request arrival trace obtained by monitoring a real-life production system. To support this, we provide generic trace analysis algorithms, define quality metrics for these algorithms, and integrate them into the DLIM tools called LIMBO.[1] The tools allow users to use our approach in a variety of use cases. As a result, the trace is represented as a compact DLIM model instance carrying a tree of mathematical functions that are combined over the modeled time. The model instance captures the major properties of the trace (e.g., burstiness, seasonality, patterns, and trends) and can be used at any time to automatically generate comparable traces, that is, the trace exhibits the same properties. Furthermore, the extracted model instance can be easily modified to reflect a target dynamic load scenario, for example, by changing the frequency of bursts or adding a given trend behavior.

In this article, we introduce and validate three automated DLIM model extraction methods: s-DLIM, p-DLIM, and hl-DLIM. s-DLIM's workflow is inspired by the time series decomposition approach STL [Cleveland et al. 1990]. p-DLIM focuses more on periodic patterns strengthening extrapolation capabilities, and the hl-DLIM extraction method works on a higher abstraction level for more compact model instances.

We highlight as major benefits of this work the new capabilities to accurately and automatically extract load intensity models with 15.2% median modeling error on average from a representative set of 10 different real-world traces. Each extraction completes in less than 0.2s on common consumer hardware. These results demonstrate and validate the capability of DLIM to capture realistic load intensity profiles.

DLIM-based applications and developments in the fields of benchmarking and system resource planning both at design time and runtime are enabled by providing the automatic model extraction processes. We demonstrate the usefulness and applicability of DLIM and its extraction processes by outlining selected existing use cases (cf. Section 6).

The remainder of this article is structured as follows: Section 2 introduces foundations on open workloads, our definition of load intensity, and discusses related work in the field of workload modeling approaches. Section 3 describes the DLIM model and the hl-DLIM model. The extraction methods are presented in detail in Section 4. Section 5 validates the models DLIM and hl-DLIM, and the extraction methods s-DLIM, p-DLIM, and hl-DLIM based on real-world traces. Section 6 uses applications of workload modeling in systems engineering to show the adoption DLIM. Section 7 concludes and provides an outlook on future research directions.

2. FOUNDATIONS AND RELATED WORK

Both DLIM and hl-DLIM have been designed to capture variations of load intensity in the form of user, job, or request arrival rates. The models employ an **open workload** view. Open workloads are defined in Schroeder et al. [2006] as workloads in which new jobs arrive independently of job completion. We use the term *load* to denote user, job, or request arrival rates containing the actual *work* units. Open workloads are typical for cloud environments as users are usually unaware of other users or the current workload.

In this article, **load intensity** denotes the arrival rate of abstract workload units (e.g., users, jobs, sessions, or requests) at a given point in time. The function $r(t)$ describes the load intensity at that point in time (t) as follows:

$$r(t) = R'(t) \quad with \quad R(t) = |\{u_{t_0} | t_0 \le t\}|, \tag{1}$$

[1]LIMBO (DLIM tools): Primary page http://descartes.tools/limbo, also hosted at SPEC RG tools repository https://research.spec.org/tools/overview/limbo.html.

where $R(t)$ is the amount of work units u_{t_0} (set cardinality of the set containing all u_{t_0}) with their respective arrival time t_0, which precedes or equals our time of observation t. This means that all work units u_{t_0} have arrived up until time t. $r(t) = R'(t)$ is the derivative of $R(t)$.

A **load intensity profile** is the function that describes the load intensity over time. Real-world examples of such profiles are shown in several figures throughout this article, including Figure 6.

Several approaches to describe and generate workloads with variable intensity exist in literature. However, they differ from our approach in the following key aspects: First, a set of approaches that work purely statistically using independent random variables and therefore do not offer models describing deterministic load intensity changes over time. Second, approaches for workload or user behavior modeling that capture the structure of the actual units of work they dispatch or emphasize the behavior of users after their arrival on the system. In contrast, DLIM models focus on the description of request or user arrivals, and not on user behavior and its impact after arrival. We combine both deterministic and statistical approaches. We group related work into at least one of the following four categories.

User Behavior Models. These models traverse graphs, where nodes represent actions that users may perform on a system. By doing so, user behavior models allow modeling of varying system load depending on the current state of the traversal. Becker et al. [2009], van Hoorn et al. [2008], Roy et al. [2013], and Beitch et al. [2010] propose workload models that capture the behavior and tasks triggered by different types of users. Zakay and Feitelson [2013] partition workloads according to the user types, and then sample workload traces for each user type to capture the user behavior. All four approaches focus on the behavior of users instead of the load intensity, which we define at the system boundary. Models like the previous ones can be easily combined with DLIM to further characterize the user behavior after a request has arrived at the system and a client session is started.

Resource Demand Focused Load Modeling. Approaches in this category focus on modeling the units of work processed by the system and estimating the resource demands created by those work units. Casale et al. [2012] focus on modeling bursty workloads, whereas Barford and Crovella [1998] focus on file distribution and popularity.

Statistical Interarrival Models. These approaches capture the workload intensity using statistical distributions. Feitelson [2002] creates a statistical model for parallel job schedulers. Li [2010] models batch workloads for eScience grids and Menascé et al. [2003] as well as Reyes-Lecuona et al. [1999] analyze workloads at multiple levels, such as request and session level. These approaches differ from our approach as they use independent random variables to capture interarrival rate distributions. Model input is usually the type of workload or certain workload characteristics, which may include arrival rate, but also other characteristics, such as burst size, whereas the output is a statistical distribution. In our case, we use the current time as the primary input (usually seconds since a predefined point in time t_0, for example, measurement start), with the model returning the concrete load intensity at that point in time.

Regression Techniques. MARS [Friedman 1991], M5 trees [Quinlan et al. 1992], or cubic forests [Kuhn et al. 2012] are capable of calibrating mathematical functions to fit a measured load intensity trace. Similarly to DLIM, these functions can be load intensity over time functions. Yet, in contrast to DLIM, they do not convey the additional information of the types and composition of load intensity variation components.

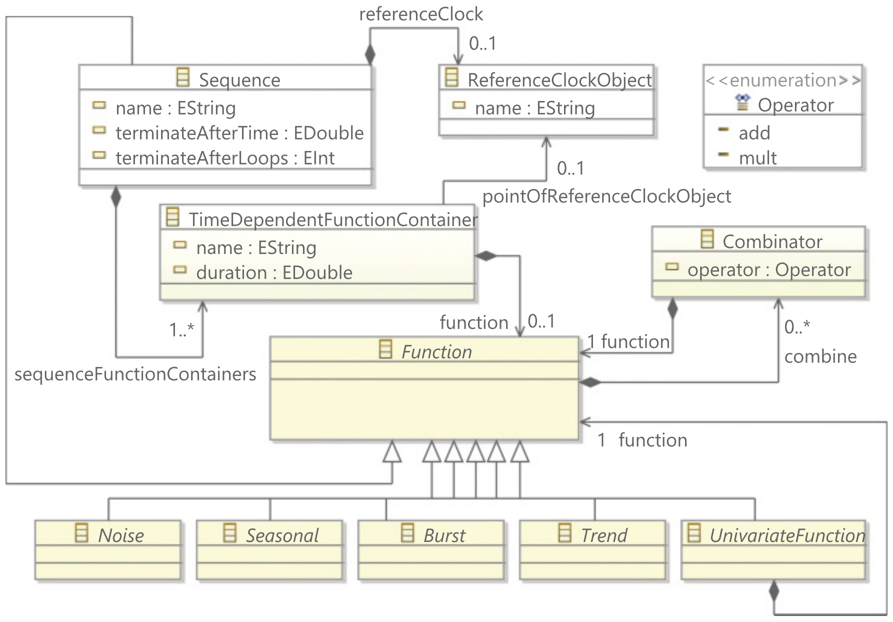

Fig. 1. Outline of the DLIM meat-model excluding a detailed description of concrete functions.

The combined deterministic and statistical approach of DLIM enables a mapping between load profile variations and their respective time stamps. DLIM uses a composition of piecewise mathematical functions. In contrast to statistical approaches, which model distributions, DLIM models a concrete load intensity for each distinct point in time. DLIM also supports random noise including spikes of various shapes and therefore represent noise, random events, and individual outliers as well. To the best of our knowledge, a directly comparable combined deterministic and statistical modeling approach for load intensity profiles is not published in literature.

3. THE DESCARTES LOAD INTENSITY MODELS DLIM AND HL-DLIM

The DLIM describes load intensity over time. Specifically, the model is aimed at describing the variations of work unit arrival rates by capturing characteristic load intensity behaviors. Its metamodel is visualized in Figure 1. It is based on the work in von Kistowski et al. [2014], which introduced a more abstract preliminary sketch of the model, missing many of the concrete functions used for load intensity description.

DLIM models load intensity by defining piecewise mathematical functions to approximate variable arrival rates. It supports load intensity profiles with periodicity and offers flexibility to adapt and incorporate unplanned events. It also allows for nested composition of model instances. This nesting results in DLIM instances always having a tree structure, where different nodes and leaves within the tree are added or multiplied onto their parents.

DLIM uses the *Sequence* as its central element for the composition of piecewise mathematical functions. A *Sequence* carries an ordered set of *TimeDependentFunctionContainers*, which describe the duration of each time interval. The containers, in turn, contain the actual mathematical functions describing the load intensity during their interval. The *Sequence*'s ordered set of *TimeDependentFunctionContainers* repeats as many times as indicated by the *terminateAfterLoops* attribute. Alternatively, the sequence repeats for the time indicated by the *terminateAfterTime* attribute. The

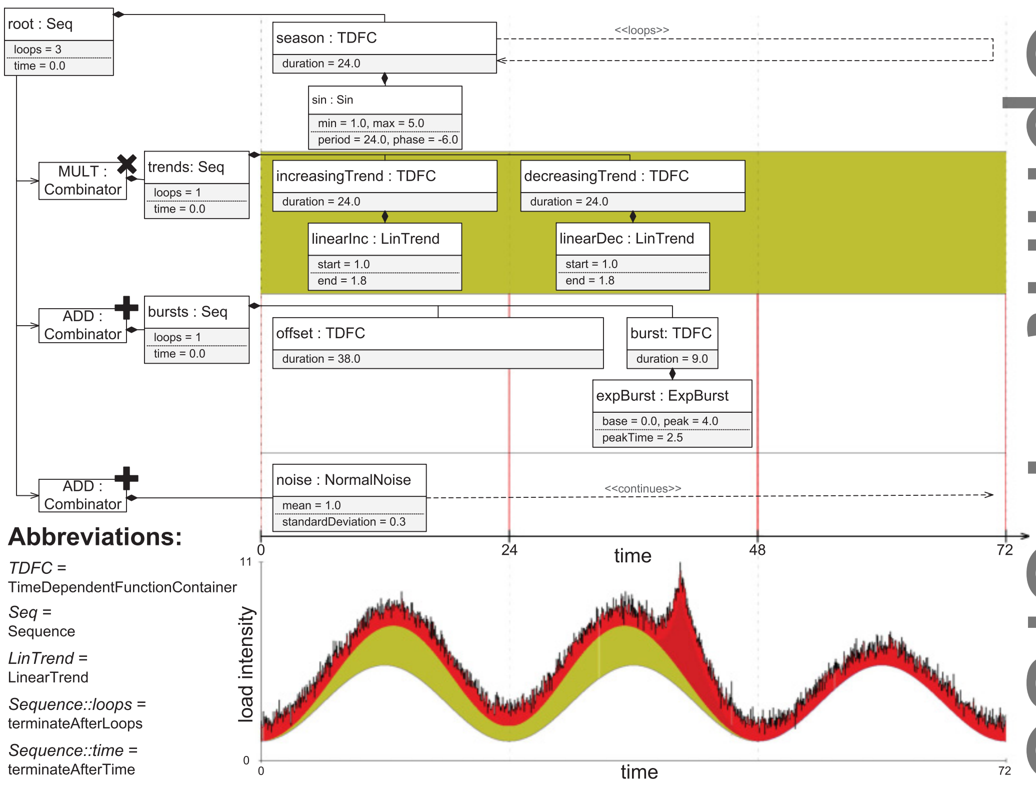

Fig. 2. Example instance of a DLIM model.

sequence terminates as soon as at least one of the termination criteria is met. Infinite sequences are not allowed and at least one termination criterion must be specified. This ensures a finite time frame and guarantees that benchmarks or predictions terminate.

Any DLIM Function may use mathematical operators called *Combinator*s to contain additional Functions. A *Combinator* allows the multiplication or addition of a *Function*'s load intensity with the load intensity as defined by another *Function*. In the context of the overall load variation over time, any *Function* contained within a *Combinator* is valid for the exact same duration as its containing parent *Function*. This containment results in trees of functions containing zero or more additional functions. All of these functions describe the load intensity behavior during a specific time period defined by the containing *TimeDependentFunctionContainer*.

The *TimeDependentFunctionContainer* describes its load intensity for a set *duration*, after which the next *TimeDependentFunctionContainer* in the parent *Sequence's* ordered set becomes valid.

Function is the abstract parent class to all classes denoting mathematical functions. It is contained within a *TimeDependentFunctionContainer*. A number of concrete children are provided that can be used as *Functions*. The default set of functions provides *Noise*, *Seasonal*, *Burst*, *Trend*, and *UnivariateFunction* specifications. New functions can be provided by inheritance. The most notable concrete *Function* is the *Sequence*. As a result, any function can contain a new *Sequence* using a mathematical operator. The containment hierarchy prevents cycles.

Figure 2 shows an example DLIM instance together with a plot of the model example in the lower part. In both the model and the plot, the red areas map to the impact of the

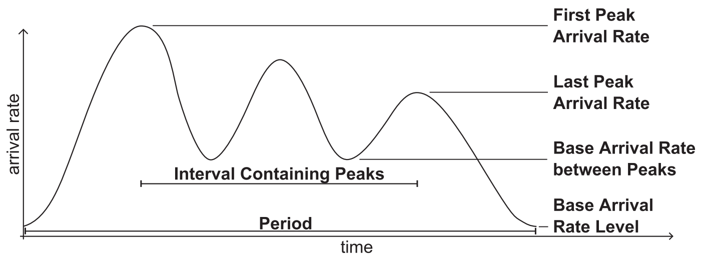

Fig. 3. hl-DLIM *Seasonal* Part.

additive combinators and the yellow areas to the impact of multiplicative combinators. The root *Sequence* (named "root") of the instance contains a *TimeDependentFunction-Container (TDFC)* "season" with a duration of 24 abstract time units. "root" repeats its single *TDFC* three times before terminating. The container itself contains a sinus function modeling the load intensity. This sinus function repeats itself. As a result, the seasonal duration of 24 is technically speaking unnecessary, but was chosen nonetheless to add additional information to the model. "root" also contains *Combinators*, which modify its load intensity. The multiplicative *Combinator* contains a separate *Sequence* ("trends"), which contains two *TDFCs*. The first of those models a linear increase in load intensity, up to a factor of 1.8, whereas the latter models a decrease back to the original level. The "trends" *Sequence* terminates before "root" and stops modifying it after its first iteration. A "burst" *Sequence* and normal noise are added to "root." Note, that *Combinators* may contain any DLIM-*Function*. This is usually a *Sequence*, as is the case for trends and bursts in this example model. Contained *Sequences* terminate on their own or at the end of the containing *Sequence* if their duration would exceed the parent's duration. All other functions, such as the noise in this example, are valid for the entire duration of their containing *Sequence*.

3.1. High-Level DLIM

DLIM offers a convenient way of structuring and ordering functions for the description of load intensity profiles. Its tree of piecewise mathematical functions already provides human users with a better understanding of the load variations. However, abstract knowledge about variations contained in complex models can still be difficult to understand, as a large tree of composite functions may be difficult to grasp. The (hl-)DLIM is a separate model that addresses this issue by providing a means to capture load intensity variations described only by a limited number of nonhierarchical parameters instead of a potentially deeply nested tree of mathematical functions. These parameters can then be used to quickly define and characterize a load intensity model.

hl-DLIM separates into a *Seasonal* and *Trend* part (inspired by the time-series decomposition approach in BFAST [Verbesselt et al. 2010]) and features a *Burst* and a *Noise* part. In contrast to DLIM, it is designed to model a subset of the most common load variation profiles in favor of better readability.

The *Seasonal* part describes the function that repeats after every seasonal duration (e.g., every day in a month-long load intensity description). hl-DLIM describes the seasonal part using the following parameters (as shown in Figure 3): *period*, *number of peaks*, *base arrival rate level*, *first peak arrival rate*, *last peak arrival rate*, and the *interval containing peaks*. Arrival rates of additional peaks between the first and last peak are derived using linear interpolation. Linear interpolation is chosen because it is the most intuitive for the modeling user. More detailed changes can be performed in (non-hl-)DLIM.

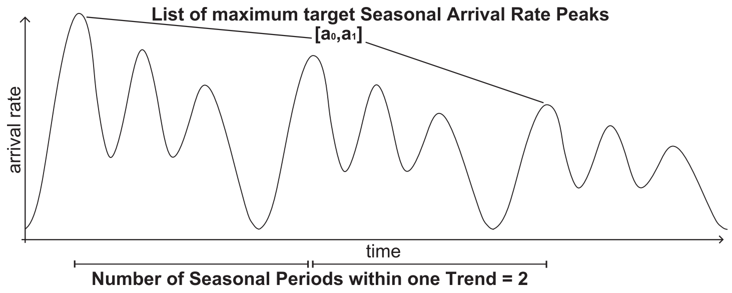
List of maximum target Seasonal Arrival Rate Peaks
[a0,a1]
arrival rate
time
Number of Seasonal Periods within one Trend = 2

Fig. 4. hl-DLIM *Trend* Part.

The *Trend* part describes an overarching function that captures the overall change in the load intensity over multiple seasonal periods. It consists of a list of equilength *Trend* segments. hl-DLIM describes the respective arrival rates at the start of each of these segments. The *Trend* must modify the arrival rate of the *Seasonal* part's maximum peak in such a way that it matches this specified target arrival rate. The actual trend segments between these specified points can be interpolated using any DLIM *Trend* function. In contrast to the *Trend* within BFAST, the hl-DLIM *Trend* can interact with the *Seasonal* part either by addition or multiplication. The restrictions posed by hl-DLIM to *Trend* modeling are intended to speed the human modeling process. (Non-hl-)DLIM allows for more varied *Trend* specifications. The *Trend* part is defined using the following parameters: *number of seasonal periods within one trend* (i.e., the length of a single trend segment), *operator* (addition or multiplication), and the *list of seasonal arrival rate peaks*. The latter defines the arrival rate at the beginning and end of the *Trend* segments. Since the list defines the maximum seasonal peak for seasonal iterations, trend segments can be interpreted as overlying interpolated functions, beginning and ending at the maximum peaks of their respective seasonal iterations. This interpretation is visualized in Figure 4.

The *Burst* part allows the definition of recurring bursts, which are added onto the existing *Seasonal* and *Trend* behavior (in contrast to DLIM, where bursts may also be multiplicative). It is defined using the following parameters: *First burst offset*, *inter-burst period*, *burst peak arrival rate*, and *burst width*.

The *Noise* part allows the addition of uniformly distributed white noise. The distribution is defined by its upper and lower bounds, which are named *Minimum Noise Rate* and *Maximum Noise Rate*. Uniform noise is used in hl-DLIM, due to its ease and intuitiveness when being specified by a human user. Other Noise distributions can easily be added to DLIM instances, which are obtained from hl-DLIM instances via a model-to-model transformation.

4. MODEL INSTANCE EXTRACTION

In this section, we present three methods for the extraction of DLIM instances from arrival rate traces, consisting of pairs of arrival rates at their respective time stamps. Each model extraction method requires few configuration parameters that we discuss in detail. Given the set of configuration parameters, the extraction runs completely automated.

We define the following three methods:

(1) **Simple DLIM Extraction Method** (s-DLIM):
Extracts a DLIM instance. This process (and its resulting DLIM instance) are inspired by the time-series decomposition approach STL [Cleveland et al. 1990]. s-DLIM extracts a repeating *Seasonal Part* and a nonrepeating *Trend Part*. The

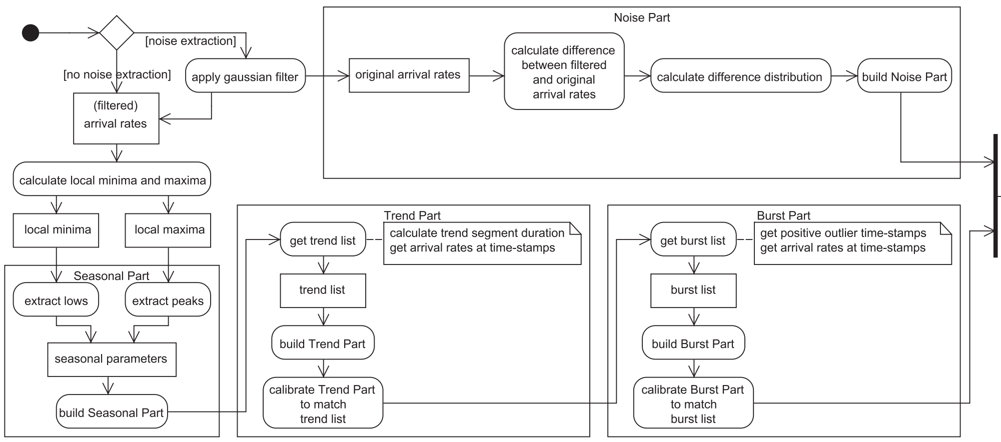
[noise extraction]
[no noise extraction]
apply gaussian filter
(filtered) arrival rates
calculate local minima and maxima
local minima
local maxima
Seasonal Part
extract lows
extract peaks
seasonal parameters
build Seasonal Part
Noise Part
original arrival rates
calculate difference between filtered and original arrival rates
calculate difference distribution
build Noise Part
Trend Part
get trend list
calculate trend segment duration get arrival rates at time-stamps
trend list
build Trend Part
calibrate Trend Part to match trend list
Burst Part
get burst list
get positive outlier time-stamps get arrival rates at time-stamps
burst list
build Burst Part
calibrate Burst Part to match burst list

Fig. 5. Activity diagram of the *Simple DLIM Extraction Method* (s-DLIM).

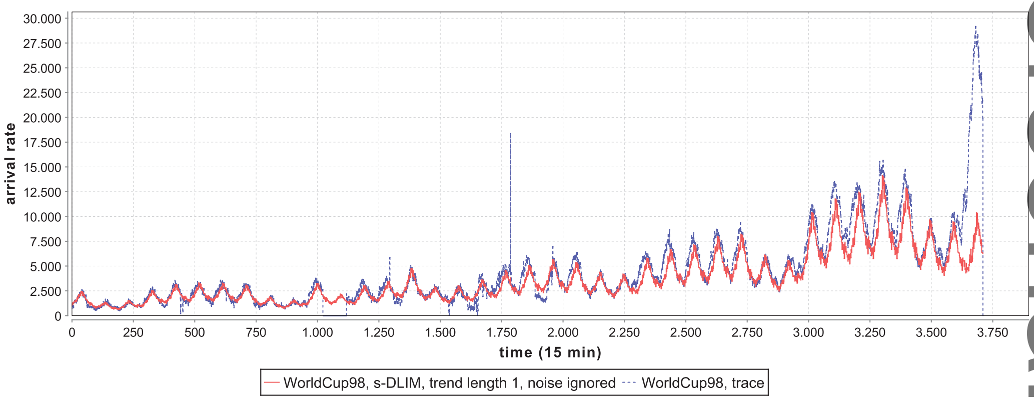

Fig. 6. Arrival rates of the original WorldCup98 trace (blue) and the extracted DLIM instance (red) using s-DLIM with a *Trend* length of 1 and ignoring noise.

nonrepeating *Trend Part* contains a list of *Trend* segments of fixed size, that interpolate between their start and end arrival rate value. The *Trend* list extends throughout the entire time duration for which the extracted model is defined. Additionally, a *Burst Part* and an optional *Noise Part* are extracted. s-DLIM is visualized in Figure 5.

(2) **Periodic DLIM Extraction Method** (p-DLIM):
This is a variation of the simple extraction process that features multiple repeating trends. Again, a DLIM instance is extracted; however, in contrast to s-DLIM, p-DLIM does not feature a single list of equal length *Trend* segments. Instead it features multiple lists of *Trends*, each containing a fixed number of *Trend* segments of (potentially) different lengths.

(3) **High-level DLIM Extraction Method** (hl-DLIM):
Extracts an hl-DLIM instance. This process is based on the simple model extraction process and uses the information extracted by the latter to derive the parameters needed to construct an hl-DLIM instance.

ALGORITHM 1: Extracting the *Seasonal* Part

```
Data: duration: seasonal period duration,
LIST: list of tuples $\vec{t} = \begin{pmatrix} arrivalRate \\ timeStamp \end{pmatrix}$,
rootSequence: root Sequence of the DLIM instance;
Function extractSeasonalPart()
    MIN ← getLocalMinima (LIST);
    MAX ← getLocalMaxima (LIST);
    peakNum ← median(number of peaks within each Seasonal iteration);
    for $i$ ← 0 to peakNum − 1 do
        peak [i].arrivalRate ← median(arrival rate of all $i_{th}$ peaks ∈ MAX within each seasonal
          iteration);
        peak [i].timeStamp ← median(time stamp of all $i_{th}$ peaks ∈ MAX within each seasonal
          iteration);
        /* In seasonal iterations with more than peakNum peaks, the $i_{th}$ peak is
           selected, so that peaks are evenly spaced throughout that seasonal
           iteration.                                                          */
        peak [i] ← $\begin{pmatrix} \text{peak}[i].arrivalRate \\ \text{peak}[i].timeStamp \end{pmatrix}$;
    end
    for $i$ ← 0 to peakNum − 1 do
        low [i].arrivalRate ← median(arrival rate of all $i_{th}$ lows ∈ MIN within each seasonal
          iteration);
        low [i].timeStamp ← median(time stamp of all $i_{th}$ lows ∈ MIN within each seasonal
          iteration);
        /* In seasonal iterations with more than peakNum lows, the $i_{th}$ low is
           selected, so that lows are evenly spaced throughout that seasonal
           iteration.                                                          */
        low [i] ← $\begin{pmatrix} \text{low}[i].arrivalRate \\ \text{low}[i].timeStamp \end{pmatrix}$;
    end
    for $i$ ← 0 to peakNum − 1 do
        interpolatingFunction ← DLIM Function starting at low [i], ending at peak [i];
        rootSequence.append(interpolatingFunction);
    end
end
```

4.1. Extracting a s-DLIM and p-DLIM Instance

The following sections describe the extraction of the different model parts by s-DLIM and p-DLIM. These two processes only differ in their approach to the extraction of the *Trend Part*.

4.1.1. Extracting the Seasonal Part. The *Seasonal Part* of the arrival rate trace is modeled using a *Sequence* of *TimeDependentFunctionContainers* and their *Functions*. Each *Function* interpolates the corresponding peaks and lows within each seasonal period. As a result, the following data needs to be derived in order to build the *Seasonal Part*:

—Duration of the dominating seasonal period: automatically derived or configured.
—Arrival rate peaks and their time stamps: automatically derived.
—Arrival rate lows and their time stamps: automatically derived.
—Function type used to interpolate between peaks and lows: (pre-)configured.

The duration of the seasonal period can be set by the user in cases when the sampling interval of the input time series is known. A trace that extends for multiple days and contains daily patterns, for example, features a period of 24 hours. In cases, where the seasonal period is not transparent, we provide an implementation of a heuristic to estimate the seasonal period duration. We follow a heuristic described in Wang et al. [2009, p. 17] and compute the autocorrelation function for all lags up to one-third of the time series length. As preprocessing, an overarching trend is removed from the input data using a regression spline. In the following step, the local maximum with the smallest lag is searched with a sliding window. Finally, we cross-check if multiples of the first local minimum lag correspond to other local minima. The peaks and lows are automatically determined by using a local minimum/maximum search on the arrival rates within the trace. The local arrival rate minima and maxima and their corresponding time stamps within a seasonal period constitute the peaks and lows. Since the trace usually contains multiple seasonal periods, the respective median arrival rate value is selected for each local maximum and minimum within the *Seasonal Part*. Selecting the median instead of the mean reduces the impact of outliers on the extracted seasonal values. Outliers can be significant, but more importantly they are usually positive outliers. Negative outliers (disconnects, maintenance breaks) are less common and do not have as much of an impact, as their minimum size is 0. Positive outliers (bursts) are more common and intended to be detected separately using our burst detection. As a result, the derived functions interpolate first between the first median low and the first median peak, then between the first median peak and the second median low, and so on. The last low must be of the same arrival rate as the first low in order for the *Seasonal Part* to repeat seamlessly. The type of the interpolating function (linear, exponential, logarithmic, sin-flank) can be selected by the user, depending on his needs. According to our experience, the sin interpolation usually results in a good model fit. The *Seasonal Part* extraction is illustrated in Algorithm 1.

4.1.2. Extracting the Trend Part. The *Trend Part* consists of a series of functions (trend segments) that are either added or multiplied onto the *Seasonal Part*. Each trend segment begins at the maximum peak of the *Seasonal Part* and ends at the maximum peak of the *Seasonal Part* in a later *Seasonal* iteration. This minimizes errors with trend calibration. The trend extraction calibrates the trend in a way that the model output arrival rate at the trend segment's beginning (or end) equals the trace's actual arrival rate at the respective point in time. The shape of the trend function (linear, exponential, logarithmic, sin) is predefined as a sin-shape, but can be changed on demand.

Trend Part for s-DLIM. The simple extraction process features a list of equal-length trend segments. These segments have a user defined duration that is a multiple of the seasonal period. Like the seasonal period it is also selected using metaknowledge about the trace. These segments are then calibrated at their beginning and end to match the arrival rates in the trace. The s-DLIM *Trend Part* extraction is displayed in Algorithm 2.

Trend Part for p-DLIM.The periodic extraction process takes into account, that multiple repeating trends may be part of the arrival rate trace. Examples are weekly and monthly trends. Since repeating trends (like the *Seasonal Part*'s dummy function) should end on the same arrival rate as the arrival rate they started on (allowing seamless repetition), each of these repeating trends contains at least two trend segments. These trend segments' duration is a multiple of the seasonal period. Unlike the s-DLIM trend segments they are not required to be of equal length, thus allowing odd multiples

ALGORITHM 2: Extracting the *Trend* Part using s-DLIM

```
Data: duration: seasonal period duration,
LIST: list of tuples $\vec{t} = \binom{arrivalRate}{timeStamp}$,
MAX: list of local maxima in LIST,
trendSequence: root Sequence of all Trend segments;
Function extractTrendPart()
    largestPeakOffset $\leftarrow$ offset of peak with largest arrival rate within a seasonal iteration;
    largestPeakArrivalRate $\leftarrow$ arrival rate of peak with largest arrival rate within a seasonal
      iteration;
    iterations $\leftarrow$ LIST.lastTuple.timeStamp/duration;
    for $i \leftarrow 0$ to iterations do
        a $\leftarrow$ nearestTuple(MAX, $i *$ duration + largestPeakOffset);
        trendPoint [i] = a/largestPeakArrivalRate;
    end
    trendSequence.append(constant trendPoint [0] with duration largestPeakOffset);
    for $i \leftarrow 0$ to iterations do
        interpolatingFunction $\leftarrow$ DLIM Function starting at trendPoint [i], ending at trendPoint
          [i+1];
        trendSequence.append(interpolatingFunction with duration duration);
    end
    trendSequence.append(constant trendPoint [iterations] with duration
      (duration $-$ largestPeakOffset));
end
Function nearestTuple(tuple list L, time)
    returns the tuple $\vec{t} = \binom{arrivalRate}{timeStamp} \in \mathbf{L}$ with minimal $d \leftarrow |\mathbf{L}.timeStamp - time|$;
end
```

of seasonal periods as total trend durations. The user selects lists of at least two trend segment durations for each repeating *Trend Part*.

4.1.3. Extracting the Burst Part. Extracting *bursts* is a matter of finding the points in time at which significant outliers from the previously extracted *Seasonal* and *Trend* parts are observed in the trace. However, for our extraction, bursts are explicitly not missed seasonal peaks. Extracted *bursts* are not intended to model the model's remaining noise, only semantic bursts. In the context of load intensity modeling, we define bursts as a significant amount of additional load that exceeds the seasonal pattern and is too short to be modeled as *trend*. To this end, we require a filter function that smooths the seasonal function to prevent modeling of missed seasonal peaks using bursts. With this filter function bursts can be detected if

$$r(t) > (f(seasonal(t)) * trend(t)) * c,$$

where $r(t)$ is the original load intensity (arrival rate), f is the filter function, *seasonal* and *trend* are the previously extracted *Seasonal* and (multiplicative) *Trend* parts, and c is a constant factor that requires bursts to be a certain factor larger than filtered season and trend. c is set to a default value of 1.2.

Once a burst is found, it is added to the root *Sequence* and then calibrated to match the arrival rate from the trace. The filtered *Seasonal Part* used as reference model in the burst recognition activity differs from the actual extracted *Seasonal Part*. It is filtered so that the *Seasonal Part* used for burst recognition activity does not interpolate between the peaks and lows of the original arrival rate trace. Instead, it interpolates only between the peaks. This removes false positives due to seasonal periods that are slightly offset in time; however, it also eliminates bursts that do not exceed the current seasonal peak. This trade-off is considered acceptable, since timewise offset seasonal periods are commonly observed.

4.1.4. Extracting the Noise Part. The *Noise Part* extraction consists of two steps: noise reduction and the calculation of the noise distribution. The idea behind our approach is to first reduce the noise observed within the arrival rates contained in the trace, and then reconstruct the reduced noise by calculating the difference between the original trace and the filtered one. Having filtered the noise, the extraction of the *Seasonal Part*, *Trend Part*, and *Burst Part* are then performed on the filtered trace. This has a significant impact on the extraction accuracy of these parts, and thus on the overall accuracy of the extracted model instance, especially when extracting hl-DLIM instances, as will be shown later in the model accuracy evaluation (Section 5). Depending on the trace, the overall accuracy of the DLIM extraction can be improved by noise elimination. In this case, we recommend applying noise extraction, even if the extracted noise component itself is deleted later on.

Noise Reduction. Noise is reduced via the application of a one-dimensional Gaussian filter on the read arrival rates. A Gaussian filter has a kernel based on the Gaussian distribution; it thus has the following form (as defined in Blinchikoff and Zverev [1986]):

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}.$$

We choose the kernel width depending on the *Seasonal* period (duration of a single seasonal iteration) and the expected number of peaks (local maxima) within a *Seasonal* period:

$$KernelWidth = \frac{SeasonalPeriod}{ExpectedMax\#SeasonalPeaks}.$$

A Gaussian filter's kernel width is defined as

$$KernelWidth = 6 \cdot \sigma - 1.$$

As a result, the standard deviation is

$$\sigma = \frac{\frac{SeasonalPeriod}{ExpectedMax\#SeasonalPeaks} + 1}{6}.$$

Calculating the Noise Distribution. The *Noise Part* is modeled as a normally distributed random variable. This variable is added to the DLIM instance's root *Sequence*. The normal distribution's mean and standard deviation are calculated as the mean and standard deviation of the differences between the filtered arrival rate trace. This is illustrated in Algorithm 3.

s-DLIM and p-DLIM both only support the extraction of normally distributed noise. Other noise distributions are not supported. hl-DLIM extraction, however, supports the extraction of uniformly distributed noise.

4.2. Extracting an hl-DLIM Instance

The *hl-DLIM extraction* is similar to s-DLIM extraction. This section only highlights the differences between those two processes.

ALGORITHM 3: Calculating the *Noise* Distribution

```
Data: LIST: list of read tuples $\vec{t} = \begin{pmatrix} arrivalRate \\ timeStamp \end{pmatrix}$;
Function calculatNoiseDistribution()
    FILTERED_LIST ← applyGaussianFilter(LIST);
    for $i \leftarrow 0$ to $|LIST| - 1$ do
        difference[i] ← LIST [i].arrivalRate - FILTERED_LIST [i].arrivalRate;
    end
    distribution ← normal distribution with mean(difference) and
     standardDistribution(difference);
end
```

4.2.1. Seasonal Part. hl-DLIM is restricted to only support peaks with an equal distance from one another. The arrival rates of such peaks are linearly interpolated between the first peak's arrival rate and the last peak's arrival rate. When extracting an hl-DLIM instance from an arrival rate trace, the difference thus lies in the *interval containing peaks* and in the search for the maximum and minimum peak. The *interval containing peaks* is calculated as the time difference between the first and the last peak; the *first peak*'s arrival rate is then set either to the minimum or maximum peak (depending on whether the first median peak has a greater or a smaller arrival rate than the last median peak in the trace) and the *last peak* is set to the corresponding counterpart.

4.2.2. Trend Part. Extracting the *Trend Part* is done almost identically as in the simple model extraction process, since hl-DLIM defines its *Trend Part* as a list of arrival rates at the beginning and end of each trend segment, identically to the arrival rate list extracted in s-DLIM. The only difference is the offset before the first trend segment begins. The trend segment always ranges from the maximum peak within one seasonal period to the maximum peak within a following seasonal period. The simple model extraction process allows this maximum peak to be any seasonal peak. hl-DLIM, however, only allows the first or last peak to be the maximum peak. As a result, the time offset for the first trend segment is slightly different.

4.2.3. Burst Part. Bursts are detected and calibrated using the same peak-only *Seasonal Part* as in *s-DLIM*. While the other model extraction processes modeled each burst individually, hl-DLIM only supports recurring bursts. Thus, only the *first burst offset* and the *interburst period* are extracted, as well as only a single *burst arrival rate*. The first burst offset is selected based on its time stamp, whereas the period between recurring bursts is calculated as the median *interburst period* from the independent bursts. The burst arrival rate is also calculated as the median burst arrival rate.

4.2.4. Noise Part. In hl-DLIM, noise is extracted using our previously described filtering approach, thus having the same noise reduction side effects as in the other model extraction processes. hl-DLIM, however, only supports a uniformly distributed random variable as noise. In order to eliminate outliers but keep the intuitiveness of hl-DLIM, hl-DLIM extraction only approximates the noise distribution of the original trace, eliminating major outliers. For this approximation the minimum and maximum values of the respective uniform distribution are selected as the 10^{th} and 90^{th} percentile of the difference between the filtered and unfiltered arrival rates.

5. MODEL ACCURACY EVALUATION

In this section, we evaluate the automated model extraction methods in terms of their achieved accuracy. By showing that 10 different real-world server traces covering periods between two weeks and seven months can be captured in the descriptive DLIM model format with reasonable accuracy with our automated extraction processes, we demonstrate the expressiveness of our proposed modeling formalism. All traces exhibit human usage patterns. The extraction methods are applied to these traces in order to extract DLIM instances and compare them to the corresponding original traces by computing the Median Absolute Percentage Errors (MdAPEs) [Hyndman and Koehler 2006]. The MdAPE is calculated by computing the median over the absolute values of the relative deviations of the extracted model for each arrival rate in the original trace. The Mean Absolute Percentage Error (MAPE) is not chosen as this measure is prone to deflection by positive outliers that are more likely to occur as negative outliers as also discussed earlier.

s-DLIM and hl-DLIM extraction are applied to extract model instances for all traces. For these extraction methods, we separately evaluate the effect of noise extraction, including noise reduction. The shape of the interpolating functions is always selected as the DLIM *SinTrend*, meaning that sin-flanks are always used for the interpolation between arrival rate peaks and lows. We chose *SinTrend* because it fits closest to the original trace in the majority of cases. For the same reason, *Exponential Increase And Decline* is always selected for *Burst* modeling (it is a child of *Burst* in the DLIM metamodel). *Trends* are selected to be multiplicative since this way they have a lower impact on arrival rate lows and a relatively high impact on arrival rate peaks (contrary to additive *Trends*, which have a constant impact on both). We do this, since arrival rate lows vary less than arrival rate peaks according to our observations.

s-DLIM is also configured with varying *Trend* lengths. Best results are expected at *Trend* length of one *Seasonal* period, whereas lower accuracy is expected at the longest evaluated *Trend* length of three *Seasonal* periods. For traces with a duration greater than 1 month, we also apply p-DLIM. p-DLIM is configured to extract weeks as a periodic *Trend* list with two *Trend* segments of the length of three and four. Additionally, it extracts a biweekly period with a *Trend* list using two *Trend* segments of the length of 7. Finally, it extracts a monthly (4-week) period with a *Trend* list using two *Trend* segments of the length of 14.

We compare the extraction error and runtime on commodity hardware (Core i7 4770, 16 GB RAM) against the STL [Cleveland et al. 1990] and BFAST time-series decomposition [Verbesselt et al. 2010] (which both return split data as opposed to a descriptive model). To enable a fair comparison, we configure STL and BFAST to extract one seasonal pattern and not more than one trend per day to match with the features of the DLIM extractors and to not further slow down BFAST. This configuration had no significant impact on the accuracy, but decreases processing speed. In contrast to DLIM, where seasonal patterns are represented by piece-wise interpolating functions, in STL and BFAST outputs, the seasonal pattern is represented as a less compact discrete function.

5.1. Internet Traffic Archive and BibSonomy Traces

The first batch of traces was retrieved from The Internet Traffic Archive/[2] The Internet Traffic Archive includes the following traces: *ClarkNet-HTTP* (Internet provider WWW server), *NASA-HTTP* (Kennedy Space Center WWW server), *Saskatchewan-HTTP (Sask.)* (University WWW server), and *WorldCup98 (WC98)* (official World Cup 98 WWW servers). Additionally, we used a 6-week-long trace of access times to the social bookmarking system *BibSonomy* [Benz et al. 2010], beginning on May 1st 2011.[3]

[2]Internet Traffic Archive: `http://ita.ee.lbl.gov/`.
[3]The request log dataset is obtainable on request for research purposes: `http://www.kde.cs.uni-kassel.de/bibsonomy/dumps/`.

Table I. Model Extraction Errors for the Internet Traffic Archive and BibSonomy Traces

Trace	1. **ClarkNet**	2. **NASA**	3. **Sask.**	4. **WC98**	5. **BibSonomy**
Extraction Parameters	MdAPE	MdAPE	MdAPE	MdAPE	MdAPE
Extractor, Trend, Noise	[%]	[%]	[%]	[%]	[%]
p-DLIM, -, extracted	too short	32.223	43.293	52.304	37.387
p-DLIM, -, eliminated	too short	28.944	35.831	53.316	35.378
p-DLIM, -, ignored	too short	23.633	35.663	53.495	36.264
s-DLIM, 1, extracted	21.195	26.446	35.551	19.735	26.988
s-DLIM, 1, eliminated	17.509	23.560	**26.492**	16.882	**21.470**
s-DLIM, 1, ignored	**12.409**	**18.812**	29.171	**12.979**	23.831
s-DLIM, 2, ignored	14.734	20.800	30.273	15.691	26.786
s-DLIM, 3, ignored	14.919	27.577	32.085	19.161	28.218
hl-DLIM, 1, extracted	20.105	26.541	37.942	16.093	27.513
hl-DLIM, 1, eliminated	19.361	24.539	33.240	15.660	25.433
hl-DLIM, 1, ignored	72.924	55.575	80.792	43.957	42.268
STL	13.540	20.384	30.134	16.041	20.299
BFAST	12.243	no result	no result	no result	no result
avg. s-DLIM runtime	4.2ms	25.2ms	118.8ms	11.8ms	125ms
avg. STL runtime	3.5ms	15.0ms	38.7ms	13.2ms	15.0ms
avg. BFAST runtime	76,276ms	no result	no result	no result	no result

All traces were parsed to arrival rate traces with a quarter-hourly resolution (96 arrival rate samples per day).

Table I shows the MdAPE for s-DLIM, p-DLIM, and the hl-DLIM extraction for different configurations. It also displays runtime of the overall most accurate extraction configuration (s-DLIM, ignoring noise, trend length 1) as an average value over 10 runs. Accuracy and average runtimes are also displayed for BFAST and STL. For some cases, BFAST did not terminate after more than one 1.5h.

The ClarkNet and NASA extraction results show that s-DLIM provides the best accuracy, especially with a *Trend* length of 1. Noise reduction does not seem to help for this particular trace during the DLIM extraction. The result does not improve when extracting the noise, as noise generated by a random variable does not reproduce the exact measured results and increases the absolute arrival rate difference between trace and model. We trace the discrepancies between the extracted model instance and the original trace to three major causes:

—In some cases, bursts are not detected with full accuracy.
—The NASA server was shut down for maintenance between time stamps 2,700 and 2,900. The extraction methods do not have contingencies for this case.
—Deviating Seasonal Patterns are a major cause of inaccuracy in the extracted models. The extraction methods all assume a single, repeating *Seasonal Part*. Depending on the trace, this assumption may be valid to a different extent. In this case, the extracted Seasonal pattern is able to approximate most days in the trace, but a number of significant deviations occur. Manual modeling in the DLIM editor can circumvent this problem, as DLIM itself supports mixes of multiple seasonal patterns. We are currently working on extending the automated extractors to make use of this feature. Ideas range from the inclusion of additional metaknowledge, such as calendar information, to the implementation of seasonal break detection.

In the case of the Saskatchewan-HTTP extraction, noise reduction improves the s-DLIM results. However, overall the results are not as good as they are for the other traces. The major explanation for the relatively poor results is once more the *Seasonal*

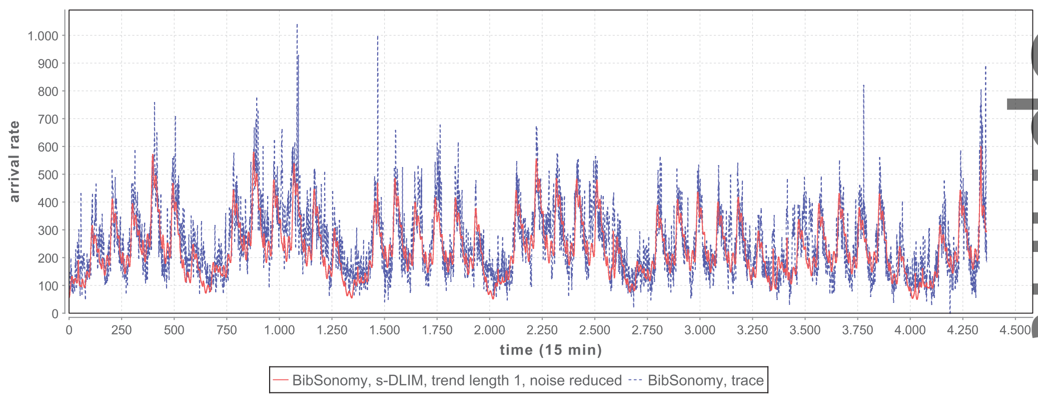

Fig. 7. Arrival rates of the original BibSonomy trace (blue) and the extracted DLIM instance (red) using s-DLIM with *Trend* length 1 and noise reduction.

pattern deviation. Since the Saskatchewan-HTTP trace extends over 7 months, the *Seasonal* patterns have a lot of room for deviation. The model extractors fail to capture this. This leads to an additional error in the *Trend* calibration, as trends are supposed to be calibrated, so that the greatest *Seasonal* peak in every *Seasonal* iteration matches the trace's nearest local arrival rate maximum. Since the *Seasonal* pattern deviation causes the extracted *Seasonal* peak's time of day to not match the trace's *Seasonal* peak's time of day, the calibration takes place at the wrong point of time. This also explains why a majority of extracted days have a lower peak than their counterparts in the original trace.

The major deviation from the trace's *Seasonal* patterns also explains why s-DLIM performs better using noise elimination for the Saskatchewan-HTTP extraction. Noise reduction helps to mitigate the effect of seasonal pattern changes over time, thus reducing the effect of the *Seasonal* pattern deviation.

Similarly to the Saskatchewan trace, s-DLIM extraction of the BibSonomy trace also improves with noise filtering. We explain this through the observation that the BibSonomy trace features a significant number of bursts, occurring at a relatively high frequency, as well as significant noise (as seen in Figure 7). Without filtering, some of these bursts are included in the seasonal pattern by the s-DLIM extractor, distorting the extracted seasonal pattern. When applying noise reduction, the influence of these bursts is diminished. Therefore, the extracted seasonal pattern is more stable, leading to increased accuracy as major bursts are still extracted during s-DLIM's burst extraction. The BibSonomy trace demonstrates that s-DLIM (and also p-DLIM) are capable of dealing with traces featuring a significant amount of noise.

p-DLIM performs well compared to the other two extraction processes. p-DLIM assumes that all trends repeat. In the case of the NASA trace, this assumption seems to be quite accurate. Even for the Saskatchewan trace, p-DLIM performs better compared to s-DLIM.

The hl-DLIM extraction shows an entirely different picture. Considering that hl-DLIM uses only a small number of predefined parameters, the extracted hl-DLIM instances are surprisingly close to the detailed DLIM models. Contrary to what was observed in the DLIM extraction, however, the hl-DLIM extraction strongly relies on noise reduction. If the noise is ignored and not filtered, hl-DLIM extraction accuracy drops dramatically. This can easily be attributed to the linear interpolation between the extracted peaks. Since hl-DLIM interpolates between the highest and lowest peak (thus only extracting two peaks), the nonfiltered trace offers a high number of noisy peaks with minimal impact on the overall arrival rate. The filtered version, however, only offers a few remaining peaks, which have a much higher impact on the overall arrival rate. Applying noise reduction forces the hl-DLIM extractor to only consider the peaks with significant impact rather than accidentally choosing outliers as peaks.

The WorldCup98 extraction results are notable in that s-DLIM and hl-DLIM extraction perform relatively well, whereas p-DLIM performs worst for all considered traces. The obvious cause of this is the observation that the WorldCup98 trace does not feature recurring trends and only features increasing trends. The s-DLIM and hl-DLIM extraction methods can handle this easily, whereas p-DLIM cannot.

The times series decomposition method STL shows worse accuracy values with the exception of the BibSonomy trace, for which it achieves a slightly better accuracy. In all cases, STL terminates a few milliseconds faster than DLIM extraction. BFAST terminates only for ClarkNet within 1.5h and achieves a comparable accuracy compared to s-DLIM. Due to the order of magnitude by which BFAST runtime differs from STL and DLIM, using it in an autonomous management context seems difficult.

5.2. Wikipedia and CICS

The second batch of traces was retrieved from the Wikipedia page view statistics.[4] They were parsed from the *projectcount* dumps, which already feature arrival rates with an hourly resolution. We restrict our analysis to the English, French, German, and Russian Wikipedia projects, covering four of the six most requested Wikipedia projects and being distributed over different time zones. All traces are from December 2013, with the exception of the English Wikipedia trace, which is from November 2013. The English December 2013 trace exhibits a major irregularity during the fourth day, which we attribute to a measurement or parsing error. While the French, German, and Russian Wikipedia projects are mostly accessed from a single time zone, the English Wikipedia is retrieved from all over the world; thus, evaluating the impact of access behavior over different time zones and helping to assess how well the DLIM extraction methods deal with local versus global access patterns.

In addition, we extract arrival rates from traces of the IBM CICS transaction processing system. These traces were logged in a banking environment with significantly different usage patterns during weekdays and weekends. These traces feature a quarter-hourly resolution (96 samples per day).

The Wikipedia extraction results in Table II confirm many of the observations made with the Internet Traffic Archive traces. Noise extraction is most useful for hl-DLIM extraction; *Trend* length of 1 as part of s-DLIM performs best. The overall accuracy, however, is significantly better than for the Internet Traffic Archive traces since the *Seasonal* pattern deviation, while still relevant, exhibits less impact than before.

The Russian Wikipedia trace differs from the other Wikipedia traces. Noise reduction also improves s-DLIM, while, as usual, being useful for hl-DLIM extraction. The overall accuracy is similar to the other Wikipedia trace extractions. For this single trace, however, the *Seasonal* patterns are shaped in such a way that the noise reduction lessens the impact of the *Seasonal* pattern deviation.

The extraction results for the English Wikipedia trace exhibit by far the best overall accuracy across all examined traces. The reason for this is the unusually high arrival rate base level. Since wikipedia.org is accessed globally at all times, the load intensity variations on top of the base level have little impact on the load variations in general. As a result, all modeling errors are also reasonably small.

[4]Wikipedia traces: `http://dumps.wikimedia.org/other/ pagecounts-raw/2013/`.

Table II. Wikipedia.org Model Extraction Errors

Trace	1. **German Wikipedia**	2. **French Wikipedia**	3. **Russian Wikipedia**	4. **English Wikipedia**	5. **IBM CICS**
Extraction Parameters	MdAPE	MdAPE	MdAPE	MdAPE	MdAPE
Extractor, Trend, Noise	[%]	[%]	[%]	[%]	[%]
s-DLIM, 1, extracted	11.215	10.472	9.964	7.764	71.311
s-DLIM, 1, eliminated	10.511	8.566	**9.912**	7.838	40.822
s-DLIM, 1, ignored	**8.538**	**7.600**	11.251	**4.855**	**29.199**
s-DLIM, 2, ignored	9.956	8.973	11.683	5.270	34.746
s-DLIM, 3, ignored	11.771	9.813	11.420	7.230	38.785
hl-DLIM, 1, extracted	11.898	8.503	12.392	7.750	80.043
hl-DLIM, 1, eliminated	11.393	8.373	12.496	7.961	59.956
hl-DLIM, 1, ignored	13.126	10.816	13.310	8.868	92.400
STL	13.309	8.671	6.747	2.580	71.997
BFAST	11.223	8.511	5.809	2.302	no result
avg. s-DLIM runtime	3.9ms	3.5ms	5.8ms	3.2ms	11.3ms
avg. STL runtime	7.5ms	7.0ms	7.0ms	7.5ms	11.3ms
avg. BFAST runtime	23,518ms	23,630ms	23,803ms	21,517ms	no result

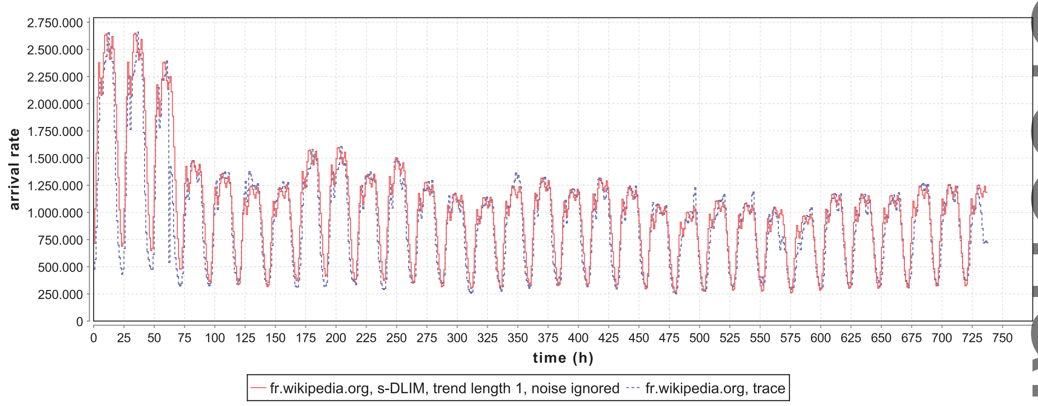

Fig. 8. Arrival rates of the original French Wikipedia trace (blue) and the extracted DLIM instance (red) using s-DLIM with *Trend* length 1 and ignoring noise.

In terms of accuracy, our extraction processes perform on the same level compared to the STL and BFAST decompositions for the Wikipedia traces and the CICS trace. s-DLIM performs better than STL and BFAST for both the German and French Wikipedia traces, as seen in Figure 8. Here, s-DLIM's accuracy profits from its support of multiplicative trends. STL and BFAST, however, provide better accuracy for the English and Russian traces. Comparing runtimes, s-DLIM is significantly faster than BFAST and slightly faster than STL. Running on the same machine, LIMBO's s-DLIM implementation performed on average 8,354 times faster than BFAST's R implementation and returned results in all cases in less than 0.2s.

For the CICS trace, STL has a high MdAPE value of 72%, while s-DLIM achieves a better but still high value of 29%. The CICS trace once more demonstrates the effect of seasonal deviation. It does not reach the accuracy of the Wikipedia workloads as one seasonal pattern cannot model the strong differences between workdays and weekends. To demonstrate that the modeling error is in large part caused by seasonal deviation, we extract a DLIM instance for the first five days in the CICS trace (Monday to Friday). This weekday model only features an MdAPE of 13.745%.

Table III. en.wikipedia.org Periodic Model Extension Errors

	Week 1, extraction	**Month, forecast**
Extraction Parameters	MdAPE	MdAPE
Extractor, Noise	[%]	[%]
p-DLIM, extracted	6.583	7.655
p-DLIM, eliminated	**6.332**	**7.292**
p-DLIM, ignored	8.082	8.546

6. APPLICATION SCENARIOS

This section presents three application domains for our DLIM approach for load profile modeling and extraction to underline the usefulness and adoption of our proposed approach. First, we illustrate the extrapolation capability of periodic DLIM models and propose a combination with statistical time series forecasting methods. The second domain is elasticity benchmarking and the third, design-time simulation. Two different application examples for the domain of design-time simulations show the benefits of DLIM-based workload descriptions for single applications as well as for resource planning in Infrastructure-as-a-Service (IaaS) scenarios.

6.1. DLIM Model as Baseline for Statistical Forecasting

Using the English Wikipedia trace, we demonstrate that future work on using p-DLIM in conjunction with forecasting methods is warranted. In cases of regular periodic user behavior, as is the case for Wikipedia, a fitted periodic DLIM model could serve as a baseline input for further forecasting efforts, for example, based on statistical time-series methods like ARIMA [Box et al. 2015]. DLIM model drift can be detected by repeated evaluation of the accuracy metric.

We extract a periodically repeating weekly trend from the first week of the Wikipedia trace. As with previous periodic extractions, we set the *Trend* list for the weekly repeating *Trend* to encompass segments of the length 3 and 4. Biweekly and monthly repeating *Trends* are omitted for this demonstration, as the shortened 1-week trace is not long enough. After extracting, we extend the extracted model by looping it for the entire monthly duration.

We evaluate the accuracy of the extended model using a pointwise comparison of the actually measured arrival rates and the arrival rates of the periodically looping DLIM instance. Table III shows the model extraction accuracy for the first week and the extended model's accuracy for the entire month. Extrapolation accuracy is high, as the noise filtered model reaches a MdAPE of 7.29%.

6.2. Elasticity Benchmarking

Autonomic adaptations in the resource allocation of elastic systems are usually triggered by changes in their workload, especially by changes in the number of arriving work units over time. It is the goal of elasticity benchmarking to measure how well elastic systems achieve the matching of resource demand and supply. The resource units of the individual systems may have different performance. The elasticity benchmarking concept proposed in Herbst et al. [2015] is based on DLIM load profiles as part of the workload definition. DLIM load profiles enable one to define a representative sequence of demand changes. The flexibility of DLIM load profiles is leveraged to scale the abstract definition to the capabilities of a system-under-test in terms of performance and scalability. This way, the sequence of demand changes can be kept identical across systems (with different performance characteristics) and allows both repeatable measurements and comparison between autoscaling alternatives.

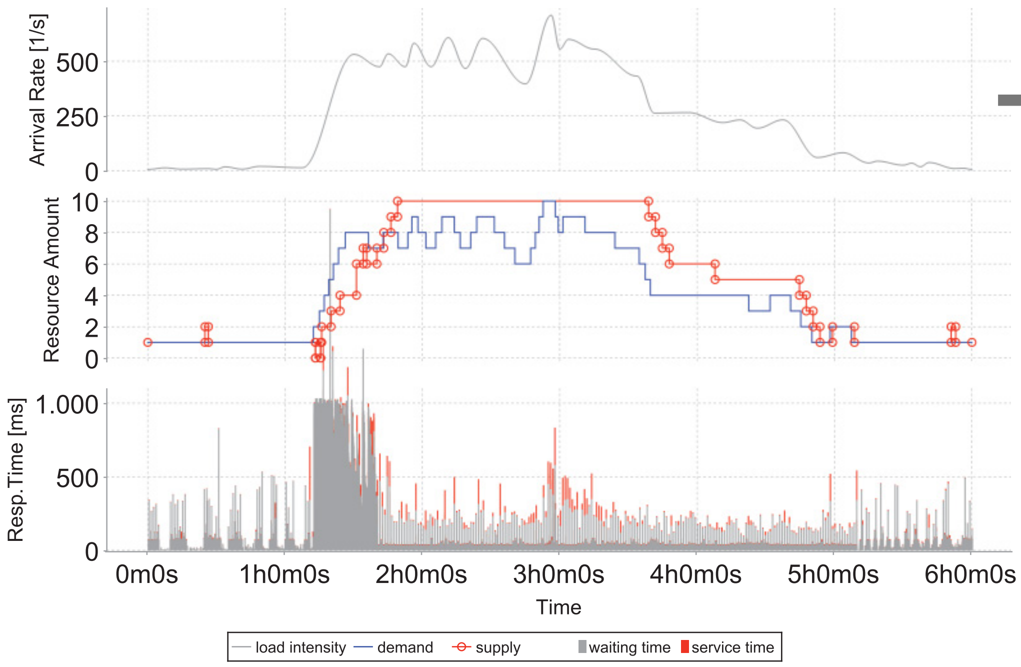

Fig. 9. Plots of an exemplary benchmark run on a public cloud.

Table IV. Elasticity Metrics for an Exemplary Benchmark Run

acc_O	acc_U	ts_O	ts_U	$jitter$
[#res.]	[#res.]	[%]	[%]	$\left[\frac{\#adap.}{min}\right]$
1.053	0.180	51.9	8.1	−0.033

In short, the elasticity benchmarking concept and its implementation called BUNGEE[5] comprises the following four steps:

(1) **Scalability Analysis:** The benchmark analyzes the System Under Test (SUT) with respect to the performance of its underlying resources and its scaling behavior.
(2) **Benchmark Calibration:** The results of the analysis are used to adjust the DLIM load intensity profile in a way that it induces the same resource demand on all compared platforms (i.e., the same sequence of demand changes).
(3) **Measurement:** The load generator exposes the SUT to a DLIM defined, time-varying, system-adjusted load profile. The benchmark monitors resource supply changes on the SUT.
(4) **Elasticity Evaluation:** Elasticity metrics are computed and used to compare the resource demand and resource supply curves with respect to different elasticity aspects.

The results of an exemplary benchmark run are plotted in Figure 9 together with derived metrics in Table IV. During the measurement step, resource demand and supply changes are recorded as plotted in the resource amount chart. Demand changes are derived from the time-varying load profile (represented by the gray line in the arrival rate graph) combined with the mapping result of the scalability analysis step. The response time chart visualizes the impact of the resource supply and demand

[5]BUNGEE Cloud Elasticity Benchmark: http://descartes.tools/bungee.

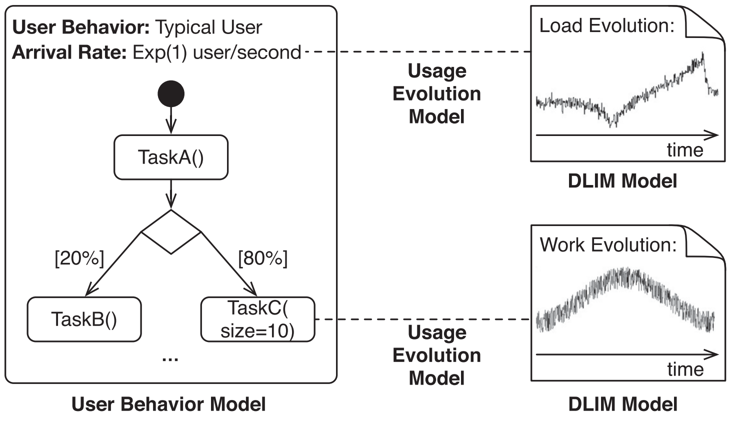

Fig. 10. Integration of user behavior model (left) and DLIM models (right).

matching on the service quality in terms of response time and timeouts. As metrics, the average amount of under- or overprovisioned resources (acc_O, acc_U) and the average time in an under- or overprovisioned state (ts_O, ts_U) are derived together with a metric *jitter* representing the average amount of missed (negative) or superfluous (positive) adaptations per time unit. A ranking of elastic systems or controllers can then be built for a fixed load profile scenario based on these metrics with a given weight definition and a baseline for normalization.

6.3. Design-Time Simulations

6.3.1. Integration of User Behavior Models. Design-time simulations like Palladio [Becker et al. 2009] mimic users of software systems according to user-specific behavioral models that do not change over time (see Section 2). In contrast, DLIM describes user arrival processes, allowing one to model dynamic load intensities but no user-specific behavior. To combine these two approaches, the CloudScale EU project [Brataas et al. 2013] integrated DLIM into the SimuLizar design-time simulator. In domains where workloads are time dependent, such an integration is crucial for the accuracy of design-time simulations. An example for such a domain is cloud computing because cloud applications dynamically react on workload changes over time.

Classical user behavior models characterize user interactions, for example, the probability of triggering a system task inducing a given amount of work. In CloudScale's integration, such models are linked to DLIM models via so-called *Usage Evolutions*. *Usage Evolutions* come in two variants: *Load Evolutions* and *Work Evolutions*.

Load Evolutions specify how load parameters evolve over time. In contrast, classical design-time simulations describe static and probabilistic load distributions. For instance, the *Arrival Rate* of typical users in the model depicted in Figure 10 is one user per second and exponentially distributed. *Load Evolutions* override such static arrival rate specifications depending on the type of workload. For open workloads, *Load Evolutions* specify the change of arrival rates over time (upper *Usage Evolution Model* in Figure 10). For closed workloads, *Load Evolutions* specify the change of the concurrent number of users over time.

Work Evolutions specify how work parameters evolve over time (lower *Usage Evolution Model* in Figure 10). In contrast, classical design-time simulations describe static work parameters, for example, a typical user in Figure 10 triggers *TaskC* after *TaskA* with a probability of 80% and with a work parameter *size* of 10 work units. *Work Evolutions* override such work specifications. For example, the lower *Usage Evolution Model* in Figure 10 overrides the specification of the size parameter by the specification of the linked *DLIM Model*, thus making the parameter time dependent.

At simulation time, SimuLizar updates workload parameters according to *Load* and *Work Evolutions*. For these updates, SimuLizar samples the linked DLIM models once per simulated time unit.

Load and Work Evolutions both extend the application context of DLIM models with user-specific behavioral models. Regarding *Load Evolutions*, the modeling of open workloads is close to the original DLIM context because DLIM models are used to specify arrival rates. In contrast, the modeling of closed workloads differs from the original DLIM context because DLIM models are used to specify the number of concurrent users. SimuLizar accordingly has to round sampled values to integer representations. Regarding *Work Evolutions*, the modeling of work parameters differs even more from the original DLIM context because arrival rates describe load parameters only.

Generalized, these observations indicate that DLIM fills a general need for modeling functions that vary over time. Whether such functions model arrival rates, user numbers, work parameters, or other attributes depends on the application domain. For example, in the domain of data center resource planning, modeling variations of resource demands becomes important and can indeed be modeled with DLIM (Section 6.3.3).

6.3.2. Simulating Elasticity and Cost-Efficiency Metrics. In this section, we describe how design-time simulations can assess properties that depend on dynamic workloads. In the cloud computing context, such properties are elasticity and cost efficiency [Lehrig et al. 2015]. The basis for design-time simulations of these properties is the DLIM model integration with user behavior models as described in the previous section: dynamic workloads are described by Usage Evolutions and autonomous adaptations can be modeled with SimuLizar.

SimuLizar currently supports the following elasticity and cost-efficiency metrics (cf. Becker et al. [2015]). The *number of service level objective violations* metric counts the number of violated performance requirements during adaptation phases. The *mean time to quality repair* metric measures the time cloud applications need to move from a state that violates performance requirements to a state that satisfies all performance requirements. The *cost over time* metric computes the operation costs accrued for using cloud computing resources per billing interval.

Elasticity and cost-efficiency metrics enable software architects to conduct according trade-off analyses with SimuLizar (service level objective violations vs. costs). The CloudScale project illustrates such analyses with DLIM integration within their *CloudStore*[6] case study.

CloudStore is an example cloud migration scenario in which software architects migrate an existing online book shop (based on a legacy implementation of the TPC-W benchmark [Council 2002]) to a cloud computing environment. Dynamic workloads are expected for CloudStore, for example, because books sell better around Christmas (increased load) and the book database is expected to grow (increased work).

Based on according Usage Evolution models for work and load evolutions, SimuLizar can simulate elasticity and cost-efficiency metrics for CloudStore. The simulation results of the existing online book shop show that the *number of service level objective violations* steadily increase over time and that the *mean time to quality repair* cannot be determined because at no point in time can a state that satisfies all performance requirements be reestablished. *Cost over time* remain constant as the existing online book shop is nonelastic and, thus, no additional resources are acquired.

[6]CloudStore https://github.com/CloudScale-Project/CloudStore.

Because these results do not satisfy CloudStore's software architects, they run CloudScale's design-time performance antipattern detection [Brataas et al. 2013]. This detection points to issues at CloudStore's business layer: wrongly configured database connection pools cause congestion effects, that is, a steadily increasing number of waiting jobs that cause the increasing number of service level objective violations.

The identified antipattern requires CloudStore's software architects to re engineer CloudStore's architecture. For such a re engineering, the architects apply CloudScale's reusable analysis templates [Lehrig 2014] that fit to the detected antipattern. In the CloudStore case study, software architects apply a template for horizontally scalable databases, thereby reconfiguring the existing database connection pool to dynamically react on workload changes.

For the re engineered CloudStore version, SimuLizar's metrics show that the *number of service level objective violations* is significantly lowered: only transient phases during which reconfigurations execute cause violations. Moreover, *mean time to quality repair* indicates that such phases last less than 10s. Because the re engineered CloudStore dynamically changes the number of consumed cloud computing resources, different *cost over time* accrue over time. These costs depend on the concrete cloud computing environment analyzed (CloudScale provides analysis templates for SAP HANA Cloud, Amazon EC2, and OpenStack). Simulation results for costs allow software architects to compare such environments on an objective and quantitative basis.

6.3.3. Data Center Resource Planning. The main use case presented so far is the extraction and modeling of variable user arrival rates. This section shows the added value of DLIM across domains and presents the transfer to the use case of modeling the behavior of Virtual Machines (VMs) in data centers. Data center operators cannot monitor VM internals but require information on the resource demand for placement and optimization decisions. For example, data center simulators [Calheiros et al. 2011] and scheduling algorithms [Beloglazov et al. 2012] use resource demand descriptions to reason on and optimize the QoS and energy efficiency of deployed VMs. This use case shows that replacing the arrival values with resource load values allows using DLIM for decision support in data center planning and management.

This research and validation is part of the CACTOS project developing a methodology and tools for large-scale data center planning and runtime management [Östberg et al. 2014]. The decisions are supported at runtime or as part of what-if analyses using predictions, both with the same DLIM instances. Groenda and Stier [2015] extended the mechanisms shown in Section 6.3.1 to support varying resource demand specifications for VMs.

VMs show dynamic resource demand variations when monitored from the outside and are black boxes for data center operators. Operators have to reason on metrics that can be observed without internal knowledge of running applications. These metrics include CPU utilization, disk, and network accesses of VMs.

Resource demands specify how much demand of a specific *type* of resource a service requires in order to be executed. Different approaches for resource demand estimation [Spinner et al. 2014] can be used to derive resource demand estimations from traces. The mapping in Figure 11 uses these techniques to infer the resource demands. Execution-platform independent demand specifications allow reasoning on relocation decisions. Palladio [Becker et al. 2009] and other predictive performance models therefore use these kinds of resource demands. The approach is explained in the following paragraphs.

Figure 11 illustrates how DLIM can be leveraged to model the load caused by black-box VMs. Every black-box VM has a set of resource demands describing the load it issues over time. In the example, VM A's resource demand is described in terms of the

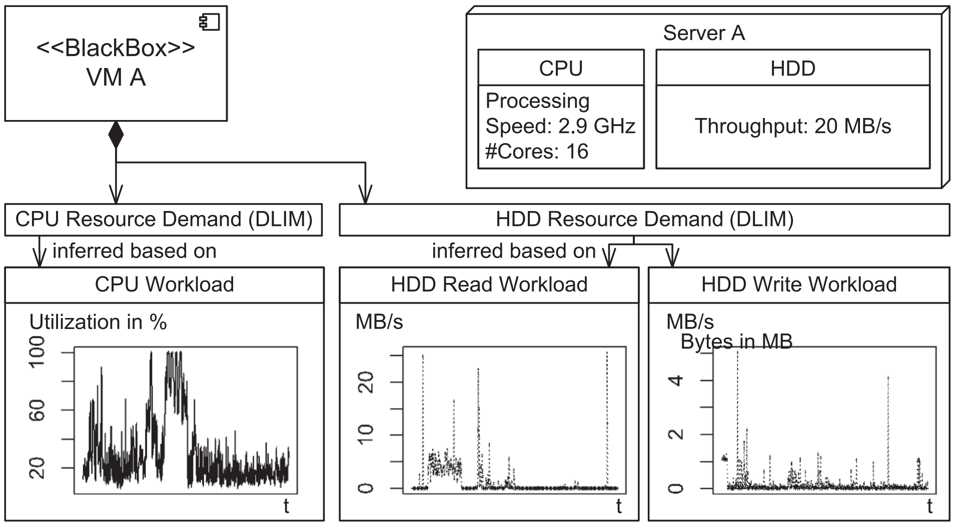

Fig. 11. Exemplary mapping of trace-based workload models to resource demand DLIM specifications.

CPU utilization and HDD accesses observed on Server A. DLIM's piecewise-defined functions that map points in time to arrival rates hereby model the *resource demand intensity* over time. The resource demand intensity describes the rate at which a VM issues resource demands. These models reference the platform on which the resource demand was monitored. This enables adjustments of the estimated demands based on the deployment environment.

Our extraction of varying resource demand for use in simulation is carried out as part of an offline batch layer. An analysis component extracts resource load intensity models from historic traces by means of linear interpolation. Our algorithm takes VM placement and migration between nodes during the period of the trace into account.

Resource demand intensity modeling for data center management and planning benefits from the flexibility of DLIM. A simple function application to an existing DLIM resource load intensity curve allows the evaluation of the model as part of an alternative data center deployment scenario. In summary, DLIM allows for a compact representation of varying resource demand for use in data center planning and management. With minimal extensions DLIM can be used to capture platform-independent descriptions of resource demand intensities that can easily be transformed for platform-specific load analyses and simulations.

7. CONCLUSIONS

This article presents the DLIM for capturing load profiles by a structured combination of piecewise mathematical functions. We introduce three methods enabling the automated extraction of DLIM instances from existing arrival rate traces:

—**Simple DLIM Extraction** (s-DLIM): Extracts DLIM instances from existing arrival rate traces. s-DLIM exhibits an accuracy with an average median error of 12.4% when optimizing the extraction configuration for each trace.

—**Periodic DLIM Extraction** (p-DLIM): Extracts DLIM instances from existing arrival rate traces. In contrast to s-DLIM, the trends within p-DLIM instances are intended to be repeated. This enables p-DLIM to be used for extrapolation, that is, in load intensity forecasting.

—**hl-DLIM Extraction**: Extracts hl-DLIM instances from existing arrival rate traces. Due to hl-DLIM's restrictions, this method is less accurate than s-DLIM. The limited accuracy, however, can be improved by applying noise reduction during the extraction process.

The results of our evaluation showed that the proposed model extraction methods are capable of extracting DLIM instances with an average modeling error of only 15.2% (MdAPE) over 10 different real-world traces that cover between 2 weeks and 7 months. The model extraction performs best for the Wikipedia traces. Extracted *Seasonal* patterns match the trace's days well and the overlying Trends are precisely calibrated. Concerning the Internet Traffic Archive traces, we identified the seasonal pattern deviations for traces extending over several months as a major challenge for future work. Changes of daily usage patterns over the course of these particularly long traces lead to a decrease in accuracy. Nevertheless, the median error remains below 27%. Furthermore, the BibSonomy trace demonstrates that the extraction mechanisms are robust and capable of dealing with noisy arrival rate patterns. The results in general show that DLIM itself is capable of accurately capturing real-world load intensity profiles, independent of the explicit extraction processes we introduce. In addition to the evaluation, three separate application scenarios demonstrate the applicability of DLIM in different contexts.

7.1. Future Work

Beyond these already existing applications of DLIM, we envision the use of automatically extracted load intensity profiles as part of autonomic and self-aware system management by employing them in the following contexts:

—*Load Forecasting*: Automatically extracted load intensity profiles can be used to forecast the changes in arrival rates at runtime. This, in turn, enables pro-active resource management and system adaptation with low computational overhead.
—*Anomaly Detection*: A load profile model can serve as a baseline for the online detection of anomalous load behavior, such as unplanned bursts.
—*Load Classification*: The compact information about load behavior contained within the model can be used to classify profile categories, enabling dynamic distinction between user or application types based on profile characteristics.

As part of future work on the LIMBO approach and the model instance extraction methods themselves, we plan the implementation of more advanced model refinement and calibration features. Our primary target will be the mitigation of the effect of *Seasonal* pattern deviation. Another avenue of future work will be the adaptation of the model instance extraction methods for workload forecasting. The accuracy of p-DLIM compared to s-DLIM in the NASA and Saskatchewan traces already shows that further work on the use of p-DLIM for load intensity forecasting is warranted. We are also working on extending LIMBO to provide compatibility with additional existing benchmarking frameworks.

REFERENCES

Paul Barford and Mark Crovella. 1998. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'98/PERFORMANCE'98)*. ACM, New York, NY, 151–160. DOI:http://dx.doi.org/doi:10.1145/277851.277897

Matthias Becker, Sebastian Lehrig, and Steffen Becker. 2015. Systematically deriving quality metrics for cloud computing systems. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE'15)*. ACM, New York, NY, 169–174. DOI:http://dx.doi.org/10.1145/2668930.2688043

Steffen Becker, Heiko Koziolek, and Ralf Reussner. 2009. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* 82, 1 (2009), 3–22. DOI:http://dx.doi.org/10.1016/j.jss.2008.03.066. Special Issue: Software Performance - Modeling and Analysis.

Aaron Beitch, Brandon Liu, Timothy Yung, Rean Griffith, Armando Fox, and David A. Patterson. 2010. *Rain: A Workload Generation Toolkit for Cloud Computing Applications*. Technical Report UCB/EECS-2010-14. EECS Department, University of California, Berkeley. Retrieved from http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-14.html.

Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. 2012. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems* 28, 5 (May 2012), 755–768.

Dominik Benz, Andreas Hotho, Robert Jäschke, Beate Krause, Folke Mitzlaff, Christoph Schmitz, and Gerd Stumme. 2010. The social bookmark and publication management system BibSonomy. *The VLDB Journal* 19, 6 (Dec. 2010), 849–875. DOI:http://dx.doi.org/10.1007/s00778-010-0208-4

Gordon Blair, Robert B. France, and Nelly Bencomo. 2009. Models@ run.time. *Computer* 42 (2009), 22–27. DOI:http://dx.doi.org/doi.ieeecomputersociety.org/10.1109/MC.2009.326

Herman J. Blinchikoff and Anatol I. Zverev. 1986. *Filtering in the Time and Frequency Domains*. Krieger Publishing Co., Inc.

George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. 2015. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons.

Gunnar Brataas, Erlend Stav, Sebastian Lehrig, Steffen Becker, Goran Kopčak, and Darko Huljenic. 2013. CloudScale: Scalability management for cloud systems. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE'13)*. ACM, New York, NY, 335–338. DOI:http://dx.doi.org/10.1145/2479871.2479920

Rodrigo N. Calheiros and more. 2011.CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience* 41, 1 (Jan. 2011), 23–50.

Giuliano Casale, Amir Kalbasi, Diwakar Krishnamurthy, and Jerry Rolia. 2012. BURN: Enabling workload burstiness in customized service benchmarks. *IEEE Transactions on Software Engineering* 38, 4 (2012), 778–793. DOI:http://dx.doi.org/10.1109/TSE.2011.58

Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. 1990. STL: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics* 6, 1 (1990), 3–73.

Transaction Processing Council. 2002. TPC-W Benchmark (Web Commerce) Specification Version 1.8. Original link http://www.tpc.org/tpcw/spec/tpcw_V1.8.pdf now at http://www.pdf-archive.com/2016/05/20/tpcw-v1-8/. (Feb. 2002).

Faban. 2006. Homepage. Retrieved from http://faban.org.

Dror G. Feitelson. 2002. Workload modeling for performance evaluation. In *Performance Evaluation of Complex Systems: Techniques and Tools*, MariaCarla Calzarossa and Salvatore Tucci (Eds.). Lecture Notes in Computer Science, Vol. 2459. Springer, Berlin, 114–141. DOI:http://dx.doi.org/doi:10.1007/3-540-45798-4_6

Jerome H. Friedman. 1991. Multivariate adaptive regression splines. *The Annals of Statistics* (1991), 1–67.

Henning Groenda and Christian Stier. 2015. Improving IaaS cloud analyses by black-box resource demand modeling. *Softwaretechnik-Trends* 35, 3 (2015). in print.

Emily H. Halili. 2008. *Apache JMeter: A Practical Beginner's Guide to Automated Testing and Performance Measurement for Your Websites*. Packt Publishing Ltd.

Nikolas Roman Herbst, Samuel Kounev, Andreas Weber, and Henning Groenda. 2015. BUNGEE: An elasticity benchmark for self-adaptive IaaS cloud environments. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'15)*. IEEE Press, 46–56.

Rob J. Hyndman and Anne B. Koehler. 2006. Another look at measures of forecast accuracy. *International Journal of Forecasting* (2006), 679–688.

Jeffrey O. Kephart and David M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (Jan. 2003), 41–50. DOI:http://dx.doi.org/10.1109/MC.2003.1160055

Max Kuhn, Steve Weston, Chris Keefer, and Nathan Coulter. 2012. Cubist Models For Regression. http://cran.r-project.org/web/packages/Cubist/vignettes/cubist.pdf.

Sebastian Lehrig. 2014. Applying architectural templates for design-time scalability and elasticity analyses of SaaS applications. In *Proceedings of the 2nd International Workshop on Hot Topics in Cloud Service Scalability (HotTopiCS'14)*. ACM, New York, NY, Article 2, 8 pages. DOI:http://dx.doi.org/10.1145/2649563.2649573

Sebastian Lehrig, Hendrik Eikerling, and Steffen Becker. 2015. Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA'15)*. ACM, New York, NY, 83–92. DOI:http://dx.doi.org/10.1145/2737182.2737185

Hui Li. 2010. Realistic workload modeling and its performance impacts in large-scale eScience grids. *IEEE Transactions on Parallel and Distributed Systems* 21, 4 (2010), 480–493. DOI:http://dx.doi.org/doi:10.1109/TPDS.2009.99

Daniel A. Menascé, Virgílio A. F. Almeida, Rudolf Riedi, Flávia Ribeiro, Rodrigo Fonseca, and Wagner Meira, Jr. 2003. A hierarchical and multiscale approach to analyze e-business workloads. *Performance Evaluation* 54, 1 (Sept. 2003), 33–57. DOI:http://dx.doi.org/doi:10.1016/S0166-5316(02)00228-6

P.-O. Östberg and more. 2014. The CACTOS vision of context-aware cloud topology optimization and simulation. In *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE Computer Society, 26–31. DOI:http://dx.doi.org/10.1109/CloudCom.2014.62

John R. Quinlan and others. 1992. Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, Vol. 92. 343–348.

Arcadio Reyes-Lecuona, E. González-Parada, E. Casilari, J. C. Casasola, and A. Diaz-Estrella. 1999. A page-oriented WWW traffic model for wireless system simulations. In *Proceedings ITC*, Vol. 16. 1271–1280.

S. Roy, T. Begin, and P. Goncalves. 2013. A complete framework for modelling and generating workload volatility of a VoD system. In *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. 1168–1174. DOI:http://dx.doi.org/doi:10.1109/IWCMC.2013.6583722

Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. 2006. Open versus closed: A cautionary tale. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3 (NSDI'06)*. USENIX Association, 18–18. http://dl.acm.org/citation.cfm?id=1267680.1267698

Simon Spinner, Giuliano Casale, Xiaoyun Zhu, and Samuel Kounev. 2014. LibReDE: A library for resource demand estimation. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE'14)*. ACM, New York, NY, 227–228. DOI:http://dx.doi.org/10.1145/2568088.2576093

André van Hoorn, Matthias Rohr, and Wilhelm Hasselbring. 2008. Generating probabilistic and intensity-varying workload for web-based software systems. In *Proceedings of the SPEC International Workshop on Performance Evaluation: Metrics, Models and Benchmarks (SIPEW'08)*. Springer-Verlag, Berlin, 124–143. DOI:http://dx.doi.org/doi:10.1007/978-3-540-69814-2_9

Jan Verbesselt, Rob Hyndman, Glenn Newnham, and Darius Culvenor. 2010. Detecting trend and seasonal changes in satellite image time series. *Remote Sensing of Environment* 114, 1 (2010), 106–115. DOI:http://dx.doi.org/10.1016/j.rse.2009.08.014

Joakim Gunnarson von Kistowski, Nikolas Roman Herbst, and Samuel Kounev. 2014. Modeling variations in load intensity over time. In *Proceedings of the 3rd International Workshop on Large-Scale Testing (LT 2014), Co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM.

Xiaozhe Wang, Kate Smith-Miles, and Rob Hyndman. 2009. Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series. *Neurocomputing* 72, 10–12 (June 2009), 2581–2594. DOI:http://dx.doi.org/10.1016/j.neucom.2008.10.017

Netanel Zakay and Dror G. Feitelson. 2013. Workload resampling for performance evaluation of parallel job schedulers. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE'13)*. ACM, New York, NY, 149–160. DOI:http://dx.doi.org/doi:10.1145/2479871.2479893

Received September 2015; revised November 2016; accepted November 2016