TuningFork: A Platform for Visualization and Analysis of Complex Real-time Systems

David F. Bacon Perry Cheng David Grove
IBM Research
{bacon, perryche, groved}@us.ibm.com

Abstract

Debugging the timing behavior of real-time systems is notoriously difficult, and with a new generation of complex systems consisting of tens of millions of lines of code, the difficulty is increasing enormously. We have developed TuningFork, a tool especially designed for visualization and analysis of large-scale real-time systems. TuningFork is capable of recording high-frequency events at sub-microsecond resolution with minimal perturbation. Users can visualize system activity online in real-time and interactively explore the data. Data can be gathered from multiple layers and/or components and synthesized into visualizations that illuminate whole system interactions. Interactive exploration of hypothesis is naturally supported by direct manipulation to quickly build up complex visualizations.

Categories and Subject Descriptors C.3 [*Special-Purpose and Application-Based Systems*]: Real-time and embedded systems

General Terms Experimentation, Performance

Keywords Visualization, Vertical Profiling, TuningFork

1. TuningFork Design Requirements

TuningFork is an Eclipse-based tool that supports online and offline visualization and analysis of real-time activity by multiple subsystems. It has been publicly available via IBM alphaWorks [2] since September 2006.

TuningFork was designed to support the construction of real-time systems from multiple independently developed sub-components running on a network of server-class multiprocessors. Achieving this goal entailed satisfying a demanding collection of requirements. The most important are summarized here, a detailed discussion can be found in [1].

Finding Needles in Haystacks. The root cause of a failure could be a single event in a week-long execution taking four microseconds instead of one. This implies that the system must support the generation and processing of a very large volume of events to precisely capture system activity.

Discovery of Unexpected Behavior. Failures can be caused by effects for which the user as yet has no quantitative measure. Thus the tool must provide visualization capabilities that facilitate observation of unexpected behavior.

Trace Collection as a Real-time Task. Trace collection occurs in the actual real-time system itself (not in the visualizer). It is critical that trace collection be implemented such that its effects are minimal, predictable, and pre-emptable.

Trace Visualization as a Real-time Task. Since faults may be safety-critical, users must be able to observe, diagnose, and correct problems as quickly as possible. This means that the tool must be able to present the information in a short, bounded amount of time after the originating event occurs.

"TiVO" Control of Visualization. To observe events in real-time, the system must be capable of rendering a dynamic image of events, rather than a static representation. Furthermore, it must be possible to pause, reverse, and change zoom factor to quickly hone in on an unusual event.

High-performance Rendering. To achieve high-quality real-time playable views, the rendering performance of each view must be very good: the user may have 6 views open in play mode, each showing information about thousands of events. In play mode we would like to achieve 24 frames per second, which leaves less than 7 ms to render each view.

Interactive Data Exploration. Many real-time faults begin as unexpected non-failing behavior. Therefore it is necessary to enable exploration of the available data by opening new views, computing and visualizing new functions over the data, changing resolutions, and moving to arbitrary points in the execution time.

Zooming Hours to Microseconds. Supporting data exploration and the search for needles in haystacks requires an ability to rapidly navigate across huge time scales: from several hours down to a few microseconds. In other words, the tool must be able to scale across 11 orders of magnitude with

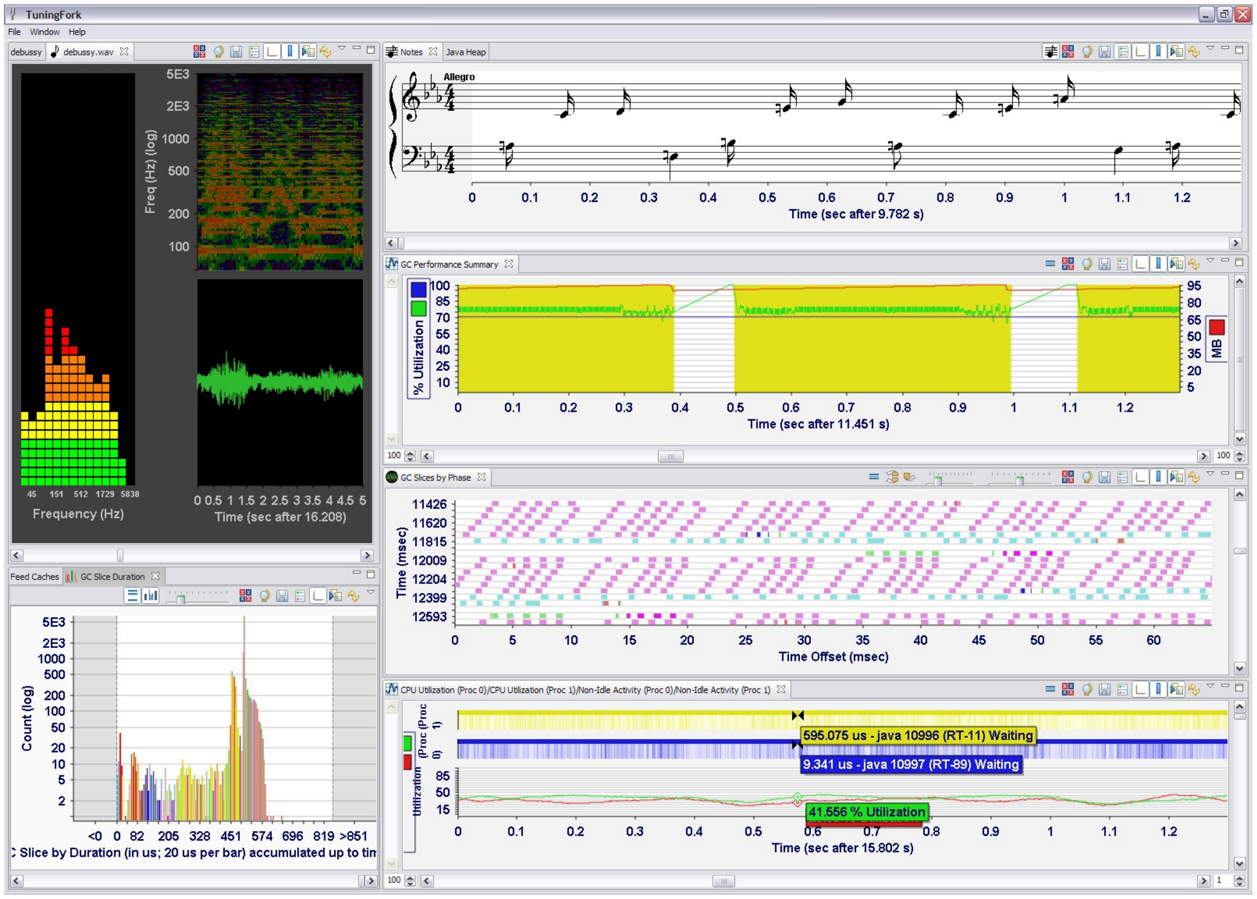

Figure 1. TuningFork visualization of the Harmonicon (an all-Java music synthesizer). The figures present a time-synchronized, vertical view of application, JVM, and operating system activity. At the top are application-specific visualizations of the sound wave (left) and midi notes (right) being synthesized by the Harmonicon. The middle two figures on the right show JVM-level performance, in particular overall garbage collection performance and scheduling. The histogram on the bottom left shows the distribution of GC pauses (log-scale y axis). The bottom right figure shows OS context switching activity and overall CPU utilization. Hovering over a point in time displays more detail for the events that happened at that instant.

interactive response. This is an issue in both the user interface – providing an intuitive and perceptually stable interface to navigate across time scales – and in the trace processing infrastructure – by providing fast random access and efficient, accurate summarization.

Analysis of Very Large Traces. A system generating a 128-bit event (64-bit timestamp and 64 bits of data) once every 10 microseconds will produce 1.6 MB/s or 138 GB/day of trace data. The system must therefore be capable of indexing, summarizing, and randomly accessing very large data sets.

Vertical Integration of Information. The real-time system is a composite of the hardware, operating system, virtual machine(s), and applications. Since these components may all affect and interfere with each other, the tool must be capable of integrating information from all of these sources.

Extensibility. Since the dynamic deployment environments are unpredictable, the tool must be highly extensible. This means accepting new trace formats, creating new data filters, and developing new visualizations.

2. Demonstration Overview

In the demonstration, we will begin by discussing TuningFork's overall architecture and capabilities. The vast majority of the demonstration will consist of an interactive demonstration of TuningFork's unique capabilities by replaying some of the scenarios in which we have used TuningFork. A sample of one such scenario is shown in Figure 1.

References

[1] Bacon, D. F., Cheng, P., Frampton, D., and Grove, D. Tuningfork: Visualization, analysis, and debugging of complex real-time systems. Tech. Rep. RC24162, IBM Research, 2007.

[2] IBM. TuningFork Visualization Tool for Real-Time Systems. URL www.alphaworks.ibm.com/tech/tuningfork, 2006.