Extending the OMNeT++ Sequence Chart for Supporting Parallel Simulations in Horizon

Georg Kunz*, Simon Tenbusch‡, James Gross‡, Klaus Wehrle*
*Communication and Distributed Systems, ‡Mobile Network Performance
RWTH Aachen University
*lastname@comsys.rwth-aachen.de, ‡lastname@umic.rwth-aachen.de

ABSTRACT

Developing parallel network simulations is a complex task. Besides getting the model right, developers of parallel simulations are striving for an additional design goal: Performance. We argue that developers need an insight into the behavior of a simulation model in order to assess and optimize its parallel performance. Specifically, given a parallel simulation model, it is imperative to identify and eliminate performance bottlenecks. To this end, we extend the sequence chart provided by the Eclipse-IDE of OMNeT++ with specific functionality to visualize, analyze, and optimize the performance of parallel simulations in the context of our OMNeT++-based parallel simulation framework HORIZON. This extended abstract presents the features and modifications of our code contribution.

1. INTRODUCTION

Visualizing event interactions has proven to be a valuable aid during the development of simulation models. For this reason, OMNeT++ provides two graphical user interfaces (GUIs), each targeting a specific purpose. The first GUI, a Tcl/Tk-based runtime visualizer, illustrates event interactions and allows for inspecting the state of the simulation at runtime. The second GUI is part of the Eclipse-based integrated development environment (IDE). It constitutes a sequence chart showing the execution order and successor relationship among events. In contrast to the runtime visualizer, this GUI is an offline tool that reads and visualizes traces generated at runtime. Despite useful for verifying the correctness of sequential simulations, both GUIs do not provide specific development support for optimizing the performance of parallel simulations.

In previous work, we developed HORIZON, an extension of OMNeT++ which allows for a parallel execution of HORIZON-enabled simulations on multi-processor machines. In the context of this work, we observed the strong need for a graphical support tool that helps developers in tuning parallel simulations. Hence, we extended the sequence chart of the Eclipse-IDE with functionality for analyzing and optimizing the performance of parallel simulations. In the following, we first briefly introduce the basics of HORIZON before presenting the functionality of our code contribution.

2. HORIZON

HORIZON enables a parallel execution of discrete event simulations on multi-processor computers by means of a novel modeling paradigm. Specifically, this paradigm expands discrete events with *durations* in simulated time to explicitly and naturally model the (processing) *delays* of physical systems, e. g., decoding packets, routing table lookups, etc. Based on this paradigm, HORIZON defines a conservative parallelization scheme that exploits the given event durations to determine independent events for parallel execution. According to this scheme, two overlapping expanded events are considered to be independent and hence processed in parallel. The rational behind this is that the result of a physical process (e. g., decoding a packet) is only available after its completion (i. e., packet completely decoded). Hence, if two expanded events overlap, the input of both events cannot depend on their mutual output as the output is not yet complete when the events begin.

3. EXTENDED SEQUENCE CHART

In order to assess the performance of an HORIZON-based parallel simulation model, we need to analyze three properties of the model:

i) the successor relationship among events,
ii) the duration of each event in simulated time, and
iii) the processing time in wall-clock time it takes to compute each event on a CPU.

The first property is already part of the functionality of the sequence chart provided by OMNeT++. In fact, visualizing the successor relationship among events constitutes the main purpose of the stock sequence chart. The other two properties, however, are only relevant in the context of HORIZON:

Event durations: The conservative synchronization scheme of HORIZON utilizes event durations in order to determine which events are safe for parallel processing. Hence, the extended sequence chart needs to illustrate the duration of each event to enable developers to visually assess the set of parallelizable events. If this set is small, it constitutes a performance bottleneck in the model.

Processing times: In addition to the event durations, the processing time of each event is relevant. Typically, only events of non-trivial computational complexity are worth parallelizing since they dominate the runtime of the simulation. In contrast, the synchronization overhead involved in parallelization often out-weights the potential performance gain for computationally simple events. Consequently, the extended sequence chart needs to visualize the computational complexity of each event to allow developers to distinguish computationally relevant from irrelevant events.

As a result, our extension of the sequence chart needs to display two time domains: the simulated time domain (event durations) and the wall-clock time domain (event processing times). In both cases, the straightforward approach is to expand the illustration of an event to span a period of time in the sequence chart. In our current implementation, we

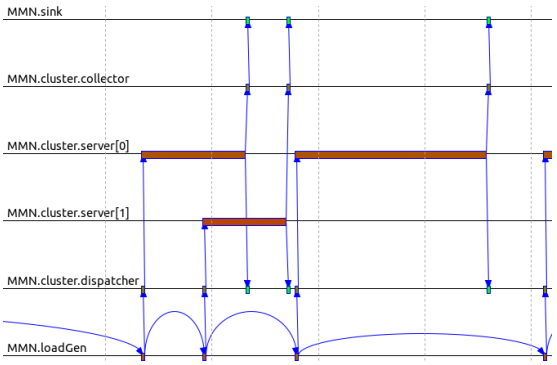

Figure 1: Screenshot showing the duration of events illustrated by the length of the "event-block" while event complexities are color-coded. Note the difference in complexity: Events on the server and loadGen modules are more complex (darker) than events on the dispatcher or sink (lighter).

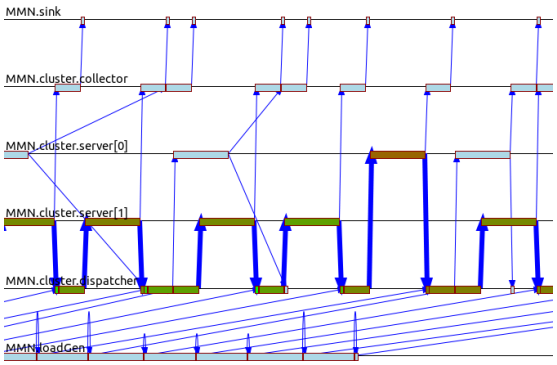

Figure 2: Screenshot showing the critical path (colored events connected by big blue arrows). In this view, the horizontal extent of an event corresponds to its computational complexity. Events on the critical path that overlap with few other events are colored reddish, while events with a high amount of overlapping complexity are colored greenish.

only utilize the x-axis for visualizing one of the time spans. Hence, users can choose whether the "length" of an event indicates either its duration or its processing time. Depending on the selection, we visualize the respective other time domain by means of a color scheme gradually ranging from green to red for increasing values. Figure 1 shows an example based on a simple MMN queueing model. Future versions of the sequence chart might make use of the y-axis to indicate the second time domain.

Usage. As for the stock version of the sequence chart, we need to trace a simulation run to obtain all relevant data for the visualization process. Although we target parallel simulations, this trace run is currently performed sequentially. This is mainly for keeping the required changes to the existing implementation simple and to ensure a correct sequential ordering of entries in the trace file. Moreover, since writing to a single trace file is an inherent bottleneck, parallel performance while tracing is expected to be low anyway.

Once a trace is complete, developers can load it in the IDE and begin their analysis. To aid this analysis process, the extended sequence chart provides supporting features in addition to illustrating the duration and complexity of events. For instance, the sequence chart allows for highlighting the critical path of the simulation: This path represents the minimum runtime which is required to execute a simulation run under consideration of event inter-dependencies and the computational complexity of the events. All events on the critical path therefore limit the runtime performance and are hence candidates for optimization. To further analyze the critical path, our sequence chart extension utilizes a red-green color scheme that indicates for each event on the path the computational complexity that takes place in parallel. More precisely, an event on the critical path with only a few parallel events is colored red since it causes a period of low computational parallelism. Conversely events on the critical path with a high degree of parallel events are colored green. This enables the developers to quickly identify potential performance bottlenecks. Figure 2 illustrates the visualization of the critical path in a trace of the MMN queueing model.

Nevertheless, since simulation traces can be large, manually checking thousands of events for bottlenecks quickly becomes tedious or infeasible. Hence, the extended sequence chart features a bottleneck detection mechanism that automatically scans the critical path for regions of low parallelism, i. e., events that overlap with only few computationally complex parallel events. Since the degree of parallelism varies heavily from model to model, developers can configure the bottleneck detection mechanism by specifying a threshold for the degree of parallelism – every event below this threshold is considered to be a bottleneck. Once a bottleneck is found, it is the task of the model developer to analyze the cause of the bottleneck and modify the model accordingly – for instance by replacing a single complex event with multiple smaller ones that can be processed in parallel.

Implementation. Our changes to the existing implementation comprise modifications of the Java-based visualization plug-in as well as the tracing library. Specifically, the modifications of the visualization plug-in involve the described drawing functionality and respective toolbar buttons to control the new features. Moreover, the changes to the tracing library include extensions of the API to enable recording and accessing of event durations and processing times. For recording the processing times, we additionally introduce a new trace file entry of type `EventEndEntry`. This addition is necessary to maintain the order of events in the trace file and minimize the impact of our changes: Originally, the occurrence of an event is logged by means of an entry of type `EventEntry` *before* the event handler is actually executed. However, the processing time of an event is known only *after* its execution. Since further entries regarding newly scheduled events are logged during the execution of an event, the new entry type de-couples logging of the start and the end of an event execution while allowing for logging arbitrary many entries in-between. Nevertheless, by hiding the new entries in the parsing process, our changes are transparent to existing plug-ins, such as the EventLogTable. In addition, the extended sequence chart is backwards compatible to traces generated by stock versions of OMNeT++. The current implementation of Horizon, the extended sequence chart, and the sample model are available in the source code repositories on our project website[1].

Acknowledgments: This research was funded by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC), German Research Foundation grant DFG EXC 89.

[1] `https://code.comsys.rwth-aachen.de/redmine/projects/horizon-public`