JPL, CMU, and Sun's Evaluation Effort of Real-Time Java

Mark Indictor
Mark.Indictor@jpl.nasa.gov

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.Reference herin to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

Overview of Golden Gate

- The Jet Propulsion Laboratory
  - Evaluate and retire risks of using RTSJ as software platform for space flight control

CMU

- Carnegie Mellon University
  - HDCP (High Dependability Computing Platform)
  - Develop methodologies for measurement and evaluation of software reliability

- Sun Microsystems
  - Promote utilization of RTSJ for mission critical applications
  - Evolve the RTSJ to better support mission critical requirements

Accomplishments: FY03

Rocky 7 rover in the JPL Mars Yard with camera mast raised

- Completed first proof-of-concept, RTSJ implementation of a small subset of the MDS (Mission Data System) architecture
  - Developed an operational rover system
  - Developed a suite of performance test programs that can be used to compare RTSJ with C++ implementations
  - Acquired experience and information about effective use of available tools.
  - Test suite provided basis for experiments comparing the two platforms.
- Experienced the pain of early adoption of new technology

Accomplishments: FY03 (cont'd)

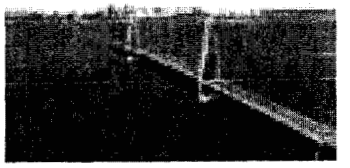
- Recognized need for standards-based benchmark suite to characterize RTSJ's performance
  - Began test suite with cooperation and input from
    - Adward Pla, AFRL/Boeing
    - Greg Bollella, Sun Microsystems
    - Jan Vitek, Open Virtual Machine (OVM)
    - Angelo Corsaro: rtjPerf
  - Suramadu was born!

- Created a preliminary report describing:
  - Golden Gate's first-year experiences
  - Preliminary performance-suite results
  - Expectations and recommendations for the future

Golden Gate: Goals

- Generate full report for Mars Science Laboratory (MSL)
  - R&TD efforts to be captured in FY04 report
    - Completed performance suite results
    - Further documentation of experiences
    - Recommendations for continued flight software development using RTSJ
- Create a positive influence on vendors
  - Promote an environment of cooperation and support
  - Help guide vendors toward developing tools that are…
    - Capable of delivering acceptable metrics for mission-critical applications
    - Verifiable and reliable
- Unified performance suite hand-off
  - Evolve suite to be general, easily comprehended, and extensible
  - Adapt for integration with Open Group TETware test bed
  - Aim for hand-off of test suite to Open Group
    - Lay groundwork for management by industry consortium

"Suramadu"



(Suramadu: A bridge connecting the islands of Java and Madura in East Java)

- Sounds like a cocktail: Madura and Java on the Rocks?

*** but seriously ***

- Represents connection and integration: a bridge
  - Transitions toward a unified, industry responsive, and standards based benchmarking suite
  - Comprehensively tests areas of greatest importance
  - Provides answers that allow critical decisions to be made

Suramadu
RTSJ Specific

- **Boundary**
  - JNI Overhead
  - Crossing the hard/soft/non real-time boundary
- **Memory**
  - Scope memory entry and exit
  - Allocation in Immortal
  - Allocation in Scope
  - Allocation in Heap
- **Latency**
  - Synchronization
  - Queuing
  - Inheritance
  - Class Loading

Suramadu
Cross-Platform Comparisons

- **Timeliness**
  - Jitter
- **Throughput**
  - Floating Point
  - Integer
  - Logical
- **Memory**
  - Allocation and Freeing
  - Footprint
    - On disk
    - RAM

Suramadu
Support

- **Support**
  - High resolution time measurement
  - Statistical calculation
  - Collection of results and output
  - Test and thread execution abstraction

Experimental Results

Comparisons performed using ANOVA on multiple result sets

- ***RTSJ Specific***
  - ***Jitter:***
    - *Tests at 10ms and 500ms periods with 0, 1, and 10 background threads*
      - *At such low loading, no difference in RT and NHRT threads.*
      - *Jitter range from 89.7 to 2294.78ns (late)*
      - *Need to load more heavily (more background threads) and retest*
- ***RTSJ to C++ Comparison***
  - ***Throughput:*** *Comparison with C++ indicates*
    - *C++ 473% faster for logical shifting operations*
    - *C++ 428% faster for integer arithmetic operations*
    - *C++ 224% faster for floating point operations*
      - *Hardware FPU could account for difference*
  - ***Startup Time:***
    - Important after safing a spacecraft
    - *Java 150% faster than C++*
      - *Attributable to VxWorks symbol table loading*

Analysis

- ***On-Disk Footprint***
  - Linux kernel smaller than VxWorks kernel
  - Java Virtual Machine Larger than VxWorks libraries
  - Java application image size larger than C++

- ***In-Memory Footprint***
  - VxWorks/C++: 156.3Kb
  - Java: 7780Kb

- ***Throughput***
  - Java Native Interface (JNI) Overhead
    - Copy semantics expensive
    - Location of JNI in system architecture critical
      - Moving higher in architecture reduces number of calls, increases amount of native (C/C++) code
    - ***Java needs support for direct unmapped memory and I/O access***
      - Specification is vague on this issue.
      - JTime supports direct memory access as long as it is mapped
  - Byte Code Interpretation
    - Java slower than C/C++
    - ***Java needs AOT or JIT!***

FY03 Findings

- Timeliness in JTime is comparable to that of C/C++
- Throughput in JTime is two to five times slower in *interpreted* Java
  - AOT and/or JIT technology is required for viability
- RTSJ needs a story describing direct access of unmapped memory and I/O space
  - Java abstraction of OS for writing device drivers
- RTSJ requires changing programming model from that of standard Java programming
  - Manual memory management (scope scratchpads and pools)
  - Arguments passed by value (copy semantics) across RT/NRT boundary

Analysis (cont'd)

- ***Tool Maturity***
  - Early adoption: problems are inevitable
  - Up-close and personal vendor support is vital
    - TimeSys was very responsive
  - Functionality as yet unimplemented
    - Direct unmapped memory and I/O access
    - RT Scheduler runnable in user space (not as root)
    - Debugging and Profiling interfaces
    - Scope-safe class libraries
      - java.util.*
      - java.math.*
      - . . .
    - AOT and JIT technologies

Plans for FY04

- Continuing test suite development
  - Implement more of the benchmarks
  - Add more benchmarks
  - Refine existing benchmark specifications
- Re-factor MDS architecture
  - Modify model to better define problem domain
  - Consider RTSJ strengths and limitations in mapping problem domain to solution
  - Identify areas where the RTSJ (or its implementation) fail to meet design needs
  - Contribute ideas toward extending RTSJ moving forward
- Devise the next round of experiments
  - Focus on System Engineer's concerns
  - Design experiments to maximize discovery
    - Focus on areas closer to boundary conditions
    - Include more, and more effective, comparisons between RTSJ and C++

Reserve Slides

suramadu.tests.jitter.JavaJitterTest

Measures the jitter of regular Java Threads.

This test It is NOT specific to the RTSJ specification, and may freely be used on regular J2ME and J2SE VMs.

PURPOSE: This test provides an indication of timeliness in a periodic thread, both with and without competing non-real-time threads running in the system.

DESIGN: Given a periodic value for a thread, this test measures the actual time between periods. A periodic thread is set up, and a time measurement is taken immediately upon entering its run() method. The difference between the measurements taken indicates the actual time between the beginning of execution of the actual thread's code. Any deviation from the required period is 'jitter'. At the conclusion of the test, statistical calculations are performed upon these time measurements, and a summary is printed.

ASSUMPTIONS: This test uses regular Java threads, and does not require an RTSJ implementation in order to run. It is used as a baseline on which to compare RTSJ performance.

OUTPUTS: The output contains the maximum, minimum, median, mean, standard deviation and mode statistical measurements on the collected data.

EXAMPLES: None.

ORIGINS: This test is based upon ideas from the Boeing/AFRL and the Sun's rtpresto suites.

suramadu.tests.jitter.RTJavaJitterTest

Measures the jitter of Realtime Threads and NoHeap Realtime Threads.

This test is specific to the RTSJ specification, and may not be run under regular J2ME and J2SE VMs.

PURPOSE: This test provides an indication of timeliness in a periodic thread, both with and without competing non-real-time threads running in the system.

DESIGN: Given a periodic value for a thread, this test measures the actual time between periods. A periodic thread is set up, and a time measurement is taken immediately upon entering its run() method. The difference between the measurements taken indicates the actual time between the beginning of execution of the actual thread's code. Any deviation from the required period is 'jitter'. At the conclusion of the test, statistical calculations are performed upon these time measurements, and a summary is printed.

ASSUMPTIONS: This test uses RTSJ Real-time and NoHeap Real-time threads, and requires an RTSJ implementation in order to run.

OUTPUTS: The output contains the maximum, minimum, median, mean, standard deviation and mode statistical measurements on the collected data.

EXAMPLES: None.

ORIGINS: This test is based upon ideas from the Boeing/AFRL and the Sun's rtpresto suites.

suramadu.tests.throughput.FloatingPoint

Measure Java's arithmetic capabilities to measure throughput during computationally intensive operations with floating point numbers.

PURPOSE: To measure the Java and RTSJ computational throughput with floating point arithmetic operations.

DESIGN: This class does floating point operations - 30 operations in one call to doOps(). The arguments to that function are simply to make sure that the compiler can't optimize away any operations. x3 must not be zero.

ASSUMPTIONS:

OUTPUTS:

EXAMPLES:

ORIGINS:

suramadu.tests.throughput.IntegerOps

Measure Java's arithmetic capabilities to measure throughput during computationally intensive operations using integers.

PURPOSE: To measure the Java and RTSJ computational throughput with integer arithmetic operations.

DESIGN: This class does integer arithmetic operations - 30 operations in one call to doOp()s. The arguments to that function are simply to make sure that the compiler can't optimize away any operations. n3 must not be zero.

ASSUMPTIONS:

OUTPUTS:

EXAMPLES:

ORIGINS:

suramadu.tests.throughput.Shifting

Measure Java's arithmetic capabilities to measure throughput during computationally intensive operations with shift operations

PURPOSE: To measure the Java and RTSJ computational throughput with shift operations.

DESIGN: This class does shift operations - 30 operations in one call to doOps(). The arguments to that function are simply to make sure that the compiler can't optimize away any operations. in the navigation of a spacecraft.

ASSUMPTIONS:

OUTPUTS:

EXAMPLES:

ORIGINS:

suramadu.tests.memory.ScopeMemoryTest

Tests the cost of entering and exiting ScopeMemory.

PURPOSE: Tests the cost of entering and exiting ScopeMemory.

DESIGN:

ASSUMPTIONS:

OUTPUTS:

EXAMPLES:

ORIGINS:

suramadu.tests.memory.VMRuntimeSize

Measure the memory footprint of a Java application.

PURPOSE: Measure the memory footprint of a Java application.

DESIGN: This object does not actually test the size of the VM. What it does instead is start and stay active for a configurable amount of time so that an external shell script can measure memory usage.

ASSUMPTIONS:

OUTPUTS:

EXAMPLES:

ORIGINS:

suramadu.tests.boundary.JNIOverhead

Measures the overhead involved in making a simple JNI call

PURPOSE: This test is particularly pertinent in calculating the error of high-resolution time measurement. Since it takes time to measure time, the results of this test can be used to correct for this overhead.

DESIGN: The JNI native method to retrieve CPU machine cycles is called repeatedly, and the value logged in an array. The differences between the first and all subsequent, consecutive calls is calculated, and statistical calculations made from these values. Since each difference represents the time it takes to return from the JNI call and then call it again, the total value can be considered the entire JNI call overhead even though it was measured in the reverse order as is considered normal.

ASSUMPTIONS: No attempt is made in this test to measure specific JNI callback or data copying methodologies.

OUTPUTS: The output contains the maximum, minimum, median, mean, standard deviation and mode statistical measurements on the collected data.

EXAMPLES: None.

ORIGINS: This test was conceived and written by JPL and the Golden Gate Team.

suramadu.util.stats.*

Provide statistical analysis support for all tests.

StatPack Static helper class to compute various statistical values of collections and arrays.

Histogram A class to facilitate the creation of a histogram data structure for examining the distribution of various kinds of numeric data.

HistogramBinSpec This class is a container for HistogramBin classes. Each HistogramBin defines the specification for a single bin in a Histogram. It is the duty of the HistogramBinSpec to enforce consistency across all HistogramBins added to it, making sure that there are not overlaps, etc.

HistogramBin This class represents a single bin in a histogram. It contains the complete definition of the criteria for being a member of this bin. By calling this classes isMember() method, it can be immediately determined whether a particular value is a member of this bin.

Counter A counter object suitable for storing in a Map

suramadu.util.time.*

Provide extremely high resolution time measurement capability.

Time This class is a utility class for measuring and calculating very high-resolution time based on the number of machine cycles between events being measured. It contains two native methods, getCycles(), and getTime() for retrieving low level time measurements from the underlying operating system. (See method descriptions for details of their functionality).

ClockspeedGHz A utility for calculating the clock speed of the test bed's processor. Provides the timebase to the Time class that allows it to calculate real-time results from cycle-count data. NOTE: This is a C sourced executable file.

suramadu.util.exec.*

Provides a group utilitiy classes and a framework for standardizing and simplifying the task of writing RTSJ Tests

JavaThreadLauncher Utility used to launch threads. It requires a Runnable object the type of thread you want to run in and the priority to run at. It will create the Thread, SchedulingParameters, ReleaseParameters MemoryParameters MemoryArea and ProcessingGroupParameters. It will the start and join to the thread.

PeriodicJavaThread Simulates the periodic capabilities of a RealtimeThread and a NoHeapRealtimeThread in the RTSJ. It allows both the period and the priority to be set.

RTThreadLauncher Utility used to launch realtime threads. It requires a Runnable object the type of thread you want to run in and the priority to run at. It will create the Thread, SchedulingParameters, ReleaseParameters MemoryParameters MemoryArea and ProcessingGroupParameters. It will the start the thread.

TestProperties The TestProperties class supports the creation and maintenance of a hierarchical set of input properties (loaded from a properties file with naming convention of <class name>.properties), and output properties (set from test implementation). Outputting raw and calculated data is supported both in raw data as well as summary format.