Journal of Systems Architecture 59 (2013) 307–314

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Grasp: Visualizing the behavior of hierarchical multiprocessor real-time systems ☆

Mike Holenderski *, Reinder J. Bril, Johan J. Lukkien

Eindhoven University of Technology, Den Dolech 2, 5600 AZ Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Available online 17 July 2012

Keywords:
Trace visualization
Hierarchical scheduling
Multiprocessors
Real-time systems

ABSTRACT

Trace visualization is a viable approach for gaining insight into the behavior of complex distributed real-time systems. Grasp is a versatile trace visualization toolset. Its flexible plugin infrastructure allows for easy extension with custom visualization and analysis techniques for automatic trace verification. This paper presents its visualization capabilities for hierarchical multiprocessor systems, including partitioned and global multiprocessor scheduling with migrating tasks and jobs, communication between jobs via shared memory and message passing, and hierarchical scheduling in combination with multiprocessor scheduling. For tracing distributed systems with asynchronous local clocks Grasp also supports the synchronization of traces from different processors during the visualization and analysis.

1. Introduction

Modern real-time systems are becoming increasingly more complex, with many tasks executing concurrently on many processors, making it difficult to understand the system behavior. A popular trend in coping with the vast number of tasks and the resulting interferences between them is to hide tasks inside components and to integrate the system from those components. This approach requires hierarchical scheduling, which has been covered extensively in the literature for uniprocessor systems. Recently, the real-time literature has been investigating applying hierarchical scheduling to multiprocessor platforms. In this paper we address the problem of how to provide insight into complex interaction patterns between jobs executing in a hierarchical multiprocessor system.

Several approaches are available for tackling the complexity of modern software systems. Ideally, every system would be meticulously documented, providing a formal yet concise description of the emergent system behavior. However, this is a long and costly process without immediate effects (such as additional functionality) and is therefore not common in practice. Examples of poorly documented code and system designs are abundant. The description of the dynamic system behavior therefore needs to be extracted from existing systems. There are modeling and verification tools available, which rely on the developers analyzing the implementation and constructing its model. These tools then employ formal methods to verify the behavior of the extracted model against an abstract model. The state of the art modeling and verification techniques, however, are not scalable and therefore can be applied to verify only a small portion of the entire system.

Visualization tools offer an interesting alternative. Existing systems can be instrumented to generate runtime traces, which can then be analyzed by engineers and researchers, leveraging their expertise and human capacity to recognize patterns, to gain insight into the system behavior. The challenge here lies in representing and presenting the information in an intuitive way, enabling the user to extract the essential properties of the analyzed system. While trace visualization on its own is insufficient for the verification of timing constraints of a real-time system, it is well suited for early design of such systems before the formal validation stage is reached.

Grasp is a toolset for tracing and visualizing the behavior of complex real-time systems. Its main strength lies in providing many different visualizations for various real-time primitives and scheduling techniques in a consistent and intuitive way. Its flexible architecture allows one to extend it easily with new visualization and analysis techniques.

We have been using Grasp extensively within our group during our research on embedded real-time systems and the development of various extensions of a commercial real-time operating system μC/OS-II, including a hierarchical scheduling framework and slot shifting. *The use of Grasp* has also been reported in [1,2,22] where it was used to gain insights into new approaches for hierarchical scheduling in Linux and VxWorks operating systems. Recently Grasp was used in the context of the SOFIA project to visualize the communication patterns between sensor nodes in a smart home environment.

In this paper we focus on visualizing traces of multiprocessor systems. The challenge here lies in representing and presenting

☆ The work presented in this paper was supported in part by the European ITEA2-CANTATA project. The Grasp toolset together with example traces is available for Linux, Mac and Windows at http://www.win.tue.nl/san/grasp.
 * Corresponding author.
 E-mail address: m.holenderski@tue.nl (M. Holenderski).

the execution and communication between jobs running on different processors in an intuitive and compact way. Moreover, if the timestamps of events occurring on a processor are recorded in its local time, special care must be taken to synchronize the individual traces. While some custom-built clusters such as IBM Blue Gene offer sufficiently accurate global clocks, most distributed systems can only rely on local clocks [3]. If the local clocks drift too far apart it may lead to inaccuracies or even errors during the trace analysis or visualization, as the causality between events may appear to be broken (e.g. messages arriving before they were sent).

1.1. Contributions

In this paper we build on top of our previous work presented in [12]. We focus on visualizing the timing of job execution and communication in the context of multiprocessor systems. In particular, we demonstrate Grasp's ability to visualize

- partitioned and global multiprocessor scheduling,
- migrating tasks and jobs,
- communication between jobs via shared memory and message passing,
- hierarchical scheduling in combination with multiprocessor scheduling.

We also describe Grasp's interface for synchronizing timestamps in traces generated in a distributed system.

1.2. Outline

Section 2 summarizes the related work, followed by an overview of the Grasp toolset in Section 3. Grasp's support for multiprocessor scheduling is presented in Section 4 and its support for hierarchical multiprocessor scheduling is presented in Section 5. Section 6 describes synchronization of distributed traces. Concluding remarks and future work are presented in Section 7.

2. Related work

Existing visualization tools for real-time systems are specialized to visualize a fixed set of behaviors. For example, the Tracealyzer [17] and TimeDoctor [21] are targeting only non-hierarchical uniprocessor systems. Making a step towards distributed systems is not trivial. Grasp, on the other hand, supports multiprocessor systems with two level virtualization.

There are several trace visualization tools which support the development of parallel programs on uniform parallel-processor platforms, such as VAMPIR [18], Paje [14], Jedule [13], or Scalasca [11]. They illustrate the execution of parallel jobs and communication between them, but they are limited to flat systems. To the best of our knowledge no visualization tools currently support the visualization of hierarchical scheduling in a uniprocessor or multiprocessor setting.

Traces accepted by most tools are lists of timed events, often in a binary format. Grasp, on the other hand, has adopted the idea of treating the trace as a script. On the one hand, Grasp traces are more verbose and require more storage space, compared to the binary format. On the other hand, they allow for large degree of flexibility, making it easy to add new events to the event model without changing the core implementation of the visualization and analysis components. This makes Grasp ideal for rapid prototyping of new visualization techniques. We have exploited this flexibility during the development of Grasp's various visualization and analysis features.

There are several tracing tools available, mainly for the Linux platform, which generate traces. Examples include the Data Stream Kernel Interface (DSKI) [6], Ftrace [10], and Dtrace [9]. DSKI is a platform independent interface standard to support collection of a variety of performance data from the operating system internals. It has been implemented on Linux. Ftrace and Dtrace are integrated in many Linux distributions. They exhibit low performance overhead and low memory footprint. In order to leverage their popularity, we have implemented a converter from the sched_switch tracer output of Ftrace, allowing to use Grasp in many Linux and Unix environments.

When tracing is used to analyze the behavior of distributed systems, then there is the additional problem of synchronizing the times of events occurring on different nodes. Time differences among distributed clocks can be characterized in terms of their relative offset and drift. If we assume constant drift, then the local time can be mapped onto the global (or master) time via linear interpolation. Existing time synchronization algorithms compute the interpolation based on the timestamps of messages exchanged back and forth with a master node [8,15] or with other nodes [16,5,4]. The timestamp synchronization in Grasp is based on [15].

The authors of [4,3] observe that while linear offset interpolation might prove satisfactory for short runs, measurement errors and time-dependent drifts may create inaccuracies and violate causality relations during longer runs (e.g. a message is received before it was sent). They propose a method for fixing these errors by postponing certain events in the trace. However, while maintaining the causality relations in traces, their methods change the timing of the events. As Grasp is targeting real-time systems, it relies on linear interpolation for synchronizing traces and if it detects inconsistencies in the ordering of events then it notifies the user that the constant drift assumption was violated.

3. Grasp overview

The Grasp toolset is composed of three entities: the Grasp Recorder, the Grasp Trace and the Grasp Player, as shown in Fig. 1.

The Grasp Recorder is embedded in the target system and is responsible for generating a trace. The generated Grasp Trace contains the raw data from a particular system run. The Grasp Player reads in a trace and displays it in an intuitive way.

3.1. Grasp Recorder

The Grasp Recorder is implemented as a library providing functions to initialize the recorder, log events, and finalize the recorder. Calls to the event logging methods are inserted at several places inside the kernel to log common events, such as context switches, arrival of tasks, or server replenishment. The recorder also provides a function to log custom events, which programmers may call inside their applications.

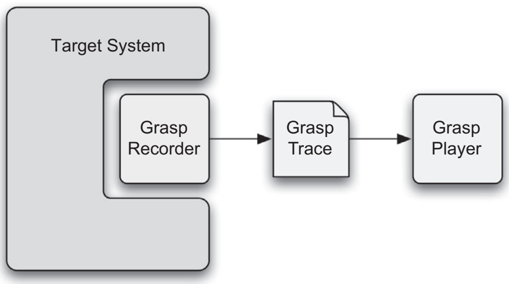

Fig. 1. Overview of the Grasp architecture.