Modeling and Extracting Load Intensity Profiles

Jóakim v. Kistowski, Nikolas Herbst, Daniel Zoller, Samuel Kounev and Andreas Hotho
University of Würzburg, Germany
{joakim.kistowski, nikolas.herbst, daniel.zoller, samuel.kounev, andreas.hotho}@uni-wuerzburg.de

***Abstract*—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. Benchmarking of systems under these constraints is difficult, as state-of-the-art benchmarking frameworks provide only limited support for emulating such dynamic and highly variable load profiles for the creation of realistic workload scenarios. Industrial benchmarks typically confine themselves to workloads with constant or stepwise increasing loads. Alternatively, they support replaying of recorded load traces. Statistical load intensity descriptions also do not sufficiently capture concrete pattern load profile variations over time.**

To address these issues, we present the *Descartes Load Intensity Model* (DLIM). DLIM provides a modeling formalism for describing load intensity variations over time. A DLIM instance can be used as a compact representation of a recorded load intensity trace, providing a powerful tool for benchmarking and performance analysis. As manually obtaining DLIM instances can be time consuming, we present three different automated extraction methods, which also help to enable autonomous system analysis for self-adaptive systems. Model expressiveness is validated using the presented extraction methods. Extracted DLIM instances exhibit a median modeling error of 12.4% on average over nine different real-world traces covering between two weeks and seven months. Additionally, extraction methods perform orders of magnitude faster than existing time series decomposition approaches.

I. 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 becomes a crucial information for managing the underlying IT resource landscape. Load profiles with large amounts of concurrent users are typically strongly influenced by deterministic patterns due to human habits, trends, calendar effects, and events. The performance evaluation of systems under these dynamic conditions poses new challenges. Benchmarking frameworks such as Faban[1], Rain [2], and JMeter [8] allow request injection rates to be configured either to constant values, or to stepwise increasing rates (e.g., for stress tests) or variable rates based on recorded workload traces. The challenges arising when applying open workload models for benchmarking are not thoroughly addressed by these frameworks, as varying load intensity profiles are a common observation in real world systems and require consideration in benchmarking.

In this paper, we introduce the *Descartes Load Intensity Model* (DLIM). DLIM offers a fine-grained, structured and accessible MOF-based meta-model to describe load intensity profiles by editing and combining piece-wise mathematical functions. At a higher abstraction level, we propose 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. A preliminary sketch of the initial ideas for these two models has been presented in [19]. This work-in-progress description also misses in-depth validation of the then incomplete models. DLIM can be used to define an arbitrary dynamic load intensity profile that can be leveraged for benchmarking purposes to evaluate the behavior of a system under different dynamic workload scenarios (e.g., bursty workloads, seasonal load spikes). This is useful in many use-cases, e.g. for both on-line and off-line evaluation of the quality of system adaptation mechanisms such as elastic resource provisioning techniques in modern cloud environments. DLIM and hl-DLIM are both MOF-based meta-models for load intensity description. This allows the use of tools and techniques provided for model driven development in conjunction with DLIM. As such, we offer a model-to-model transformation from hl-DLIM to the detailed DLIM. 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 type information, which is useful when analyzing or modifying load intensity variations. Further DLIM-based applications and developments in the fields of benchmarking and system resource management at run-time are enabled by providing automatic model extraction processes. Specifically, 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 run-time. This, in turn, enables pro-active resource management and system adaptation.
- **Anomaly Detection**: A load profile model can serve as a baseline for the on-line 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.

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. Given a trace, the trace can be represented in a mathematical form as a compact DLIM model instance. The latter captures the major properties of the trace (e.g., burstiness, seasonality, patterns and trends)

[1]Faban `http://faban.org`

and can be used at any time to automatically generate an equivalent trace (exhibiting the same properties). Furthermore, starting with an extracted model instance, the instance can be easily modified to reflect a given target dynamic load scenario under investigation, e.g., changing the frequency of bursts or adding a given trend behavior. Load intensity profiles, represented as DLIM model instances, can be used in a variety of application scenarios, e.g., for emulating request/job arrivals in benchmarking experiments, or for analysing mathematical properties of real-life workloads (e.g., burstiness, seasonality). In the latter case, load intensity profiles provide valuable information for performance modeling and capacity planning studies. Manual construction and maintenance of DLIM model instances becomes infeasible in complex scenarios or at run-time usage. hl-DLIM addresses this by providing a more concise way for load intensity profile description. This enables the quick and easy creation of an initial model instance. For the easy creation of model instances based on real-world data, we introduce and validate three automated DLIM model extraction methods as presented in detail in Section VI: First, the *Simple DLIM Extraction Method* (s-DLIM) for DLIM instances is based on the idea of time series decomposition as taken in Breaks For Additive Season and Trends (BFAST) [17]. Second, the *Periodic DLIM Extraction Method* (p-DLIM) for DLIM instances features various types of repeating patterns. Third, the *hl-DLIM Extraction Method* extracts hl-DLIM instances on a higher abstraction level than DLIM. We highlight as major benefits of this work the new capabilities to accurately automatically extract load intensity models (12,7% median modeling error on average) from a representative set of nine different real-world traces. Each extraction completes in less than 0.2 seconds. These results demonstrate and validate the capability of DLIM to capture realistic load intensity profiles. An implementation of the models and extraction methods is available as part of the LIMBO toolkit[2]. LIMBO's architecture and extensibility options are further described in [18].

The rest of this paper is structured as follows: Section II discusses existing workload modeling approaches, Section III explains open workloads and our definition of load intensity. Sections IV and V describe the two models in detail, with the model instance extraction methods being discussed in detail in Section VI. An evaluation of the proposed models and extraction processes based on real world traces is conducted in Section VII. The paper wraps up with our conclusions in Section VIII.

II. Related Work

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 works purely statistical using independent random variables and therefore do not offer models describing load intensity changes over time. Second, approaches for workload or user behavior modeling model 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, not user behavior and impact after arrival. This is done by combining both deterministic and statistical approaches, which goes beyond existing purely statistical modeling approaches. We group related work into at least one of the following three categories:

- **User behavior models**: [16], [14] and [2] propose workload models that capture the behavior and tasks triggered by different types of users. [20] partition workloads according to the user types, and then sample workload traces for each user type to capture the user behavior. This differs from our approach, which is focused on modeling user arrival processes. We note that models like the above 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 modeling**, modeling the specific work units: These approaches focus on modeling the units of work processed by the system and estimating the system's resource demands. [5] focus on modeling bursty workloads, whereas [1] focus on file distribution and popularity.
- **Statistical inter-arrival models**: These approaches capture the workload intensity using statistical distributions. [6] creates a statistical model for parallel job schedulers. [10] models batch workloads for eScience grids and [11] as well as [13] 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 inter-arrival rate distributions. These random variables do not describe changes in load intensity behavior over time.
- **Regression techniques**, such as MARS [7], M5 trees [12], or cubic forests [9] are capable of calibrating mathematical functions to fit a measured arrival rate trace. In contrast to DLIM, they do not convey the additional information of the types and composition of load intensity variation components.

The combined deterministic and statistical approach of DLIM enables a better mapping between arrival rate variations and their respective time-stamps. A composite piece-wise mathematical function, as used in DLIM, can capture load intensity profiles more effectively than independent random variables. This approach also enables a better understanding of a benchmark's behavior for users intending to use DLIM for benchmarking purposes. The drawback of a deterministic model is the difficulty in capturing random behavior. For this reason DLIM includes an optional random noise element, which deviates from the otherwise deterministic functions and enables a combined deterministic and statistical modeling approach.

III. Foundations

Both DLIM and hl-DLIM have been designed to capture variations of load intensity in the form of user, job, or request arrival rates. For this, they employ an **open workload** view. Schroeder et al. [15] define open workloads as workloads, in which new jobs arrive independently of job completions. DLIM or hl-DLIM instances describe the intensity profile of an open workload. They do not make any assumptions about the completion times of the work units. This decision is based on the assumption that users in a typical cloud environment are

[2]LIMBO `http://descartes.tools/limbo`

unaware of one another and access a software service without having any knowledge of other users' behavior.

For this paper, we define **load intensity** as a function describing *arrival rates* of workload units (e.g. users, sessions, or requests) over time. We define the *arrival rate* $r(t)$ at time t as follows:

$$r(t) = R'(t)$$
$$\text{with } R(t) = |\{u_{t_0} | t_0 \leq t\}|$$

where $R(t)$ is the amount of *work units* u_{t_0} with *arrival time* t_0 that have arrived up until time t. R' is the derivative thereof.

IV. Descartes Load Intensity Model

The Descartes Load Intensity Model (DLIM) describes arrival rates over time and is visualized in Fig. 1. A more abstract preliminary sketch was initially presented in [19]. Specifically, the model is aimed at describing the variations of work unit arrival rates by capturing characteristic load intensity behaviors. DLIM achieves this by defining piece-wise mathematical functions to approximate variable arrival rates, with support for periodicity, flexibility to adapt and incorporate unplanned events, and composability of model instances.

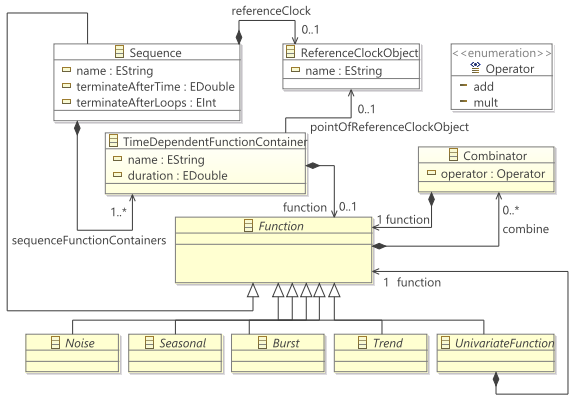

Fig. 1. The Descartes Load Intensity Meta-Model without the child implementations of the abstract *Noise*, *Burst*, *Seasonal*, and *Trend*.

Being a composition of piece-wise mathematical functions, DLIM uses a *Sequence* of functions as its central element. Functions may be added or multiplied with one another using mathematical operators. The result of this approach is a sequence of function-trees, which describe the arrival rate behavior during a time period as defined by the containing *Sequence*. Specifically, a *Sequence* carries a number of *TimeDependentFunctionContainers*, which describe the duration of each interval and are executed in sequence. The containers, in turn, contain the actual mathematical functions describing the arrival rates. The *Sequence*'s execution repeats as many times as indicated by the *terminateAfterLoops* attribute. Alternatively, the sequence repeats for the time indicated by the *terminateAfterTime* attribute. If both are set, we calculate the final duration as the minimum of either the looping time or the specific *terminateAfterTime* attribute. Infinite sequences are not allowed in order to guarantee termination. Any *Function* can be combined with other *Functions* using a *Combinator*, which results in a *TimeDependentFunctionContainer* carrying an entire tree of functions. The *TimeDependentFunctionContainer* describes its arrival rates for a set *duration*, after which the next *TimeDependentFunctionContainer* in the parent *Sequence's* list is processed.

Function is the abstract parent class to mathematical functions contained within a *TimeDependentFunctionContainer*. Being abstract, it cannot be instantiated. Instead a number of non-abstract children are provided that can be used as *Functions*. The most notable concrete *Function* is the *Sequence*, which effectively means that any *TimeDependentFunctionContainer* may hold a *Sequence* in its *Function* tree. This contained *Sequence* must be unique, preventing cyclical containment dependencies. The other concrete *Functions* fall into one of the following categories (each represented by their abstract super-class): *Seasonal*, *Burst*, *Noise*, or *Trend*.

A *Function* holds a list of *Combinators*. A *Combinator* allows the combination of this *Function*'s arrival rates with the arrival rates generated by other concurrently running *Functions*. The *Combinator* contains operators such as $+$ and $*$. Any *Function* contained within a *Combinator* is defined for the exact same duration as its containing parent *Function*.

V. High-level DLIM

DLIM offers a convenient way of structuring and ordering functions for the description of load intensity profiles. However, from the standpoint of a human user, it provides limited abstract knowledge about those variations, as the tree of composite piece-wise mathematical functions may be difficult to grasp. hl-DLIM addresses this issue by providing means to capture load intensity variations with a limited number of parameters. These parameters can then be used to quickly define and characterize a load intensity model. Inspired by the time-series decomposition approach in BFAST [17], hl-DLIM is split into a *Seasonal* and *Trend* part. Additionally, as hl-DLIM is intended for modeling load profiles, it 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 underlying composite 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 Fig. 2): *period*, *number of peaks*, *base arrival rate level*, *first peak arrival rate*, and *last peak arrival rate*. Additional peak arrival rates are derived using linear interpolation.

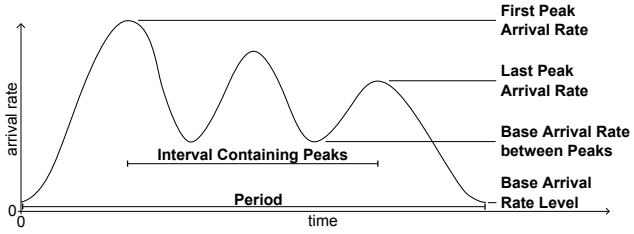

Fig. 2. hl-DLIM *Seasonal* Part.

The *Trend* part describes an overarching function that captures the overall change in the load intensity over multiple seasonal periods. It does so by constructing a list of equi-length *Trend* segments. These segments describe the respective arrival rates at their start and end points. The arrival rate of the *Seasonal* part is then interpolated to match the segment's arrival rate. In contrast to the *Trend* within BFAST, the hl-DLIM *Trend* can interact with the *Seasonal* part either by addition or multiplication. 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. The *Trend* segment functions are defined so that they always begin and end at the largest *Seasonal Peak*. As a result, the values contained in this list define the resulting maximum peak after applying the *Trend* at the corresponding point in time. The point in time at which each arrival rate in this list is defined is always the time of the largest peak in a *Seasonal* iteration.

The *Burst* part allows the definition of recurring bursts, which are added onto the existing *Seasonal* and *Trend* behavior. 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 a uniform distributed noise function defined by its *Minimum Noise Rate* and *Maximum Noise Rate*. Other Noise distributions can easily be added to DLIM instances, which are obtained from hl-DLIM instances via a model-to-model transformation.

VI. Model Instance Extraction

In this section, we present three methods for the extraction of DLIM instances from arrival rate traces. Each method requires a set of configuration parameters. Some of the steps of the methods can be easily performed manually, others require automation in order to be completed within a reasonable time frame. 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 used in BFAST [17]. s-DLIM extracts a repeating *Seasonal Part* and a non-repeating *Trend Part*. The non-repeating *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 Fig. 3.
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) **hl-DLIM Extraction Method**:
   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.

A. 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*.

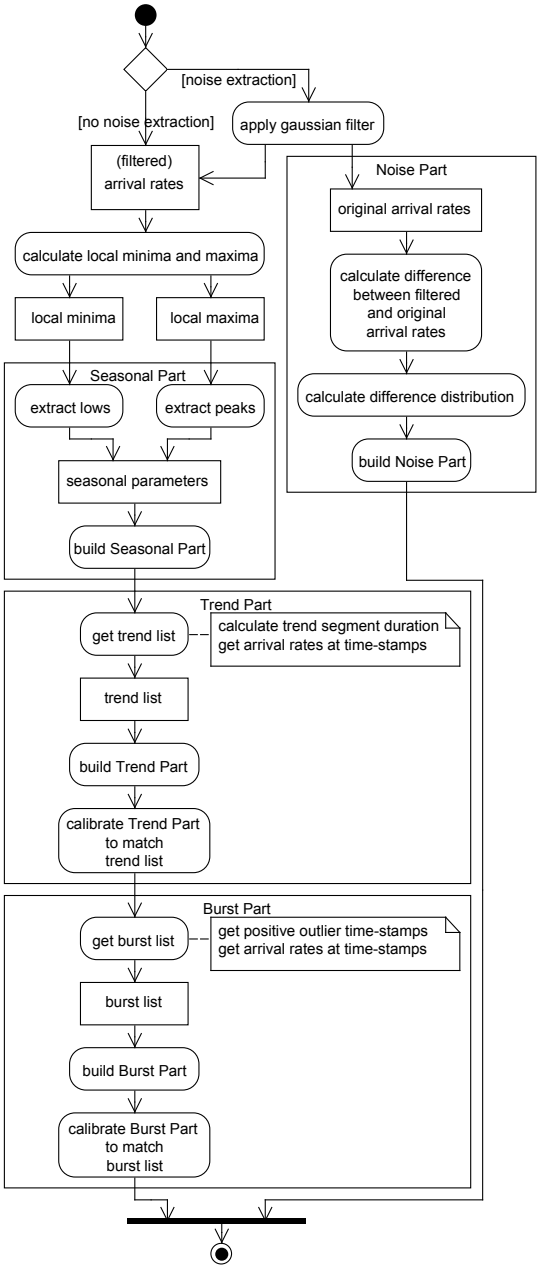

Fig. 3. Activity Diagram of the *Simple DLIM Extraction Method* (s-DLIM).

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 seasonal period
- Arrival rate peaks and their time-stamps
- Arrival rate lows and their time-stamps
- Function type used to interpolate between peaks and lows.

The seasonal period (length) is set by the user. It is usually selected using meta-knowledge about the trace. A trace that extends for multiple days and contains daily patterns, for example, features a period of 24 hours. The peaks and lows are

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** $i \leftarrow 0$ **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** $i \leftarrow 0$ **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** $i \leftarrow 0$ **to** peakNum $- 1$ **do**
  - interpolatingFunction ← DLIM *Function* starting at low [i], ending at peak [i];
  - rootSequence.`append` (interpolatingFunction);
- **end**

end

Algorithm 1: Extracting the *Seasonal* Part

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 value. 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) 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.

2) Extracting the Trend Part: The *Trend Part* consist 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.

a) 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 meta-knowledge 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.

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

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. Once a burst is found, it is added to the root *Sequence* and then calibrated to match the arrival rate from the trace. Finding a burst requires the arrival rate in the trace to differ significantly from the predicted value based on the *Seasonal* and *Trend* parts. In order to eliminate false positives due to *Seasonal Parts* that are offset time-wise, the *Seasonal Part* used for the reference model in the burst recognition activity differs from the actual extracted *Seasonal Part*. The difference is that the *Seasonal Part* used in the 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 time-wise offset seasonal periods are commonly observed.

Data: duration: seasonal period duration,
LIST: list of tuples $\vec{t} = \begin{pmatrix} arrivalRate \\ timeStamp \end{pmatrix}$,
MAX: list of local maxima in LIST,
trendSequence: root *Sequence* of all *Trend* segments;

Function `extractTrendPart()`
  largestPeakOffset ← offset of peak with largest arrival rate within a seasonal iteration;
  largestPeakArrivalRate ← arrival rate of peak with largest arrival rate within a seasonal iteration;
  iterations ← LIST.$lastTuple.timeStamp$/duration;
  for $i \leftarrow 0$ **to** iterations **do**
    a ← `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 ← 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 $\vec{t} = \begin{pmatrix} arrivalRate \\ timeStamp \end{pmatrix} \in \mathbf{L}$ with minimal $d \leftarrow |\mathbf{L}.timeStamp - time|$;
end

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

4) Extracting the Noise Part: The *Noise Part* extraction consists of two steps: Noise reduction and the calculation fo 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 VII). 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.

a) 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 [4]):

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

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** $|\text{LIST}| - 1$ **do**
    difference[i] ← LIST [i].arrivalRate - FILTERED_LIST [i].arrivalRate;
  end
  distribution ← normal distribution with `mean(`difference`)` and `standardDistribution(`difference`)`;
end

Algorithm 3: Calculating the *Noise* distribution.

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:

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

b) 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.

B. 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.

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 counter-part.

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.

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 *inter burst 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 *inter burst period* from the independent bursts. The burst arrival rate is also calculated as the median burst arrival rate.

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, the minimum and maximum value 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.

VII. Model Accuracy Evaluation

We evaluate the presented model extraction methods based on nine different real-world Web server traces covering between two weeks and seven months. The traces all are strongly influenced by 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 relative median errors.

s-DLIM and hl-DLIM extraction are applied to extract model instances for all traces. For these extraction methods we also 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 reasons, *ExponentialIncreaseAndDecline* is always selected for *Burst* modeling (it is a child of *Burst* in the DLIM meta-model). *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 one 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 3 and 4. Additionally, it extracts a bi-weekly 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 evaluate the model extraction accuracy by computing the relative errors for each pair of corresponding entries in the extracted model instance and trace. The median of the relative error values is presented in this paper. The mean relative error is prone to deflection by positive outliers. Moreover, we compare the extraction error and run-time on commodity hardware (Core i7 4770, 16 GB RAM) against the BFAST time-series decomposition [17] (which returns split data as opposed to a descriptive model). To enable a fair comparison, we configured BFAST to extract one seasonal pattern and not more than one trend per day. In contrast to DLIM, where seasonal patterns are represented by piece-wise interpolating functions, in BFAST's output, the seasonal pattern is represented as a less compact discrete function.

A. Internet Traffic Archive and BibSonomy Traces

The first batch of traces was retrieved from The Internet Traffic Archive[3]. The Internet Traffic Archive includes the following traces: *ClarkNet-HTTP* (Internet provider WWW server), *NASA-HTTP* (Kennedy Space Center WWW server), *Saskatchewan-HTTP* (University WWW server), and *World-Cup98* (official World Cup 98 WWW servers). Additionally, we used a six week long trace of access times to the social bookmarking system *BibSonomy* [3], beginning on May 1st 2011[4]. All traces were parsed to arrival rate traces with a quarter-hourly resolution (96 arrival rate samples per day).

Table I shows the relative median errors for s-DLIM, p-DLIM, and the hl-DLIM extraction for different configurations. It also displays run-time of the overall most accurate extraction configuration (s-DLIM, ignoring noise, trend length 1) as an average value over ten runs. In cases in which BFAST decomposition terminated, errors and run-times are also displayed for BFAST. 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 2700 and 2900. The extraction methods do not have contingencies for this case.
- Deviating Seasonal Patters 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

[3]Internet Traffic Archive: `http://ita.ee.lbl.gov/`

[4]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. **Saskatchewan**	4. **WorldCup98**	5. **BibSonomy**
Extraction Parameters	**relative median (%)**	**relative median (%)**	**relative median (%)**	**relative median (%)**	**relative median (%)**
p-DLIM, noise extracted	too short	32.223	43.293	52.304	37.387
p-DLIM, noise eliminated	too short	28.944	35.831	53.316	35.378
p-DLIM, noise ignored	too short	23.633	35.663	53.495	36.264
s-DLIM, Trend length 1, noise extracted	21.195	26.446	35.551	19.735	26.988
s-DLIM, Trend length 1, noise eliminated	17.509	23.56	**26.492**	16.882	**21.479**
s-DLIM, Trend length 1, noise ignored	**12.409**	**18.812**	29.171	**12.979**	23.831
s-DLIM, Trend length 2, noise ignored	14.734	20.8	30.273	15.691	26.786
s-DLIM, Trend length 3, noise ignored	14.919	27.577	32.085	19.161	28.218
hl-DLIM, Trend length 1, noise extracted	20.105	26.541	37.942	16.093	27.513
hl-DLIM, Trend length 1, noise eliminated	19.361	24.539	33.24	15.66	25.433
hl-DLIM, Trend length 1, noise ignored	72.924	55.575	80.792	43.957	42.268
BFAST	12.243	no result	no result	no result	no result
average s-DLIM run-time (ms)	4.2	25.2	118.8	11.8	125
average BFAST run-time (ms)	76276	no result	no result	no result	no result

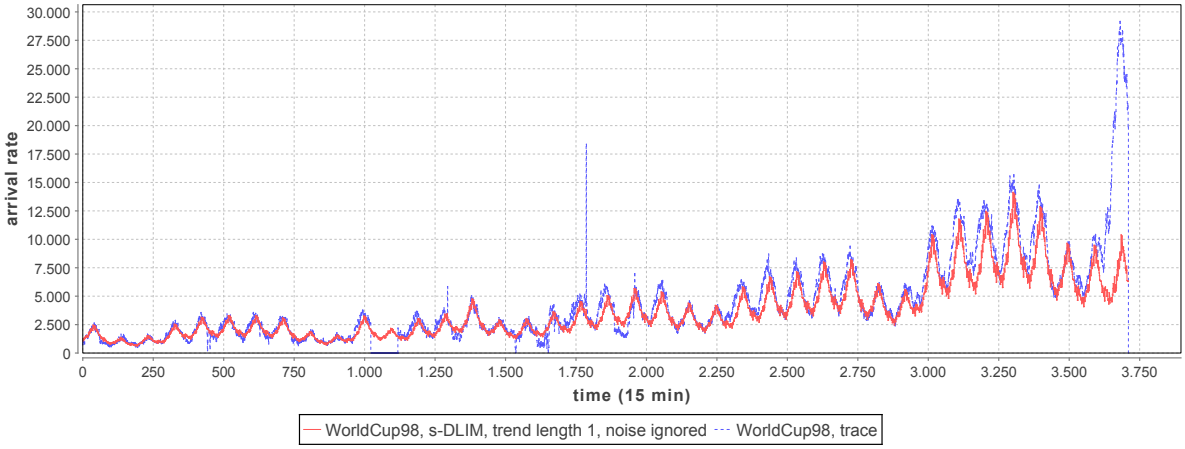

Fig. 4. Arrival rates of the original WorldCup98 trace (blue) and the extracted DLIM instance (red) using s-DLIM with a *Trend* length of 1 and ignoring noise.

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 meta-knowledge, such as calendar information, to the implementation of seasonal break detection, as introduced in BFAST [17].

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* 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 then 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 Fig. 5). 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-DLIMs 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 relatively accurate. Even for the Saskatchewan trace, p-DLIM performs relatively well when compared to s-DLIM.

The hl-DLIM extraction shows an entirely different picture. Considering that hl-DLIM uses only a small number of pre-defined 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

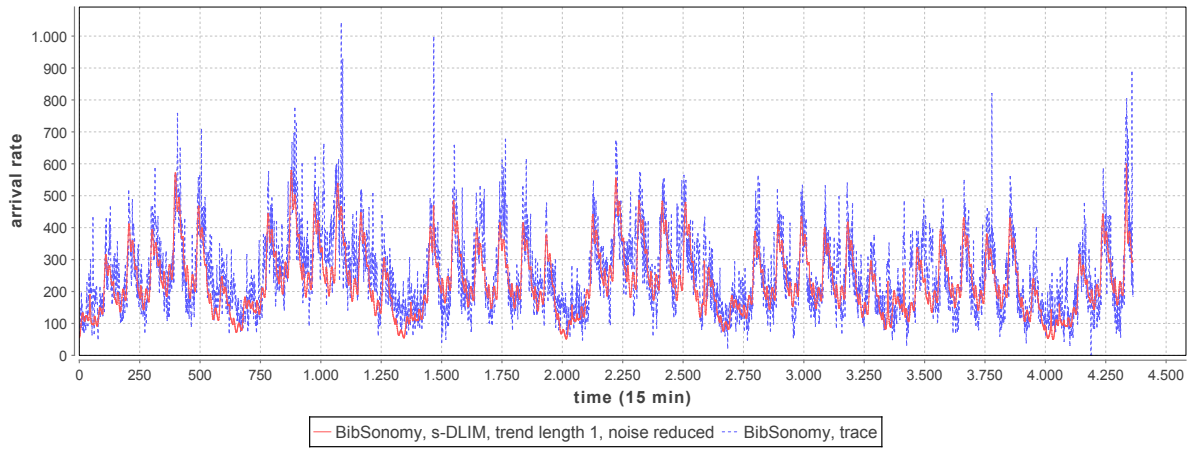

Fig. 5. Arrival rates of the original BibSonomy trace (blue) and the extracted DLIM instance (red) using s-DLIM with *Trend* length 1 and noise reduction.

extracting two peaks), the non-filtered 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.

Comparing the accuracy of our extraction methods with that of BFAST proves difficult, given that, for four of the five considered traces, BFAST did not terminate within 1.5 hours, which would make BFAST execution at least 45000 time slower than s-DLIM in these cases. However, the ClarkNet trace extraction, shows that our extraction methods exhibit accuracy comparable to the accuracy of BFAST in cases where BFAST terminates in a reasonable amount of time. To eliminate the possibility of BFAST not terminating due to configuration errors on our side, we ran the same configuration on shortened versions of the respective traces. BFAST's time-series analysis of these shortened traces terminated, however, the latter are too short to meet our criteria of sufficiently long traces with recurring seasonal patterns and trends.

B. Wikipedia Traces

The second batch of traces was retrieved from the Wikipedia page view statistics[5]. They were parsed from the *project-count* 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 4th 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 vs. global access patterns.

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 relative load variations in general. As a result, all modeling errors are also relatively small.

In terms of accuracy, our extraction processes perform as well as the BFAST decompositions. s-DLIM performs better than BFAST for both the German and French Wikipedia traces. Here, s-DLIM's accuracy profits from its support of multiplicative trends. BFAST does, however, provide better accuracy for the English and Russian traces. In these cases BFAST's sophisticated trend calibration mechanisms outperform s-DLIM. s-DLIM is, however, significantly faster than BFAST. Running on the same machine, LIMBO's s-DLIM implementation performed on average 8354 times faster than BFAST's R implementation and returned results in all cases in less than 0.2 seconds.

[5]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**
Extraction Parameters	**relative median (%)**	**relative median (%)**	**relative median (%)**	**relative median (%)**
s-DLIM, Trend length 1, noise extracted	11.215	10.472	9.964	7.764
s-DLIM, Trend length 1, noise eliminated	10.511	8.566	**9.912**	7.838
s-DLIM, Trend length 1, noise ignored	**8.538**	**7.6**	11.251	**4.855**
s-DLIM, Trend length 2, noise ignored	9.956	8.973	11.683	5.27
s-DLIM, Trend length 3, noise ignored	11.771	9.813	11.42	7.23
hl-DLIM, Trend length 1, noise extracted	11.898	8.503	12.392	7.75
hl-DLIM, Trend length 1, noise eliminated 1	11.393	8.373	12.496	7.961
hl-DLIM, Trend length 1, noise ignored	13.126	10.816	13.31	8.868
BFAST	11.223	8.511	5.809	2.302
average s-DLIM run-time (ms)	3.9	3.5	5.8	3.2
average BFAST run-time (ms)	23518	23630	23803	21517

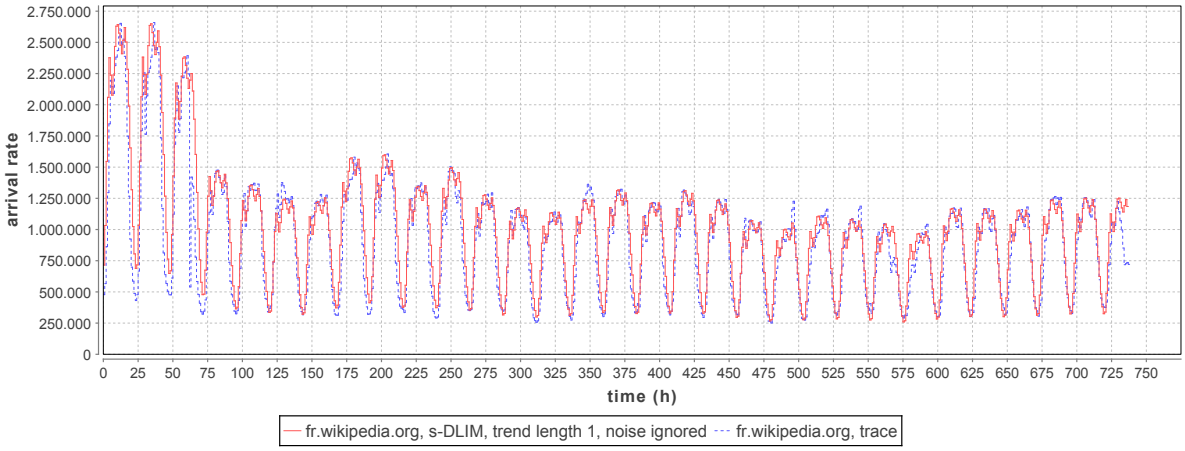

Fig. 6. Arrival rates of the original French Wikipedia trace (blue) and the extracted DLIM instance (red) using s-DLIM with *Trend* length 1 and ignoring noise.

VIII. Conclusions

This paper presents the Descartes Load Intensity Model (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's use 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 good accuracy from nine different real-world load intensity traces. The model extraction performs best for the Wikipedia traces. Extracted *Seasonal* patterns match the trace's days well and the overlaying 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. In addition, the results show that DLIM itself is capable of accurately capturing real-world load intensity profiles, independent of the explicit extraction processes we introduce.

As part of future work on LIMBO and the model instance extraction methods, 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 relative 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.

IX. Acknowledgments

This research has been supported by the Research Group[6] of the Standard Performance Evaluation Corporation (SPEC)[7].

[6]SPEC Research: `http://research.spec.org`
[7]SPEC: `http://www.spec.org`

REFERENCES

[1] P. Barford and M. Crovella. 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, pages 151–160, New York, NY, USA, 1998. ACM.

[2] A. Beitch, B. Liu, T. Yung, R. Griffith, A. Fox, and D. A. Patterson. Rain: A workload generation toolkit for cloud computing applications. Technical Report UCB/EECS-2010-14, EECS Department, University of California, Berkeley, Feb 2010.

[3] D. Benz, A. Hotho, R. Jäschke, B. Krause, F. Mitzlaff, C. Schmitz, and G. Stumme. The social bookmark and publication management system BibSonomy. *The VLDB Journal*, 19(6):849–875, Dec. 2010.

[4] H. J. Blinchikoff and A. I. Zverev. *Filtering in the time and frequency domains*. Krieger Publishing Co., Inc., 1986.

[5] G. Casale, A. Kalbasi, D. Krishnamurthy, and J. Rolia. Burn: Enabling workload burstiness in customized service benchmarks. *IEEE Transactions on Software Engineering*, 38(4):778–793, 2012.

[6] D. Feitelson. Workload modeling for performance evaluation. In M. Calzarossa and S. Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science*, pages 114–141. Springer Berlin Heidelberg, 2002.

[7] J. H. Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[8] E. H. Halili. *Apache JMeter: A Practical Beginner's Guide to Automated Testing and performance measurement for your websites*. Packt Publishing Ltd, 2008.

[9] M. Kuhn, S. Weston, C. Keefer, and N. Coulter. Cubist models for regression, 2012. http://cran.r-project.org/web/packages/Cubist/vignettes/cubist.pdf, Last accessed: Oct 2014.

[10] H. Li. Realistic workload modeling and its performance impacts in large-scale escience grids. *Parallel and Distributed Systems, IEEE Transactions on*, 21(4):480–493, 2010.

[11] D. A. Menascé, V. A. F. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira, Jr. A hierarchical and multiscale approach to analyze e-business workloads. *Perform. Eval.*, 54(1):33–57, Sept. 2003.

[12] J. R. Quinlan et al. Learning with continuous classes. In *Proceedings of the 5th Australian joint Conference on Artificial Intelligence*, volume 92, pages 343–348. Singapore, 1992.

[13] A. Reyes-Lecuona, E. González-Parada, E. Casilari, J. Casasola, and A. Diaz-Estrella. A page-oriented www traffic model for wireless system simulations. In *Proceedings ITC*, volume 16, pages 1271–1280, 1999.

[14] S. Roy, T. Begin, and P. Goncalves. A complete framework for modelling and generating workload volatility of a vod system. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, pages 1168–1174, 2013.

[15] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: a cautionary tale. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 18–18, Berkeley, CA, USA, 2006. USENIX Association.

[16] A. van Hoorn, M. Rohr, and W. Hasselbring. 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, pages 124–143, Berlin, Heidelberg, 2008. Springer-Verlag.

[17] J. Verbesselt, R. Hyndman, G. Newnham, and D. Culvenor. Detecting trend and seasonal changes in satellite image time series. *Remote Sensing of Environment*, 114(1):106 – 115, 2010.

[18] J. G. von Kistowski, N. R. Herbst, and S. Kounev. LIMBO: A Tool For Modeling Variable Load Intensities. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM, March 2014.

[19] J. G. von Kistowski, N. R. Herbst, and S. Kounev. 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, March 2014.

[20] N. Zakay and D. G. Feitelson. Workload resampling for performance evaluation of parallel job schedulers. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 149–160, New York, NY, USA, 2013. ACM.