Using and Extending LIMBO for Descriptive Modeling of Arrival Behaviors

Jóakim v. Kistowski, Nikolas Herbst, Samuel Kounev

Chair for Computer Science II, Software Engineering
University of Würzburg
joakim.kistowski@uni-wuerzburg.de
nikolas.herbst@uni-wuerzburg.de
samuel.kounev@uni-wuerzburg.de

Abstract: LIMBO is a tool for the creation of load profiles with variable intensity over time both from scratch and from existing data. Primarily, LIMBO's intended use is the description of load arrival behaviours in open workloads. Specifically, LIMBO can be employed for the creation of custom request or user arrival time-stamps or for the re-parameterization of existing traces.

LIMBO bases on the Descartes Load Intensity Model (DLIM) for the formalized description of its load intensity profiles. The DLIM formalism can be understood as a structure for piece-wise defined and combined mathematical functions. We outline DLIM and its elements and demonstrate its integration within LIMBO.

LIMBO is capable of generating request or user arrival time stamps from DLIM instances. In a next step, these generated time-stamps can be used for both open workload based benchmarking and simulations. The TimestampTimer plug-in for JMeter already allows the former. LIMBO also offers a range of tools for easy load intensity modeling and analysis, including, but not limited to, a visual decomposition of load intensity time-series into seasonal and trend parts, a simplified load intensity model as part of a model creation wizard, and an automated model instance extractor.

As part of our LIMBO tutorial, we explain these features in detail. We demonstrate common use cases for LIMBO and show how they are supported. We also focus on LIMBOs extensible architecture and discuss how to use LIMBO as part of another tool-chain.

1 Introduction

Today's cloud and web-based IT services need to handle huge amounts of concurrent users. Customers access services independently of one another and expect reliable quality-of-service under highly variable and dynamic load intensities. In this context, any knowledge about a service's load intensity profile is becoming a crucial information for managing the underlying IT resource landscape. Load profiles with large amounts of concurrent users are typically strongly influenced by human habits, trends, and events. This includes strong deterministic factors such as time of the day, day of the week, common working hours and planned events.

Common benchmarking frameworks such as Faban[1], Rain [BLY+10], and JMeter [Hal08] allow job injection rates to be configured either to constant values, stepwise increasing rates (e.g., for stress tests), or rates based on recorded workload traces. The tool we present in this document aims at closing the gap between highly dynamic load intensity profiles observed in real life and the current lack of support for flexible handling of variable load intensities in benchmarking frameworks.

In [vKHK14b], we introduce two modeling formalisms at different abstraction levels: At the lower abstraction level, the *Descartes Load Intensity Model* (DLIM) offers a structured and accessible way of describing the load intensity over time by editing and combining mathematical functions. The *high-level DLIM* (hl-DLIM) allows the description of load variations using few defined parameters that characterize the seasonal patterns, trends, as well as bursts and noise elements. An example load profile consisting out of a seasonal part, a trend and bursts is presented in Fig. 1.

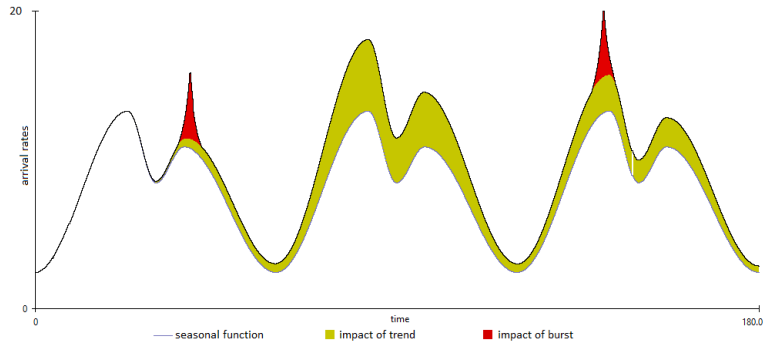

Figure 1: Example profile with a seasonal pattern, trend, and bursts.

In this document, we present LIMBO[2] [vKHK14a] - an Eclipse-based tool for handling and instantiating load intensity models based on DLIM. LIMBO offers an accessible way of editing DLIM instances and extracting them from existing traces. It also supports using hl-DLIM parameters for easy creation of new DLIM instances through a model creation wizard. In addition, we provide the *TimestampTimer* plugin for JMeter [Hal08], which enables the use of LIMBO-generated time stamps for the definition of JMeter work unit start times. We also provide extensive documentation for LIMBO on our website[2], including a thorough tutorial. In this document, we provide a description of the LIMBO tool and its architecture. We demonstrate LIMBO's extensibility and its use as part of other code projects.

[1] Faban `http://faban.org`
[2] LIMBO `http://go.uni-wuerzburg.de/limbo`

2 Definition of Load Intensity

In the context of LIMBO, *load intensity* is a discrete function describing *arrival rates* of workload units (e.g. users, sessions or requests) over time. We assume that the work units are of a homogeneous type and 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 all *work units* u_{t_0}, with their respective *arrival time* t_0, that have arrived up until time t.

3 Use-Cases for LIMBO

LIMBO can be used in many contexts, some of which might not be directly apparent. The following sections describe a few use-case scenarios in which the ability to create and modify a load intensity model is extremely helpful. This is intended to also demonstrate the usefulness of load intensity modeling in general and to give the reader a few additional ideas on what to do with LIMBO.

3.1 Core Use-Cases

LIMBO was created with the specific goal of being used as part of the following use cases:

- Creation of artificial load intensity profiles for specific benchmarking purposes
- Extraction of existing load intensity profiles from pre-existing traces.

LIMBO features an extensible architecture, which allows LIMBO's application beyond these core use cases, e.g.:

1. **Creating artificial Load Intensity Profiles for Benchmarking**

   A model describing the load intensity variations over time can be used to create request or user arrival time-stamps that can then be used to define the beginning time of a unit of work within a benchmarking framework. This enables a user to use a multitude of different varying workloads, which in turn helps with the benchmarking of system properties that deal with such variations (such as elasticity) [WHGK14].

   This use-case describes the possibility of a custom created load intensity profile, that has been specifically designed to help with the benchmarking of such a property. Of course, this load intensity profile may be subject to additional requirements, such as representability.

2. **Creating a Load Intensity Model Instance from an existing Trace**

   A model instance can be used to describe a past real-world load profile (within a certain error). This opens up a number of sub-use-cases:

   - *Parametrization of Request / User Arrival Traces*

     Among others, Zakay et al. [ZF13] sample request traces for benchmark workload generation. When doing so several problems can arise. The trace might be taken from a system that is magnitudes larger than the test system on which the benchmark is to be executed. The trace might also have been taken over a long time period and has to be temporally compressed for the benchmark. When using a load intensity model instead of a simple trace, these problems become easily manageable. They can be managed either by modifying the model instance directly, or through parametrization of the request time-stamp generation.

   - *Anonymization of Request / User Arrival Traces*

     Request traces of real cloud or web based systems often include additional information that may contain information about the system's users. Even the exact time-stamps themselves may still provide a reader of the trace with the ability to extract information about the behavior of single users.

     An abstract load variation representation helps to minimize this problem. System providers, who are concerned about customer anonymity, might be more likely to provide usage information for research purposes in an abstract form as made possible by a load intensity model.

3.2 Additional Use-Cases

These additional use-cases are either derived from the two previous cases or constitute new approaches to LIMBO's Load Intensity Model. They describe more complex scenarios, in which the features provided by LIMBO can play a central part.

1. **Load Intensity Forecasting**

   A model instance that has been derived from the incoming request trace of a currently running system can be used to predict future request intensity variations. Doing so can also help with the detection of unplanned events that deviate from an extrapolated periodic model. Such a forecasting mechanism could be deployed on cloud systems. In that context it would help to improve dynamic resource management, by increasing the efficiency of elastic resource re-allocation.

2. **Anomaly Detection**

   A calibrated load intensity model instance could serve as a baseline allowing the computation of anomaly metrics. This approach incorporating DLIM model instances as baseline will most presumably end up with a higher anomaly detection

accuracy and less false positives [Bie12]. Such a baseline can also be used in other fields of computer science, since it can always be used for comparison against anomalies. In a security context, it might be used for finding access patterns that deviate from usual access patterns in the form of their load intensity as part of an intrusion detection benchmark as proposed in [MK12].

4 LIMBO

LIMBO allows editing of load intensity models based on DLIM and supports guided model creation using the parameters defined in hl-DLIM.

4.1 Models

- **Descartes Load Intensity Model**: DLIM describes request arrival rates over time and offers a way to define a piece-wise mathematical function for the approximation of variable arrival rates with support for (partial) periodicity, flexibility and composability.
- **High-level DLIM**: hl-DLIM offers abstracted knowledge about load intensity variations modeled through a limited number of workload parameters. Inspired by the time series decomposition approach in BFAST [VHNC10], a hl-DLIM instance describes a *Seasonal* and *Trend* part. Additionally, it features a *Burst* and *Noise* part.

4.2 Features

LIMBO offers a significant number of different features, all targeted at enabling easy and comprehensive creation and modification of load intensity profiles. LIMBO has been implemented using an extensible architecture. It is thus open for extension with additional features. At this time, LIMBO's major features are:

1. **Creation of new load intensity profiles using hl-DLIM**: LIMBO enables the use of hl-DLIM parameters for easy creation of new DLIM instances through a model creation wizard.
2. **Modification of DLIM load profiles**: LIMBO allows for modification of DLIM load profiles, by adding, removing, and modifying the piece-wise mathematical functions of which these profiles are composed.
3. **Visualization of load profiles**: LIMBO includes a graphical view for the display of DLIM instances. This view also contains a more detailed visualization feature, which decomposes DLIM instances into their seasonal parts, trends, and bursts. It then displays each part's contribution towards the total load intensity.

4. **Timestamp generation**: Load intensity profiles can be used to generate request or user arrival time stamps. These time stamps can be used as input for common benchmarking frameworks, such as JMeter [Hal08]. LIMBO's extensible architecture also allows for easy addition of additional time-stamp exporters for other benchmarking frameworks, such as FinCos [MBM13].

5. **Timestamp use for load generation**: We provide the *TimestampTimer* plugin for JMeter. This plugin allows the use of LIMBO-generated time-stamps for the definition of work unit start times.

6. **Model instance extraction**: DLIM instances can be extracted from existing arrival rate traces using one of the implemented model instance extractors:

   - *Simple DLIM Extraction Process (s-DLIM)*: An accurate extraction process, which extracts DLIM instances from existing arrival rate traces. Our evaluation of s-DLIM accuracy in [vK14] shows a median extraction error of 12.4%. Comparison with BFAST[VHNC10] also shows, that s-DLIM provides excellent performance, with all extractions completing in less than 0.2 seconds and providing an average speedup of 8354 compared to BFAST decomposition.
   - *Periodic DLIM Extraction Process (p-DLIM)*: A less accurate extraction process to extract DLIM instances from existing arrival rate traces. Other than s-DLIM, p-DLIM instances are intended to be repeated for load intensity forecasting.
   - *high-level DLIM Extraction Process*: Extracts hl-DLIM instances from existing arrival rate traces.

   LIMBO's extensible architecture allows furhter model extraction and calibration methods to be integraded, as well as for reading other file formats, such as trace files as exported by Kieker [vHWH12].

4.3 Implementation

LIMBO, the tooling for DLIM and hl-DLIM models, is realized as a plug-in for the Eclipse IDE. It provides an editor for the creation and modification of model instances, as well as additional utilities for using the created models. Using DLIM's EMF-generated code base as a basis, the following features have been implemented:

- **Model Evaluation**: Support for the DLIM function output calculation and manual refinement of model instances.

- **Modeling Process Assistance**: Includes an automated process for the creation and extraction of DLIM instances. Additionally, LIMBO provides a model instantiation guidance by means of a wizard.

- **Utilities**: Additional functionality is provided for existing DLIM instances. Including functionality for the generation of arrival rate series from a time-stamp series,

and a tool that calculates the difference between an arrival rate trace and a model instance.

LIMBO consists of five individual plug-ins as visualized in Fig. 2. Note that all packages and plug-ins begin with the prefix "*tools.descartes.dlim*". For better readability "*tools.descartes*" is omitted for the remainder of this paper.

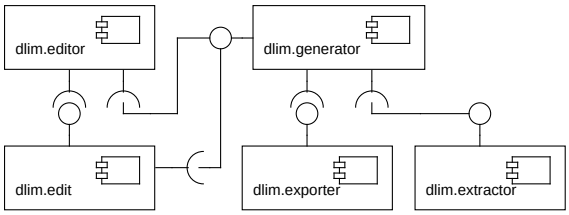

Figure 2: LIMBO architecture.

1. **DLIM Generator** The *dlim.generator* plug-in contains the DLIM element interfaces and implementations, as well as their default utilities (e.g., for validation). It also contains model evaluation tools, as well as arrival rate and time-stamp series generators. It features two extension points:

   - **Exporter** extension points supports custom implementations by implementing the *dlim.exporter.IDlimExporter* interface. Default exporters are contained in the *dlim.exporter* plug-in.
   - **Extractor** extension point allows the addition of extractors for deriving a model instance from an existing trace. Extractors must implement *dlim.reader.IDlimArrivalRateReader* for their trace parser and *dlim.extractor.IDlimExtractor* for the model instance creator. Default extractors are contained in the *dlim.extractor* plug-in.

   The *dlim.generator* plug-in also provides LIMBO's core functionalities for use as part of other projects. The most important provided packages are:

   - **dlim**: This package contains the model element interfaces. It is generated by the EMF genmodel, but has been modified to return a *CustomDlimFactoryImpl* Instance for the *DlimFactory.eINSTANCE*, instead of the generated *DlimFactoryImpl*.
   - **dlim.generator**: This package contains the model evaluation logic, primarily used for arrival rate and time-stamp series generation. The **ModelEvaluator** class, specifically, is the primary access point to all model evaluation logic. It is instantiated using the model's root element (which is always a *Sequence*) and a seed for the random number generator (for *Noise* evaluation). It provides the **getArrivalRateAtTime(double rootTime)** method, which returns the model's resulting arrival rate for a given time.

- **dlim.exporter.utils**

  This package contains utilities that help when implementing a new exporter. The use of these utilities is highly recommended.

  - **ArrivalRateGenerator**: Provides functionality to sample a list of arrival rates from a DLIM instance, represented by its *dlim.generator.ModelEvaluator*.
  - **TimeStampWriter**: Provides functionality to generate a list of request time stamps using a list of arrival rates (as is provided by *ArrivalRateGenerator*.

- **dlim.reader**

  This package contains classes and interfaces responsible for the parsing of time series.

  - **ArrivalRateReader**: Provides functionality to read arrival rates from an arrival rate file. Can read either a single arrival rate at a given time, or returns a list of all *ArrivalRateTuples* contained in the file.
  - **IDlimArrivalRateReader**: Interface for an arrival rate reader. Must be implemented by a reader for the *Extractor* extension point.
  - **DefaultArrivalRateReader**: A default implementation of *IDlimArrivalRateReader*. Is able to read arrival rate files of the same format as produced by the arrival rate file exporter.
  - **RequestTimeSeriesReader**: Provides functionality to parse a request time-stamp trace into an file containing the arrival rates per second for each second.

2. **DLIM Generator Edit**

   This plug-in contains the providers used by the editor, which provide display specific information, such as the display images and labels of model elements.

3. **DLIM Generator Editor**

   The *dlim.editor* plug-in contains all GUI elements and their utilities. It also contains implicit modeling process knowledge in its GUI.

4. **DLIM Exporter**

   The *dlim.exporter* plugin offers three default implementations of the *dlim.generator* plugin's *dlim.exporter. IDlimExporter* interface and the *exporter* extension point:

   - **DlimArrivalRateExporter**: Exports an arrival rate time series.
   - **DlimEqualDistanceRequestStampExporter**: Exports request time stamps with an equal distance from one another within each sampled arrival rate interval.
   - **DlimUniformRequestStampExporter**: Exports request time stamps with a uniform random sampling within each sampled arrival rate interval.

5. **DLIM Extractor**

   The *dlim.extractor* plug-in offers two default implementations of the *dlim.extractor.IDlimExtractor* interface and the *extractor* extension point:

   - **PeriodicProcessExtractor**: Extracts a DLIM instance based on the Periodic DLIM Extraction Processes (p-DLIM).
   - **SimpleProcessExtractor**: Extracts a DLIM instance based on the Simple DLIM extraction process (s-DLIM).

   Both *extractor* extension point implementations in this plug-in use the provided default *dlim.reader.ArrivalRateReader* provided by the *dlim.generator* plug-in.

5 Conclusions

This paper provides a detailed description of LIMBO: A toolkit for creating and editing of DLIM instances. By enabling the flexible handling of load intensity profiles, LIMBO addresses a strong need in the areas of benchmarking and elastic capacity management. We describe the features of LIMBO and summarize the use cases and fields of possible application. We also provide a detailed description of LIMBO's architecture with a focus on extension points and functionality provided for use in other projects.

References

[Bie12] Tillmann Carlos Bielefeld. Online performance anomaly detection for large-scale software systems, 2012.

[BLY+10] Aaron Beitch, Brandon Liu, Timothy Yung, Rean Griffith, Armando Fox, and David 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.

[Hal08] Emily H Halili. *Apache JMeter: A Practical Beginner's Guide to Automated Testing and performance measurement for your websites*. Packt Publishing Ltd, 2008.

[MBM13] Marcelo R.N. Mendes, Pedro Bizarro, and Paulo Marques. FINCoS: Benchmark Tools for Event Processing Systems. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 431–432, New York, NY, USA, 2013. ACM.

[MK12] Aleksandar Milenkoski and Samuel Kounev. Towards Benchmarking Intrusion Detection Systems for Virtualized Cloud Environments. In *Proceedings of the 7th International Conference for Internet Technology and Secured Transactions (ICITST 2012)*, pages 562–563, New York, USA, December 2012. IEEE.

[VHNC10] Jan Verbesselt, Rob Hyndman, Glenn Newnham, and Darius Culvenor. Detecting trend and seasonal changes in satellite image time series. *Remote Sensing of Environment*, 114(1):106 – 115, 2010.

[vHWH12] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 247–248, New York, NY, USA, 2012. ACM.

[vK14] Jóakim v. Kistowski. Master's Thesis: Modeling Variatons in Load Intensity Profiles, March 2014.

[vKHK14a] Jóakim Gunnarson von Kistowski, Nikolas Roman Herbst, and Samuel Kounev. LIMBO: A Tool For Modeling Variable Load Intensities. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*, ICPE '14, pages 225–226, New York, NY, USA, March 2014. ACM.

[vKHK14b] Jóakim Gunnarson von Kistowski, Nikolas Roman Herbst, and Samuel 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)*, pages 1–4, New York, NY, USA, March 2014. ACM.

[WHGK14] Andreas Weber, Nikolas Roman Herbst, Henning Groenda, and Samuel Kounev. Towards a Resource Elasticity Benchmark for Cloud Environments. In *Proceedings of the 2nd International Workshop on Hot Topics in Cloud Service Scalability (HotTopiCS 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM, March 2014.

[ZF13] Netanel Zakay and Dror 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.