Chauffeur: A Framework for Measuring Energy Efficiency of Servers – Part 1

Jeremy Arnold
arnoldje@us.ibm.com

IBM Corporation

International Conference on Performance Engineering 2014

Introduction

The Chauffeur Worklet Development Kit (WDK) is a new framework for creating worklets to measure the performance and energy efficiency of servers.

- Part 1 will cover the design principles for Chauffeur. It will also include a demonstration of how to install and run Chauffeur, and make configuration changes.
- Part 2 will cover how to write new worklets and other code to customize Chauffeur behavior.

About Me

I am a Senior Software Engineer at IBM, currently focusing on cloud performance for Power Systems. Most of my 16 years at IBM have been focused on performance – for Java, WebSphere Application Server, IBM Systems Director, PowerVC, and miscelaneous related products.

I have been working on the SPEC power committee since 2006, and was one of the main developers for the SPECpower_ssj2008 benchmark and for SERT, and the architect of the Chauffeur framework.

I was awarded a B.S. in Computer Science from Utah State University, and in January, 2014 I completed my M.S. in Computer Science from University of Minnesota.

Agenda

About SPEC

- SPEC is a world-wide non-profit consortium formed in 1998 to establish, maintain, and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers
- Over 80 computer hardware and software vendors, educational institutions, and government agencies are members of SPEC
- SPEC has developed over 30 industry-standard benchmarks for system performance evaluation in a variety of application areas
- SPEC hosts a large public repository of over 20,000 well-documented, peer-reviewed benchmark results

The SPEC Power Committee

The SPEC Power committee was formed in 2006 to develop standard methods and metrics for comparing power consumption of server-class computers. Major contributions from the committee include:

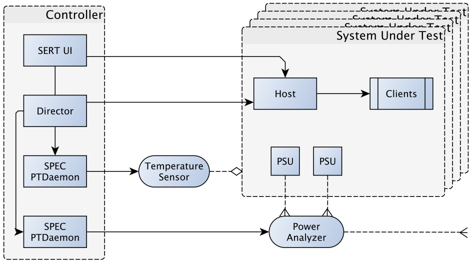
- The SPEC Power and Performance Benchmark Methodology, which describes best practices for measuring power consumption of servers [13]
- The SPECpower_ssj2008 benchmark, the first-industry standard benchmark that evalutes the power and performance characteristics of single- and multi-node servers [14]
- SPEC PTDaemon, used by multiple benchmarks to interface with various power analyzers and temperature sensors [12]
- The Server Efficiency Rating Tool (SERT), designed to measure energy efficiency of a wide variety of servers, and part of the US ENERGY STAR program [15]

SERT Overview

SPEC produced SERT to give a broad measure of energy efficiency for servers [8, 16].

Design Goals:

- Stress multiple system components
- Multiple synthetic worklets
- Multiple load levels
- Cross-platform support
- Test "as-shipped"



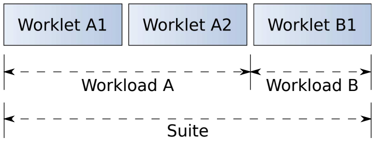
What is Chauffeur?

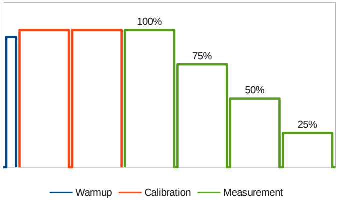
Chauffeur is a framework for building benchmarks and workloads at variable utilization, with a focus on measuring energy efficiency [7, 9].

While created to meet the requirements of SERT, Chauffeur was designed to be flexible for use in future research and development.

Workload developers only need to implement the core logic for their workload, and use Chauffeur infrastructure for common function. Many aspects of Chauffeur behavior are configurable. New code can be plugged in without changing Chauffeur itself.

Typical Chauffeur Execution





Each Chauffeur runs includes a Suite of one or more Workloads. Each Workload is composed of one or more Worklets.

Most worklets have three phases of execution: warmup, calibration, and measurement.

The calibration phase determines the maximum capacity of the system.

During the measurement phase, transactions are scheduled with appropriate delays to achieve the desired target throughput.

The Chauffeur Worklet Development Kit

SPEC released the Chauffeur WDK (Worklet Development Kit) to support research and the development of new worklets. The Chauffeur WDK is available from SPEC for a nominal fee. The WDK includes:

- The Chauffeur framework (source code and binaries)
- Sample worklets
- PTDaemon
- User's Guide and other documentation (also available on-line)
- Support is primarily via a web forum available to licensees.

Documentation and the order form are at
`http://www.spec.org/chauffeur-wdk/`

Reasons for Extending Chauffeur

- New workloads
- Different behavior
- Data collection
- Custom reports

Agenda

Design Criteria for Energy Efficiency Benchmarks

Benchmarks can be evaluated based on the following general criteria [3, 4, 5, 6, 10, 11, 17]:

1. Relevance
2. Reproducibility
3. Fairness
4. Verifiability
5. Usability

Energy efficiency benchmarks are subject to the same criteria, but each has additional facets specific to measuring energy efficiency.

Relevance

Relevant benchmarks mimic the behavior of some class of real applications.

Scalable:	Ability to use the resources of a wide range of systems
Variable Utilization:	Energy efficiency varies at different utilizations [1, 2]
Multi-system:	Energy sometimes can't be measured accurately for individual systems (e.g blades)

Reproducibility

Benchmarks should produce results which can be reproduced by others.

Consistency:	Running the benchmark multiple times under the same conditions will produce the same results
Description:	The hardware and software components and configuration are described in sufficient detail to allow an equivalent environment to be constructed

Fairness

Systems can compete on their merits without artificial constraints.

Optional Energy:	Results with and without energy measurements may not be comparable
Components:	Which components of the system must have power measured?

Verifiability

Results can be verified to be accurate.

Power Accuracy:	Accuracy of data from power analyzers depends on ranges and readings; requires dynamic verification

Usability

Easy-to-use benchmarks tend to have more results and better accuracy.

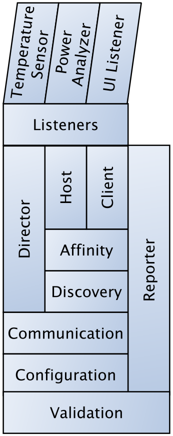
Self-describing:	Includes tools for automatically discovering system details
Energy Data Collection:	Use of SPEC PTDaemon or other tools to automatically collect power data

Agenda

Chauffeur Design

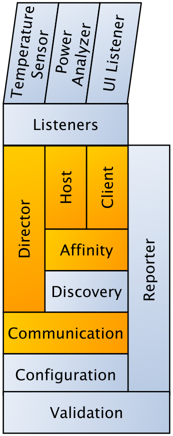
Key design points for the Chauffeur harness include:

- Multiple Hosts and Multiple Threads
- Energy Measurements
- Multiple Workload Support
- Easy to Run
- Flexible Configuration and Data Collection
- Self-validation
- Customizable Reporting
- Portability



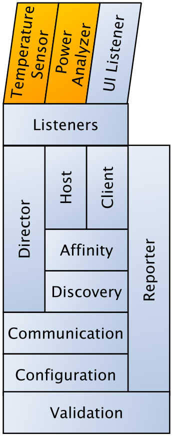
Multiple Hosts and Multiple Threads

- The Chauffeur "Director" and SPEC PTDaemon run on a controller system separate from the System Under Test (SUT)
- A Chauffeur "Host" runs on each server. There may be multiple Hosts (e.g. blades or virtual machines)
- Each Host launches one or more Client JVMs to execute worklet code.
- Platform-specific affinity code is used with most worklets to bind each client to specific processors
- For most worklets, Chauffeur uses one worker thread per logical processor



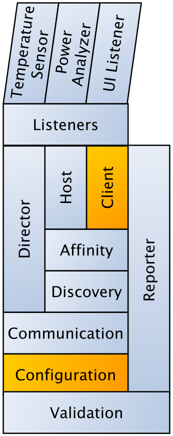
Energy Measurements

- Chauffeur is designed for measuring both performance and power
- Any workload can run at multiple load levels (e.g. 100%, 67%, and 33% of the system's capacity) to measure energy consumption at different loads
- SPEC PTDaemon is used to interface with Power Analyzers and Temperature Sensors.
- A "range setting run" can be used to identify appropriate range settings for power analyzers.



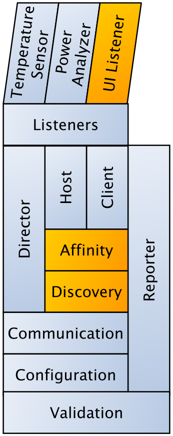
Multiple Workload Support

- Chauffeur can be used to run a single workload or a suite of multiple workloads
- In some cases workloads can be merged to run concurrently.
- Infrastructure provided by Chauffeur allows workload code to focus on business logic



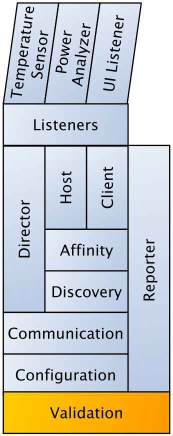
Easy to Run

- Workloads are self-calibrating
- Client JVMs are launched automatically, using platform-specific affinity commands
- Java heap sizes calculated automatically
- System discovery scripts can collect system configuration data



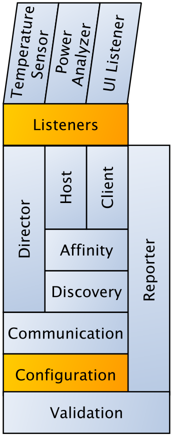
Self-validation

- Chauffeur includes a framework for validating the results of a run
- Validation is primarily performed at the end of the run, but some validation during the run can give early feedback to users
- Validators can mark the run invalid, or may provide warnings or informational messages.
- Sample validation in SERT:
    - Code has not been modified or recompiled
    - Configuration options are consistent with run rules
    - Power measurements meet accuracy thresholds
    - Minimum ambient temperature $\geq 20^{\circ}C$
    - Detect manual modification of results after a run is complete



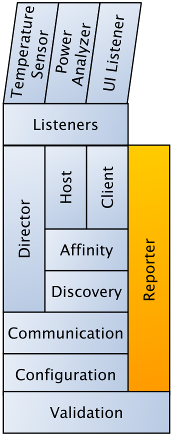
Flexible Configuration and Data Collection

- "Listener" interface for flexible data collection
- Runtime behavior configurable via XML file
  - Length of each measurement interval
  - Sequence of workloads and load levels
  - Worklet-specific configuration parameters
  - Client JVM count and options
- Plugins to support new behavior without Chauffeur modifications



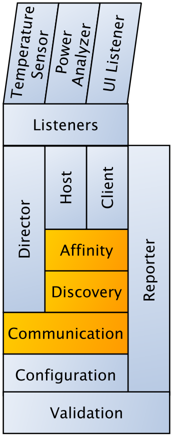
Customizable Reporting

- Results of each run are stored in an XML file
- At the end of each run, the Chauffeur Reporter generates HTML and plain text reports from the data
- In addition to the standard reports, the Reporter supports custom reports. They are generated by XSL transforms to either plain text (e.g. CSV files) or to a high-level XML-based representation of the report, which is then used to generate either HTML or text output.
- Custom reports can present existing data in a new way, or provide data that is not included in the existing reports – including data produced by custom listeners



Portability

- Chauffeur primarily written in Java
- Small amount of native code to call OS-specific APIs to collect discovery data and collect information for affinity
- Platforms that don't have native code can run without discovery or affinity, or specify affinity manually
- Includes support for calling native code within worklet transactions via Java Native Interface (JNI)
- Communication protocols designed to be language-neutral, so a future version could transition to native code at a courser granularity, or the entire Client could be reimplemented in another language



Agenda

Agenda

Installation

Extract the ChauffeurWDK distribution to any location on your Controller system and the System Under Test (SUT). For testing purposes, it's possible to use the same system as both the Controller and the SUT, but this isn't recommended for measurements.

Controller: Edit `test-environment.xml` to describe the hardware and software details of the test.

Controller: In `director.bat/.sh`, set `JAVA` to point to a Java 6 (or higher) JRE. Set `HOSTS` to the hostname or IP of your SUT(s). Run this script.

SUT: In `host.bat/.sh`, set `JAVA` to point to a Java 6 (or higher) JRE. Run this script.

Launching PTDaemon

PTDaemon configures and collects data from power analyzers and temperature sensors

See the PTDaemon Accepted Devices List for a list of supported devices: For test purposes, the "Dummy" device can be used.
`http://www.spec.org/power/docs/SPECpower-Device_List.html`

Follow the manual for your device along with the PTDaemon Measurement Setup Guide for information on setting up the device.

PTDaemon runs on the system that the device is physically connected to. This is normally the controller system, but can be a separate system.

Edit the `PTDaemon/runpower.*` and `runtemp.*` scripts as needed for each device. Run the script for each device.

Chauffeur Reports

At the end of each run, Chauffeur will create summary and detail reports, in HTML and plain text formats. These reports show the results as well as details about the system configuration.

The system configuration details are copied from `test-environment.xml` to the results file (`results.xml`). If necessary, these details can be edited directly in `results.xml` and then you can run the reporter manually to regenerate the report. If you make changes to the results themselves, the reporter will detect the modification and mark the run invalid.

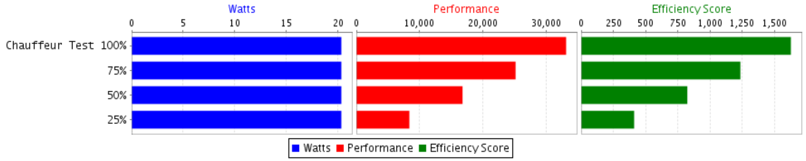
Sample Report

Chauffeur™ Report

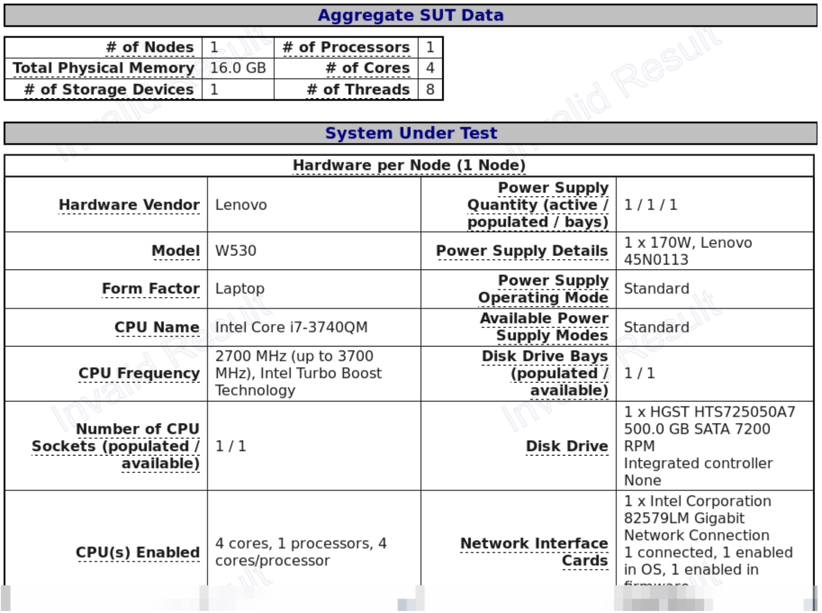
Lenovo W530			
Test Sponsor	IBM	**Software Availability**	Feb-2013
Tested By	IBM	**Hardware/Firmware Availability**	Jun-2013
SPEC License #	11	**System Source**	Single Supplier
Test Location	Rochester, MN, USA	**Test Date**	Mar 14, 2014

INVALID: Values from this report are not valid.
INVALID: The power analyzer Dummy is not an accepted device
INVALID: The temperature sensor DummyTemp is not an accepted device

Summary



Workload	**Worklet**	**Load Level**	**Performance Score**	**Average Active Power (W)**	**Efficiency Score**
Chauffeur Test	Chauffeur Test	100%	33,089.016	20.3	1,628.0
		75%	25,118.292	20.3	1,235.8
		50%	16,750.647	20.3	824.1
		25%	8,360.391	20.3	411.3



Aggregate SUT Data

# of Nodes	1	**# of Processors**	1
Total Physical Memory	16.0 GB	**# of Cores**	4
# of Storage Devices	1	**# of Threads**	8

System Under Test

Hardware per Node (1 Node)

Hardware Vendor	Lenovo	**Power Supply Quantity (active / populated / bays)**	1 / 1 / 1
Model	W530	**Power Supply Details**	1 x 170W, Lenovo 45N0113
Form Factor	Laptop	**Power Supply Operating Mode**	Standard
CPU Name	Intel Core i7-3740QM	**Available Power Supply Modes**	Standard
CPU Frequency	2700 MHz (up to 3700 MHz), Intel Turbo Boost Technology	**Disk Drive Bays (populated / available)**	1 / 1
Number of CPU Sockets (populated / available)	1 / 1	**Disk Drive**	1 x HGST HTS725050A7 500.0 GB SATA 7200 RPM Integrated controller None
CPU(s) Enabled	4 cores, 1 processors, 4 cores/processor	**Network Interface Cards**	1 x Intel Corporation 82579LM Gigabit Network Connection 1 connected, 1 enabled in OS, 1 enabled in firmware

Agenda

Basic Configuration

A Chauffeur run is configured with a `config.xml` file. Creating this file from scratch can be a bit daunting. Making modifications to an existing file is fairly straightforward.

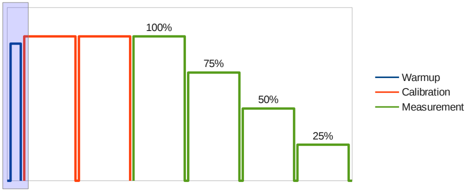
Our goal was to provide flexibility to developers but limit the need for configuration by users.

config.xml

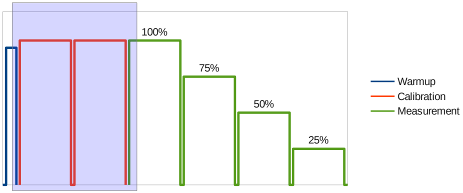
```xml
<?xml version="1.0" encoding="UTF-8"?>
<chauffeur xmlns="http://spec.org/power_chauffeur"
           xmlns:xi="http://www.w3.org/2001/XInclude">

  <xi:include href="test-environment.xml"/>

  <definitions>
    <interval-length id="testIntervalLength">
      <premeasurement>5s</premeasurement>
      <measurement>1m</measurement>
      <postmeasurement>5s</postmeasurement>
    </interval-length>
  </definitions>
```

```xml
<suite>
  <description className="org.spec.chauffeur.test.ChauffeurTest"/>

  <xi:include href="listeners.xml"/>

  <client-configuration>
    <clients key="Test">
      <count>1</count>
      <option-set/>
    </clients>
  </client-configuration>
```

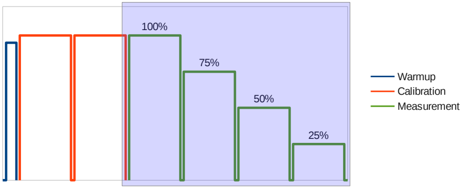
```xml
    <workload enabled="true">
      <name>Chauffeur Test</name>

      <worklet enabled="true">
        <launch-definition>
          <configuration-key>Test</configuration-key>
        </launch-definition>

        <workletDefinition>
          <location>
            org/spec/chauffeur/test/chauffeurTestWorklet.xml
          </location>

          <classpath>
            <entry>lib/chauffeurTest.jar</entry>
          </classpath>

          <parameters>
            <parameter name="minimum-sleep-time">500ms</parameter>
            <parameter name="maximum-sleep-time">5000ms</parameter>
          </parameters>
        </workletDefinition>
```



```
<warmup-phase>
  <sequence>
    <interval-series className="NoDelaySeries">
      <interval-count>1</interval-count>
    </interval-series>
    <interval-length>
      <premeasurement>0s</premeasurement>
      <measurement>1m</measurement>
      <postmeasurement>0s</postmeasurement>
    </interval-length>
  </sequence>
</warmup-phase>
```



```
<calibration-phase>
  <sequence>
    <interval-series className="NoDelaySeries">
      <interval-count>2</interval-count>
    </interval-series>
    <interval-length ref="testIntervalLength"/>
  </sequence>
  <calibrator className="AverageThroughputCalibrator">
    <average-intervals>2</average-intervals>
  </calibrator>
</calibration-phase>
```



```
        <measurement-phase>
          <sequence>
            <interval-series className="GraduatedMeasurementSeries">
              <interval-count>4</interval-count>
            </interval-series>

            <interval-length ref="testIntervalLength"/>
          </sequence>
        </measurement-phase>
      </worklet>
    </workload>
  </suite>
</chauffeur>
```

Agenda

Changing Interval Length

An interval length includes premeasurement, measurement, and post-measurement periods. Only transactions from the measurement period are recorded. The other periods help ensure the system is running at steady state during the measurement.

```
<interval-length>
  <premeasurement>30s</premeasurement>
  <measurement>2m</measurement>
  <postmeasurement>15 seconds</postmeasurement>
</interval-length>
```

Durations can be specified with a variety of units: nanoseconds (ns), microseconds (us, μs), milliseconds (ms), seconds (s), minutes (m), hours (h), days (d).

Changing Client Count and Options

```
<client-configuration>
  <clients key="Test">
    <count>1</count>
    <option-set/>
  </clients>
</client-configuration>
```

```
<client-configuration>
  <clients key="Test">
    <count>logicalCores / 2</count>
    <option-set>
      <parameter>-server</parameter>
    <option-set>
  </clients>
</client-configuration>
```

The <count> can be a JavaScript expression, and can make use of variables:

- `logicalCores`
- `physicalCores`
- `numaNodes`
- `physicalMemoryBytes`
- `paramMap` (worklet parameters)

Manual Calibration

Rather than using calibration to find the maximum throughput, you can specify the calibrated throughput manually. Replace the `<calibrator>` in `config.xml` with a `ConstantValueCalibrator`.

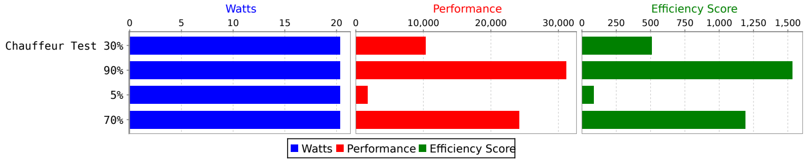
```xml
<calibration-phase>
  <sequence>
    <interval-series className="NoDelaySeries">
      <interval-count>1</interval-count>
    </interval-series>
    <interval-length ref="testIntervalLength"/>
  </sequence>
  <calibrator className="ConstantValueCalibrator">
    <scenarios-per-second>20000</scenarios-per-second>
  </calibrator>
</calibration-phase>
```

You can find the calibrated scenarios-per-second from a previous run in the `<calibrationResult>` section of `results.xml`. Note that there is currently a known issue in Chauffeur which requires you to have at least one calibration interval.

Customized Measurement Sequence

Real systems don't normally move from 100% to 75% to 50% to 25% of their maximum throughput. You can set your own sequence of percentages using a `ThroughputPercentageSeries`:

```
<measurement-phase>
  <sequence>
    <interval-series className="ThroughputPercentageSeries">
      <interval-count>4</interval-count>
      <percentage-sequence>30 90 0 70</percentage-sequence>
    </interval-series>
    <interval-length ref="testIntervalLength"/>
  </sequence>
</measurement-phase>
```



Notes on Customized Measurement Sequences

If the `<interval-count>` is different from the number of `<percentage-series>` entries, the sequence will be repeated as many times as needed to reach the required number of intervals.

The `ThroughputPercentageSeries` can be combined with the `ConstantValueCalibrator` to execute a series of specific throughput values.

Be careful when making frequent large changes to the load level – as the power usage of the system changes, the power analyzer may need to switch ranges. If the system doesn't have time to settle to the new load, the measurements may not be accurate.

Agenda

References

The `id` attribute can be used to assign an identifier to any element in the configuration. Then any element with a matching `ref` attribute will be replaced with the element it refers to.

This is useful for sharing the same configuration in multiple places.

The `<definitions>` section can be used to define elements for use elsewhere (with `id`/`ref`). For example, commonly modified options can be kept at the top of the file and referred to where they are needed.

```xml
<definitions>
  <interval-length id="testIntervalLength">
    <premeasurement>5s</premeasurement>
    <measurement>1m</measurement>
    <postmeasurement>5s</postmeasurement>
  </interval-length>
</definitions>

...

<measurement-phase>
  <sequence>
    <interval-series className="GraduatedMeasurementSeries">
      <interval-count>2</interval-count>
    </interval-series>

    <interval-length ref="testIntervalLength"/>
  </sequence>
</measurement-phase>
```

Include Files

The configuration can be spread across multiple files to make it easier to read, and to allow for reuse. For Chauffeur, the test environment (`test-environment.xml`) and listener definitions (`listeners.xml`) are often handled this way.

Inclusion is handled via the XInclude standard.

```
<chauffeur xmlns="http://spec.org/power_chauffeur"
           xmlns:xi="http://www.w3.org/2001/XInclude">

  <xi:include href="test-environment.xml"/>
```

Agenda

Test Environment Description

The `test-environment.xml` file describes the system under test. Nothing in this file influences the behavior of the run. The information in this file is copied to the results file and used by the Reporter to show the system configuration.

Filling in the system details is optional. You can also edit the values in the results file after the run and re-run the Reporter to update it with the new information. By convention, default text values begin with an underscore. The Reporter will highlight these fields in yellow to remind you which fields have not been filled in.

test-environment.xml (excerpt)

```xml
<Node>
  <Quantity>1</Quantity>
  <Hardware>
    <Availability>2013-06</Availability>
    <Vendor href="http://www.lenovo.com">Lenovo</Vendor>
    <Model>W530</Model>
    <FormFactor>Laptop</FormFactor>
    <Historical>false</Historical>
    <NumaNodes>1</NumaNodes>
    <AvailableSockets>1</AvailableSockets>
    <Cpu>
      <Name>Intel Core i7-3740QM</Name>
      <FrequencyMHz>2700</FrequencyMHz>
      <TurboFrequencyMHz>3700</TurboFrequencyMHz>
      <TurboMode>Intel Turbo Boost Technology</TurboMode>
      <PopulatedSockets>1</PopulatedSockets>
      <Cores>4</Cores>
      <ActiveCores>4</ActiveCores>
      <CoresPerChip>4</CoresPerChip>
      <HardwareThreadsPerCore>2</HardwareThreadsPerCore>
      ...
```

Agenda

Conclusion

The Chauffeur framework can be used to build worklets to measure performance and energy efficiency. So far we've discussed the design of Chauffeur and how to configure it for some basic experiments.

In Part 2 we'll move on to the actual development of new worklets. We'll also cover how to build Chauffeur listeners to collect new data, and how to build custom reports. And we'll touch on how to plug new behavior in to Chauffeur for more advanced experiments.

Questions

Bibliography I

[1] Barroso, L., and Holzle, U.
The case for energy-proportional computing.
Computer 40, 12 (Dec. 2007), 33 –37.

[2] Fan, X., Weber, W.-D., and Barroso, L. A.
Power provisioning for a warehouse-sized computer.
In *Proceedings of the 34th annual international symposium on Computer architecture* (New York, NY, USA, 2007), ISCA '07, ACM, pp. 13–23.

[3] García-Castro, R., and Gómez-Pérez, A.
Benchmark suites for improving the RDF (S) importers and exporters of ontology development tools.
The Semantic Web: Research and Applications (2006), 155–169.

[4] Gustafson, J., and Snell, Q.
HINT: A new way to measure computer performance.
In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on* (1995), vol. 2, IEEE, pp. 392–401.

[5] Henning, J.
SPEC CPU2000: measuring CPU performance in the new millennium.
Computer 33, 7 (2000), 28–35.

Bibliography II

[6] Huppler, K.
The art of building a good benchmark.
Performance Evaluation and Benchmarking (2009), 18–30.

[7] Lange, K. D., Arnold, J. A., Block, H., Totura, N., Beckett, J., and Tricker, M. G.
Further implementation aspects of the Server Efficiency Rating Tool (SERT).
In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering* (New York, NY, USA, 2013), ICPE '13, ACM, pp. 349–360.

[8] Lange, K.-D., and Tricker, M. G.
The design and development of the Server Efficiency Rating Tool (SERT).
In *Proceedings of the second joint WOSP/SIPEW international conference on Performance engineering* (New York, NY, USA, 2011), ICPE '11, ACM, pp. 145–150.

[9] Lange, K.-D., Tricker, M. G., Arnold, J. A., Block, H., and Koopmann, C.
The implementation of the Server Efficiency Rating Tool.
In *Proceedings of the third joint WOSP/SIPEW international conference on performance engineering* (New York, NY, USA, 2012), ICPE '12, ACM, pp. 133–144.

Bibliography III

[10] Sim, S. E., Easterbrook, S., and Holt, R. C.
Using benchmarking to advance research: a challenge to software engineering.
In *Proceedings of the 25th International Conference on Software Engineering* (Washington, DC, USA, 2003), ICSE '03, IEEE Computer Society, pp. 74–83.

[11] Skadron, K., Martonosi, M., August, D., Hill, M., Lilja, D., and Pai, V.
Challenges in computer architecture evaluation.
Computer 36, 8 (Aug. 2003), 30 – 36.

[12] SPEC power committee.
`http://www.spec.org/power/`.

[13] SPEC power and performance benchmark methodology.
`http://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf`.

[14] SPECpower_ssj2008 benchmark.
`http://www.spec.org/power_ssj2008/`.

[15] SPEC Server Efficiency Rating Tool (SERT).
`http://www.spec.org/sert/`, Aug. 2012.

Bibliography IV

[16] Server Efficiency Rating Tool (SERT) design document.
http://www.spec.org/sert/docs/SERT-Design_Doc.pdf, Aug. 2012.

[17] Stefani, F., Macii, D., Moschitta, A., and Petri, D.
FFT benchmarking for digital signal processing technologies.
In *17th IMEKO World Congress* (2003).