SPEC – Power and Performance

Design Document

Control and Collect System

SPECpower_ssj2008

Standard Performance Evaluation Corporation

Table of Contents

SVN Revision: 1137

SVN Date: 2012/05/30 12:32:29

1 Introduction

This document provides an overview of the Control and Collection System commonly known by the acronym "CCS", an element of the SPECpower_ssj2008 benchmark software framework. Implementation details are provided for functions that are unique to the SPECpower benchmark framework.

Design requirements are described with some historical perspective and explanation of the design choices and/or tradeoffs. The design effort was intended to produce capabilities that will benefit the current and future implementations. CCS functions are briefly described. This document is intended to provide insight into the inner workings. How to set up and run the benchmark is described in detail in the SPECpower_ssj2008 user guide.

We advise that you read the "*SPECpower_ssj2008-Design_Overview*" at http://www.spec.org/power/docs/SPECpower_ssj2008-Design_Overview.pdf before this document.

To check for possible updates to this document, please see http://www.spec.org/power/docs/SPECpower_ssj2008-Design_ccs.pdf.

1.1 Overview

CCS is one of the essential elements of the SPEC performance and power measurement framework first delivered in December, 2007; the first SPECpower benchmark, SPECpower_ssj2008.

CCS is a multi-threaded Java application that controls and enables the coordinated collection of data from multiple data sources such as a workload running on a separate SUT (System Under Test), a power analyzer, and a temperature meter. Data from these sources is collected on a periodic basis, written to a log file and otherwise captured for reporting at the end of a benchmark run.

The interface from CCS to all data sources is through standard TCP/IP socket protocols, providing physical independence from the data sources such that they may be local or remote to the CCS host machine. This independence also enables those sources on separate machines to be running in different operating environments.

The SPECpower measurement framework, as shown in the diagram to the right, provides a set of capabilities adaptable to a wide range of computing topologies and operating environments.

Note that "Any OS" refers to those that support a Java Virtual Machine (JVM).

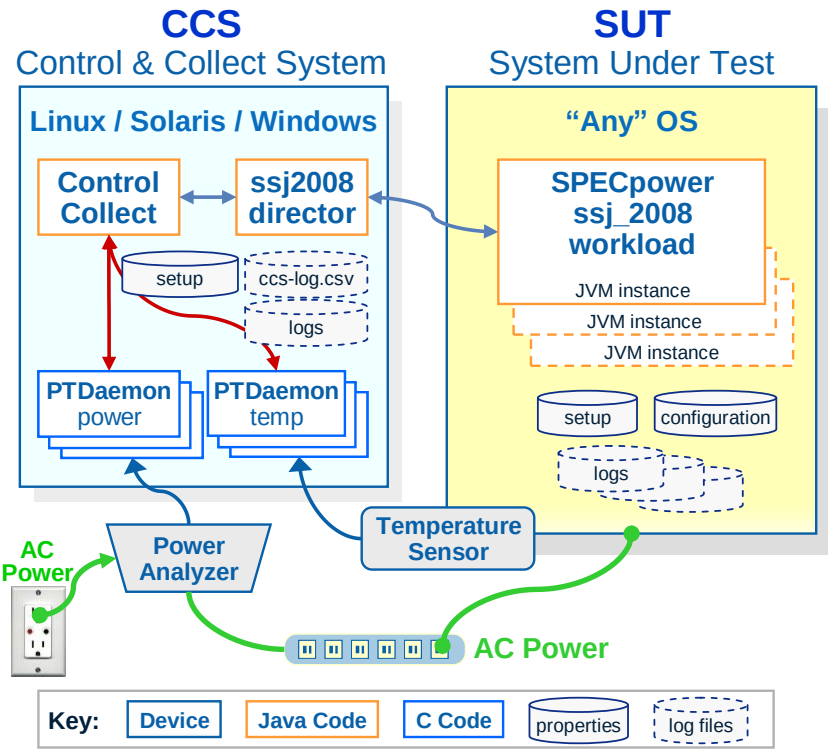

Figure 1 – Block Diagram

The software elements of the framework with their essential functions are:

- Control & Collect System (CCS)
  - Synchronize components startup
  - Collect data from sources
  - Consolidate collected data into per second log
    - power, performance, temp,
  - Summary per load level
  - Report raw data consolidation
- Power & Temperature Daemon (PTDaemon)
  - Connect to meter and control collection of data
  - Format, accumulate and pass data to CCS
- Director (ssj2008)
  - Workload (JVM) instance control
  - Synchronize of multiple workload instances
  - Track workload state
  - Consolidate results and pass to CCS
- Workload (ssj2008)
  - Schedule and execute transactions
  - Determine workload intervals and "state"
  - Report generation

A high value framework attribute is the control that brings consistency in measurements. The framework has also been designed for future use with features that provide extensibility, the ability to collect data from multiple sources on multiple machines that could in fact be in multiple locations.

1.2 Background and Goals

The SPECpower committee was chartered in January of 2006 to create a benchmark that would address the emerging need to measure power consumption and performance of server class computer systems under application-like loads. The intersection of performance and power has become an important attribute of computer systems, sometimes labeled efficiency. A standard method of measuring and reporting both would require a disciplined approach beyond what was then available on the open market.
The fact that one workload or benchmark could not represent the spectrum of server usage was generally accepted and therefore the SPECpower committee is determined to create more than one "benchmark" in more than one application segment.
This need generated a parallel requirement for a consistent power data collection method or framework that could at least be adaptable to a variety of workloads and at best to address the complexity of server hardware designs such as blade systems plus the potential need to include storage and networking devices. From these discussions on the breadth of future needs, the notion of a "measurement server" was born. Coupled with the SPEC imperative to bring consistency to performance measurement and reporting, the requirements list grew, the potential solutions were refined, and a compelling set of benefits surfaced.
The CCS design is the product of collaboration across the members of the SPECpower committee and its collective set of skills and broad experience.
The general needs and goals are described above. The more specific set of requirements of a control system are outlined below.

1.3 Design Basics - Requirements

Multiple Data Sources

At minimum performance and power data must be collected and logged. Provisions should be made for multiple power data sources such as a SUT and one or more storage arrays. Concurrently measuring multiple test systems was considered a future challenge and requirements specific to multiple power and test systems were continually considered in the design phase. All the data sources must be ready and multiple instances of the workload have to be synchronized for startup and during measurements.

Separation from the Workload

Control and collection software running on a system separate from the workload provides the flexibility to fix and enhance the collection software with no impact whatsoever on the workload. The “CCS” can improve, adapt to new topologies and remain performance neutral, and in fact independent of any given workload.

Standard Communication Protocols

The collection mechanisms will be required to interact with several data sources and run in at least three popular operating environments. The communication must use standard protocols to interoperate across networks and multiple operating system (OS) environments.

Measure Power on Control System

Power measurement is through devices known as Power Analyzers, many with the capability to communicate with computer systems. This dictated the need for software (then known as a “power daemon”) to control and collect data from various brands and models of power analyzers.
Collection of power data must not impact the performance of the workload, or worse, the workload should not impact timely collection of power data. For example, a workload running at peak performance will likely consume all available CPU cycles, and will inhibit the ability of the power code to collect at precisely 1 second intervals.
To interact with power analyzers and temperature sensors, SPEC created software must access, setup and write/read to platform I/O devices, for instance serial and USB ports. This low level platform and device interaction requires that the software be written in a language less portable than Java (C/C++ was chosen). This environment then dictates a separate binary for _every_ OS. Each distinct OS has different APIs and protocols for serial ports and multiple source code modules are necessary.

Portability and Platform Support

The number of operating environments potentially required to be supported by the power daemon became a concern. A SPEC workload must be as portable as possible across a very broad range of operating environments. The control-collect system will run well on a desktop or even a modern notebook platform and therefore could be readily limited to a select few operating environments; reducing the number of different code variations.
Current support for CCS is limited to Linux, Sun Solaris and Microsoft Windows.
The recommended minimum system is with processors running at clock speed of at least 1GHz, with 1GB of memory.

Control of data collection (and logging) intervals

Power data must be collected or sampled at a minimum of once per second. One second is the minimum sample rate of most performance tools. An accumulation of power measurements once per second, power in watts, can be easily converted to energy consumption where a watt second is equivalent to a joule. The sum of the one second samples yields energy consumed for that number of seconds.

1.4 Benefits of the Design

- Control Collect on Separate System
  - Zero performance impact on workload
  - Quick Integration of New Workloads
  - Changes do not impact workload
    - Can fix and enhance elements
    - Can add new power analyzers and temperature meters
- Multiple Measurement Devices
  - Can collect power from SUT and storage and/or other devices
  - Can collect temperature at inflow and outflow concurrently
  - Multiple concurrent SUTs, i.e. multiple measurement sessions
  - Future potential to measure Blades (multiple systems as one)
  - Future potential to measure across multiple OS images – i.e. Virtualization environments
- Reasonable Effort and Costs to Support the Framework

2 CCS Data Sources

2.1 The CCS Properties File

The CCS properties file provides an easy method to customize various aspects of the data control and collection side of the SPECpower framework.
The properties file is where data sources are declared, details of location and connection are defined, power analyzer setup can be defined, and finally a section where the particulars of the systems and devices (configuration) are entered for insertion in the final report(s).

Defining Data Sources

Data sources are defined in the CCS properties file along with hardware and software configuration specifics of the system that hosts the CCS and PTDaemon. Data source names are multi-tiered to fit the conventions used throughout the framework and to enable definition of multiple sources.
CCS connects to, synchronizes data collection, and captures measurements from all data sources defined in the CCS properties file (ccs.props). Data sources exchange data only with CCS –there is no data communication between data sources.
For consistent communications, each data source is designated as a data server and CCS as a client. CCS (the client) polls each data source (server) for the most current measurements.
When responses have arrived from all data sources, CCS initiates actions based on data received, typically combining the responses and generating a log entry.
As shown in the figure below, data sources are typically three:

1. ssj director
2. power analyzer
3. temperature sensor

It is possible to configure all data sources on the same system, all data sources on one separate system, or, even all data sources each on its own separate system.
The "ssj director" is the only performance data source, regardless of the number of workload instances or the number of SUTs measured. CCS communicates only with the director which is the consolidation point for workload information. The ssj director passes to CCS only the aggregate performance, again, regardless the number of active workload instances.

Global and Detail Data Sources -

Global data source entries are declared by these entries in the file "ccs.props":

- ccs.wkld = ssj_dir
- ccs.ptd = pwr1, temp1
- ccs.vam = vam1

There is one workload, "ssj_dir" (the ssj director), and two instances of the PTDaemon, one for "pwr1" and one for "temp1".

A detail data source definition must contain:

1. Element identifier
2. ccs
3. Data source identifier
4. wkld, ptd or vam
5. Data source unique name
6. user selected, e.g. "pwr1"
7. Data source device type
8. One of: ssj, poweranalyzer, tempsensor, or vam
9. Data source location
10. Hostname or IP address
11. Data source port number
12. TCP/IP port #

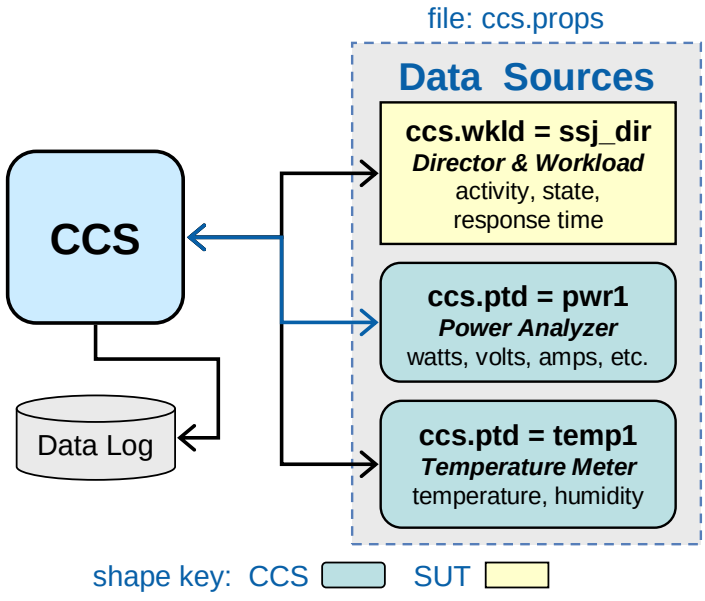

Figure 2 – CCS Data Sources

The data source unique name (#3

above) must match the name provided in the global data source declaration.

The bullet items below are global and detail data source definition samples for the most common three data sources:

- SPECpower ssj director

```
ccs.wkld = ssj_dir                  global data source
ccs.wkld.ssj_dir.type = SSJ         detail definition
ccs.wkld.ssj_dir.IP = localhost
ccs.wkld.ssj_dir.Port = 8886
```

- Power Analyzer and Temperature Sensor

```
ccs.ptd = pwr1, temp1               global data sources
ccs.ptd.pwr1.type = PowerAnalyzer   detail definition
ccs.ptd.pwr1.IP = localhost
ccs.ptd.pwr1.Port = 8888

ccs.ptd.temp1.type = TempSensor     detail definition
ccs.ptd.temp1.IP = localhost
ccs.ptd.temp1.Port = 8889
```

Note the use of TCP/IP port numbers. When the PTDaemon is run to collect power or temperature, a matching TCP/IP port number must be provided.

Adding Data Sources

Multiple data sources are easily added by defining new definitions that follow the established syntax. This example defines three power analyzers, the last (pwr3) is on a system separate from the CCS (network connected):

```
ccs.ptd = pwr1, pwr2, pwr3          global data sources
ccs.ptd.pwr1.type = PowerAnalyzer   first power analyzer
ccs.ptd.pwr1.IP = localhost
ccs.ptd.pwr1.Port = 8888            default port #

ccs.ptd.pwr2.type = PowerAnalyzer   second power analyzer
ccs.ptd.pwr2.IP = localhost
ccs.ptd.pwr2.Port = 8890            unique port #

ccs.ptd.pwr3.type = PowerAnalyzer   third power analyzer
ccs.ptd.pwr3.IP = 192.168.1.3       IP of remote system
ccs.ptd.pwr3.Port = 8891            unique port #
```

Removing or Disabling Data Sources

Data sources in the ccs.props file can be easily disabled and then later re-enabled by simply changing the global data source lines in the ccs.props file, or naming the ccs.props files and then changing the props file name in the runccs.bat file.
Disabling a data source in the ccs.props file is shown in this example:

```
ccs.ptd = pwr1,pwr2,pwr3            global data sources
```

Change the above line or comment it out as shown below.

```
# ccs.ptd = pwr1, pwr2, pwr3        global data sources
ccs.ptd = pwr1, pwr2                new global data
sources
```

Detail entries for the data sources, in this example “pwr3” may remain intact. They will be ignored in the absence of the global definition statement.

Power Analyzer Range Setting Overview

Measuring power consumption of server class platforms requires a power measurement device, commonly known as a power analyzer. There are many brands and models with wide ranges of capabilities. Each may internally operate quite differently one from another.
Most power analyzers accepted for the production of fully compliant results with the SPECpower_ssj2008 benchmark can measure a very wide range of current and voltage levels.
To assure the best possible accuracy of power measurements, the power analyzer must be properly set up for the conditions present in your measurement environment.
Auto-ranging is a capability built-in to most power analyzers and for some projects auto-ranging is an acceptable choice. It is important to recognize that auto-ranging is not the setting of first choice for power measurement of SPEC benchmarks.
Among the various meter brands and models, SPEC has found that delays can occur during auto-ranging. Delay duration can be short or sometimes several seconds. Power reading errors are sometimes generated while the device self-adjusts to a higher or lower range – in particular for current (amps). To avoid these issues, the SPEC benchmark developers have created methods to control and change settings during "quiet times" of the benchmark.
With the release of SPECpower_ssj2008 v1.1, range setting capabilities have been implemented to assure the highest degree of measurement accuracy.
CCS and the PTDaemon provide the capability to set current and voltage ranges on select power analyzers.
The ranges can be adjusted at each load level or just prior to a load that requires a range change.

Controlling Power Analyzer Range Settings

For the SPECpower_ssj2008 benchmark, ranges can be set for current (amps) and voltage. These ranges can be set or changed for each benchmark load level.
Through the range setting properties, CCS, during the "inter" stage between load levels, will send the designated settings to the PTDaemon, which if the the device supports programmatic setting, will direct the meter to set the range. A time duration of 10 seconds is, via the inter level space, allowed for the meter to complete the action.
In the ccs.props file, these two entries enable the entry of ranges for current and voltage.

```
ccs.ptd.pwr1.current_range_settings=auto
ccs.ptd.pwr1.voltage_range_settings=auto
```

When "auto" is specified, CCS will direct the PTDaemon to set the power analyzer during the initialization phase. No further range settings need be or will be performed.
If a range setting is left blank, or no setting property is present for that meter (or commented out), then existing meter settings will be used.
Some analyzers must be set up on the front panel. If range settings are entered at the analyzer, those range(s) must be recorded in the configuration section below.
Multiple load levels in the benchmark provide the possibility that power draw will increase or decrease across a power range of a particular power analyzer. Provisions have been implemented to potentially change a range prior to each of the 14 load levels.
Those 14 load levels are the default and minimum required for a compliant benchmark run. 3 calibration levels and 11 graduated load levels.
Ranges for the levels are specified by their position in a comma separated list that may contain one or more range values. The position of a range in the list is a 1:1 match to the load level. Spaces are ignored. Comma separators are required except after the last value.

The table and examples below provide more guidance.

cal1	cal2	cal3	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%	idle
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Example 1:
Auto for all levels


```
ptd.pwr1.current_range_settings = auto
```

Example 2:

4 amps for calibration and 100% levels, then 2 amps for all other load levels

```
ptd.pwr1.current_range_settings = 4, 4, 4, 4, 2
```

Example 3:

5 amps for levels down to 80%, then 2 amps for levels 70% and below:

```
ptd.pwr1.current_range_settings= 5, 5, 5, 5, 5, 5, 2
```

or,

```
ptd.pwr1.current_range_settings= 5, , , , , , 2
```

Given voltage is expected to be constant for the duration of the benchmark, the voltage_range need only be set for the first level; for example:

```
ccs.ptd.pwr1.voltage_range_settings = 120
```

Other CCS Properties (defined in CCS properties file)

CCS.run-id: Each run can be identified by a user provided run Identification code that will be logged to the ccs-log.csv file. This can be useful to record a number or text string for tracking purposes. The run-id does not appear in the SPECpower reports.

CCS timeout: The CCS timeout determines the number of seconds CCS will wait for any one of the defined data sources. 300 seconds is the default, usually more than enough time to start up the other benchmark elements. This value may be smaller or larger than the default depending on your needs.

CCS debug: Intended primarily for benchmark developers ccs.debug=TRUE will cause CCS to generate console output for nearly every interaction with data sources. The output may be captured by redirecting the output of CCS (>ccs-debug.txt) to your choice of device. Generating the debug output will slow down the execution of CCS (and any other application) therefore it is not advisable to enable unless a functional problem is suspected and you have guidance from an expert.

The defaults for these functions are:

```
ccs.debug = false
ccs.runId = 000-000-000
ccs.timeout = 300
```

The ccs.timeout value is in seconds.

Reserved Words

Reserved Words are those pre-defined by CCS and should not be changed, nor should user supplied terms replicate any reserved words.
This is the current list of reserved words:

ccs	IP	port	PowerAnalyzer	ptd	runID
SSJ	TempSensor	timeout	type	VAM.	wkld

Reserved words are not case sensitive.

Configuration Section – CCS.PROPS

For reporting purposes, it is necessary to document the configuration of the CCS system, the power analyzer, the temperature meter and the system(s) hosting the power and temperature devices. The last section of the ccs.props file provides name value pairs for the user to customize that information. Comments in-line describe the expected entries. See the sample properties file in Appendix A.

The “raw file”

Upon successful completion of a benchmark run, the CCS configuration properties will be merged with a similar set of properties from the SUT(s) side and the accumulated set of results from the measurements at the various load levels. SUT configuration is sourced from the files “SPECpower_ssj_config.props” and “SPECpower_ssj_config_sut.props”.

During the final phase of the benchmark, all the configuration and results information data is stored in a file known as the "raw file". It has a file extension of ".raw" and can be found in the SPECpower\ssj2008\results\ssj.nnn directory. The raw file serves as input to the reporter function and is vehicle used for submitting results to SPEC – should the user choose to publish results with SPEC.

3 CCS Execution Phases

The figure at right shows the various execution phases as CCS sets up communication links to the data sources, coordinates the start of measurements, begins data collection and logging, consolidates data from the SUT and the Control system, and finally calls the reporter to produce the SPECpower_ssj2008 text and html documents.

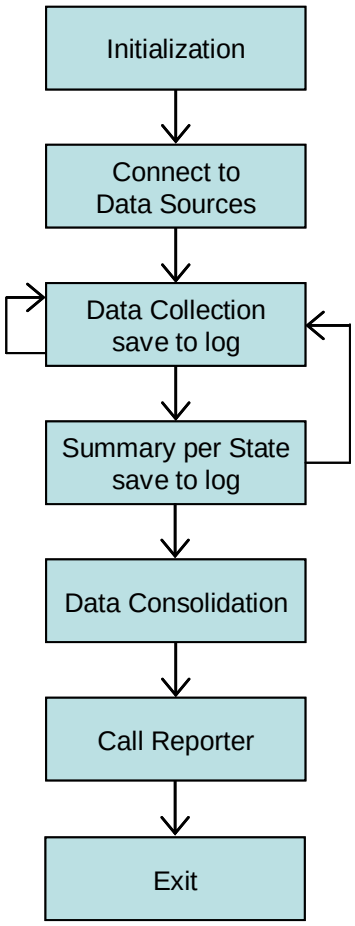

Figure 3 – CCS Phases

3.1 Connect to Data Sources

Based on the values entered for the global data source properties of "ccs.wkld=" and "ccs.ptd=", a source specific worker thread will be created for each entry.
The data source modules are initialized according to the corresponding sub-section values (from the properties file) with the type of data expected, the hostname or IP address and a TCP/IP port number. Examples of these basic entries are:

```
ccs.ptd.pwr1.type = PowerAnalyzer
ccs.ptd.pwr1.IP = localhost
ccs.ptd.pwr1.Port = 8888
```

Once all worker threads are created, TCP/IP socket connections will be established between the CCS host system and all data source applications on their respective host systems.

To accommodate the start order or method of starting (may be manual) for multiple data sources, plus the unknown latencies of their initialization phases, connection requests will repeat until either connected or the time out period expires (ccs.timeout = 300 seconds by default).
Once all data source have connected, CCS begins the exchange of credentials with the data source servers to ensure a valid connection, check compatibility and where appropriate, capture device specific information.

Data Collection

Based on the sampling rate, the CCS worker threads will poll the defined data sources and with successful, correct response, consolidate the workload performance, power and temperature and write into a log record, along with other pertinent information such as timestamps.

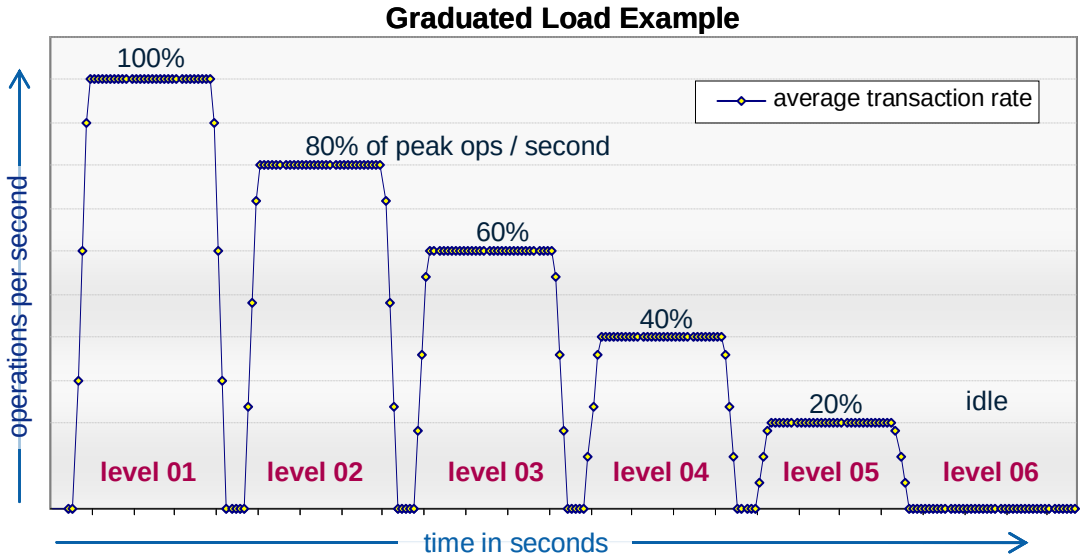

Figure 4 – Graduated Load Levels

For any given workload, and more importantly a graduated workload, it is essential that consistency mechanisms be developed to know precisely when a given phase starts and ends, and to define the phase during which both power and performance are measured for inclusion in the benchmark metrics.
This is implemented through the use of workload states that define each phase and are collected with the measurements, logged, and then used to define the "measurement intervals".

Workload States and State Changes

Accurate, consistent and repeatable measurement of both performance and power requires mechanisms to assure that a period known as the measurement interval is carefully defined and controlled.
This control is implemented through the definition of states which identify the various phases of the workload. The state is combined with the level number and load type, and becomes a field in the measurement record written to the detailed CCS log file.
For power and performance measurement, the SPECpower_ssj2008 workload defines eleven load levels ranging from 100% to 0% of the peak throughput. The 0% level is the idle measurement.

There are four distinct phases in any one load level, depicted in figure 5 below.

1. "inter" is a period between load levels. This method creates a break between load levels that eases post run visual analysis (the default value is 5 secs)
2. "ramp up" is a period of time that allows the application to reach a level of processing that will continue for the duration of (the default value is 30 secs)
3. "recording" is where data is collected and summarized in post-processing steps. This is the "measurement interval" (the default value is 240 secs)
4. "ramp down" is a period of time where the application will finish outstanding transactions before the load level ends (the default value is 30 secs)

State changes for one workload level are illustrated in the following chart.

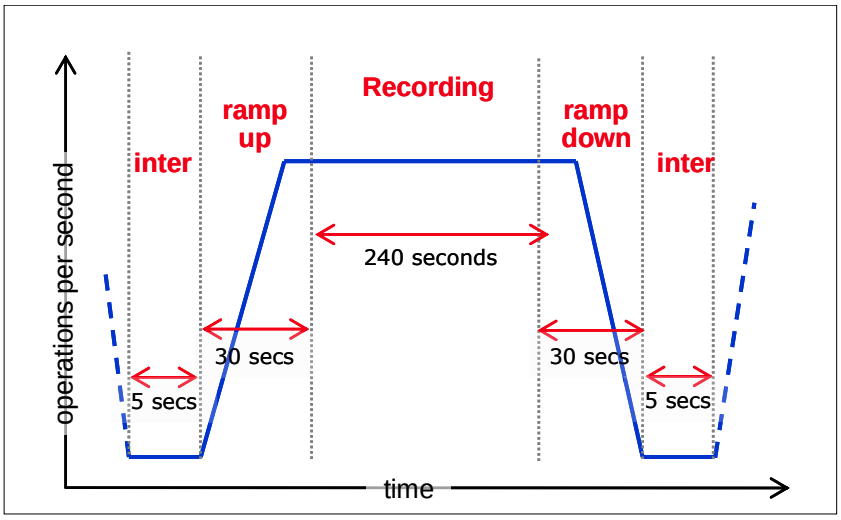

Figure 5 – States Diagram

State Code	Description	Default Duration	transactions recorded	power recorded	temperature recorded
inter	Inter-Level phase	5	No	Yes	Yes
_nnn_lvl_ru_	Load Level Ramp Up	30	No	Yes	Yes
_nnn_lvl_rc_	Load Level Recording	240	Yes	Yes	Yes
_nnn_lvl_rd	Load Level Ramp Down	30	No	Yes	Yes
_nnn_lvl_sum_	Load Level Summary	N/A	No	Yes	Yes

Table 1

The duration of each of the levels in figure 5 may be changed to accommodate a wide range of experiments. Entries that control workload behavior are found in the files “specpower_ssj.props” and “SPECpower_ssj_EXPERT.props”.
With each state change, CCS generates a “Summary” record written to the ccs-log.csv file with “###” as the first three characters. This unique tag for the summary records enables them to be quickly identified and/or extracted.
See the log file description below.

Sampling Rates

The frequency at which data is collected is the sampling rate, which for a compliant benchmark run is once every one second.
One sample per second was chosen for several reasons. Knowing the power consumption in watts every second is equivalent to measuring energy use in “watt seconds” or joules. Further, 1 second is the lowest sampling rate of common performance tools included with currently popular operating systems.
Notes on data collection: If a data source updates data at a rate faster than the CCS polling rate, the samples between will be lost, unless the data source averages its samples. Similarly, if a data source updates its data at a rate slower than the CCS polling rate, the same data values may be logged multiple times.

Data Logging

CCS log file is a second by second record of measurements from all defined data sources plus other information useful for validating the function of the benchmark and for viewing system performance, power and temperature behavior in great detail.
This collection of measured data can be extremely useful in “research mode” where the benchmark is used for various types of platform behavioral studies.

CCS collects data from the designated sources at the rate of one second and writes all data to a log file in a comma separated format. The log file name is of the form: ssj.nnnn.ccs-log.csv, where nnnn is a serial number created to make each run unique. Even unsuccessful runs will increment this number. This same serial number is in the parent directory name.
An example of a full log file name is: ssj.0299.ccs-log.csv

When a state change occurs, CCS collects "summary" data accumulated by the PTDaemon, combines with a performance summary from the ssj director, then writes a specially coded summary record to the log file. Summary records are identified by ### [wkld.ssj_dir] in the first field and have a unique format.

CCS Log file content

CCS log file is a second by second record of measurements from all defined data sources. This file is written in comma separated format to facilitate use with spreadsheet programs that enable viewing as rows and columns.
The log file content consists of two sections. The first is a listing of name-value pairs (properties) that are a record of the settings and environment for the benchmark run. The second section is a series of rows of measurement data from the data sources plus other information useful for validating the function of the benchmark results.

The current (minimum) contents of the CCS log file are described in the table below.

Column #	Column Heading	Description
1	CCS-ser	Log record serial number or, ### denoting a summary record
2	CCS_Time	Timestamp; when log record was written
3	action	Defines action source of log record. Currently one of REQ_TX_DATA or REQ_TX_SUMMARY
4	Run-ID	User defined identification tag (defined in ccs.props file)
5	wkld.ssj_dir.CCS_RT(ms)	Response time measured by CCS request to ssj director, Request to response, in milliseconds
6	wkld.ssj_dir.Wk-state	Workload State Code
7	wkld.ssj_dir.Trans	Transactions processed, cumulative within this load level.
8	wkld.ssj_dir.whse	Number of warehouses (equals number of ssj2008 threads)
9	wkld.ssj_dir.BatchRT	Batch response time in milliseconds; cumulative within a load level
10	wkld.ssj_dir.BatchCount	Batches processed, cumulative within this load level. One batch = 1,000 operations (not transactions)
11	wkld.ssj_dir.avg-txs	Only in summary records. Average transactions (ssj_ops) processed in the load level.
12	ptd.pwr1.CCS_RT(ms)	Response time measured by CCS request to PT Daemon, request to response, in milliseconds.
13	ptd.pwr1.Watts	Watts from PTDaemon. Average within sample rate, if meter samples >1 per second.
14	ptd.pwr1.Amp	Amperes from PTDaemon.
15	ptd.pwr1.Volt	Volts from PTDaemon
16	ptd.pwr1.PF	Power Factor from PTDaemon
17	ptd.pwr1.Uncertainty	Power readings uncertainty as calculated by the PTDaemon
18	ptd.pwr1.Notes	Only in summary records. String from PT Daemon summarizes power data over interval. A comma separated string that contains average, min, max, the number of samples and number of errors and number of good samples for watts, amps, volts and PF.
19	ptd.temp1.Temperature	Temperature from PTDaemon (temperature sensor), in degrees centigrade.
20	ptd.temp1.Humidity	Humidity from PTDaemon (temperature sensor), Only if provided by temperature sensor. Not required
21	ptd.temp1.Notes	Only in summary records. String from PT Daemon summarizes temperature/humidity over interval. A comma separated string that contains average, min, max, the number of samples, number of errors and number of good samples for temperature and humidity (humidity if available).
---	Other as defined	More columns may be present depending the number and names of data sources defined in the ccs.props file.

Table 2

When a second power analyzer is configured, columns will be added in the ccs-log.csv file with column heading names similar to those below:

```
ptd.pwr1.Watts (plus amps, volts, PF, etc. )
ptd.pwr2.Watts
ptd.pwr3.Watts
```

The same naming scheme is implemented for multiple temperature sensors.

Profiling Power with Performance

The information in the CCS log provides the unique opportunity to produce a “power profile” of the workload as shown in the chart below.

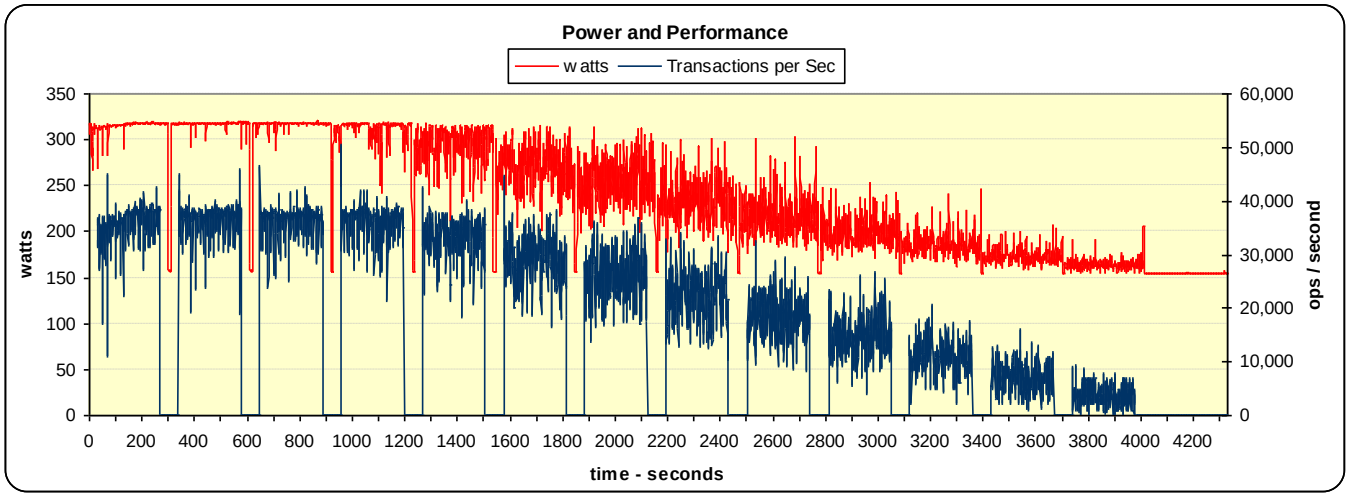

Figure 6 – Profile of Power and Performance

3.2 Data Consolidation

Upon completion of the workload, a “done” state will be recognized. Results of the workload run on the SUT and configuration specific data is transferred via the Director to the CCS system such that all standard benchmark results files are on one system in one directory.

In this “done” phase, data from the CCS collection and the workload produced logs for one or more JVM instances, plus information generated at run-time, is consolidated into what is knows as the “raw” file named as “ssj.nnnn.raw” where ‘nnnn” is the serial number of the run. For a standard (compliant) benchmark run, this file contains some 2,500 records.

During consolidation and at other points in processing the data, extensive validation of the results is done to give the benchmarker confidence that the run has executed successfully. Messages produced are warnings and do not effect the results of the run.

The raw file is the input to the reporter phase. The reporter code is contained within the ssj.jar file (which must reside on both the Controller system and the SUT).

3.3 Report Generation

The reporter is run automatically, and produces two types of human readable formatted reports, 1. a text file, and 2. html files with formatted content and charts known as the Full Disclosure Report (FDR). The primary report file name is of the form ssj.nnnn.html and .txt.

A JVM instance report will be produced for each of the one or more instances of the workload (multiple JVMs). Those file names are of the form ssj.nnnn.<instance #>.html and .txt.

Once the reporter produces the reports, CCS terminates normally.

The reporter function is covered in more detail in the “*SSJ Workload Design Document*” at http://www.spec.org/power/docs/SPECpower_ssj2008-Design_ssj.pdf

The Visual Activity Monitor - VAM

CCS supports the use of a graphic display program known as VAM, the SPEC Visual Activity Monitor. VAM is a separate application program that, given the proper configuration, can collect and display data from as many as three instances of CCS.

VAM was built largely for the purpose of demonstrating difference or sameness with the power, performance and performance per watt of two systems - separately running the SPECpower benchmark, either in a standard mode or with a customized (ssj_2008) properties file. Using the flexible options of the "research mode", specific load sequences can be run and repeated
VAM use is entirely optional. Use of VAM is neither required nor prohibited when producing "compliant" benchmark results.

3.4 VAM Setup

Setting up for VAM requires entries in the CCS.PROPS file. First a data source entry then the detail to identify the type, set the location and designate a TCP/IP port for CCS – VAM interaction.
The example below is the default in the ccs.props file. "ccs.vam" is commented out by default.
To enable use of VAM, simply remove the comment.

```
#ccs.vam = vam1                    Global
Data Source

ccs.vam.vam1.type = VAM
ccs.vam.vam1.IP = localhost    Host name or IP address
ccs.vam.vam1.Port = 8905       TCP/IP Port.  8905 is
default.
```

Corresponding entries in the vam.props file will look just like those in this example:

```
vam.datasource = 2USUT              Global Data Source
2USut.name = _Turboblaster 2U       Text for VAM display
2USut.color = red                   text color
2USut.port = 8905                   TCP/IP Port number
```

VAM can display data from up to three instances of CCS. Each CCS instance will have a CCS.PROPS file with entries similar to those above. The primary difference is the TCP/IP port number. That port number must be unique for each CCS instance.
See the "*Visual Activity Monitor (VAM)*" section in the User Guide at http://www.spec.org/power/docs/SPECpower_ssj2008-User_Guide.pdf for more information on setup and customizing the VAM display.

4 Appendix A – Sample Properties File: ccs.props

```
#######################################################
##
##     CCS property file - sample
##
#######################################################

#
#   "#" character in column 1 designates a comment line.
#

#######################################################
##
## Global Properties
##
#######################################################

#
#   runId is an arbitrary string to describe your test.
#   timeout value is in seconds, default is 300.
#

ccs.runId = 0000-0000-0000
ccs.timeout = 300

#######################################################
##
##  Data Source Properties
##
#######################################################

#
#   There are two categories of data sources.
#   1.    Workload
#   2.    Power and/or Temperature
#
#   1.    Workload - also defines the system upon which the
#         Director (in ssj.jar) runs.
#   1.1   Example data source: ccs.wkld = ssj_dir
#   1.2   "ssj_dir" is the user-supplied 'tag' that ties to
#         detail configuration entries.
#   1.3   Only one "ccs.wkld =" entry is expected
#
#   2.    Power and/or Temperature - The identification of a
#         particular power and/or temperature measurement device
#   2.1   Examples of define power analyzer
#   2.1.1 Example of single power data source: ccs.ptd = pwr1
#   2.1.2 Example of two power data sources: ccs.ptd = pwr1, pwr2
#   2.1.3 Multiple power analyzers may be specified, each
#         will require a data source entry (as above) and a
#         matching set of detail configuration entries.
#   2.1.4 In the examples above; "pwr1" and "pwr2" are user selected.
#   2.1.5 Other power analyzer results are logged in
#         ssj.nnnn.ccs-log.csv
#
#   2.2   Examples of temperature sensor definition
```



```
#    2.2.1 single temperature data source: ccs.ptd = tmp1
#    2.2.2 two temperature sources: ccs.ptd = tmp1, tmp2
#    2.2.3 Multiple temperature sensors may be specified, each
#          will require a data source entry and a
#          matching set of detail configuration entries.
#    2.2.4 In the examples above; "tmp1" and "tmp2" are user selected.
#    2.2.5 Other temperature sensor readings are logged in
#          ssj.nnnn.ccs-log.csv
#
#    2.3   Examples of power and temperature sensor definition
#    2.3.1 Example of one power analyzer and one temperature sensor
#          data source: ccs.ptd = pwr1, tmp1
#    2.3.2 Multiple power/temperature devices may be specified, each
#          will require a data source entry and a
#          matching set of detail configuration entries.
#    2.3.3 In the examples above; "pwr1" and "tmp1" are user selected.
#
#    For each data source entry, a detail section-set
#    is required to provide:
#     Device Type (reserved words, see below)
#     Tag (alphanumeric string of users choice, no spaces),
#     IP address (name or IP)
#     TCP/IP port # (must match port # of other program)
#
#    Example detail section-set:
#     ## configure connection to Power analyzer 1
#     ccs.ptd.pwr1.type = PowerAnalyzer
#     ccs.ptd.pwr1.IP = localhost
#     ccs.ptd.pwr1.Port = 8888
#
#    Notes on examples above:
#     "PowerAnalyzer" is the device type,
#     --> "type" is always a reserved word.
#     "pwr1" is the tag in the example above, user selected.
#
#    NOTE:
#     For a given data source type, all entries must be on
#      one line, for example, two power analyzers, two temp meters:
#      ccs.ptd = pwr1, pwr2, pwr3, temp1, temp2
#
#    Reserved Words:
#      ccs, SSJ, PowerAalyzer, TempSensor,
#      wkld, ptd, type, IP, Port, RunID, timeout, VAM.
#    Reserved words are not case sensitive.
#
######################################################

#
#    Data Source Entries;
#     Three are defined by default.
#      1. the workload "ssj_dir"
#      2. ptd sources (power and temperature)
#      3. VAM source (Visual Activity Monitor);
#         VAM is commented out by default.
#

ccs.wkld = ssj_dir
ccs.ptd = pwr1, temp1
#ccs.vam = vam1
```



```
######################################################
##
##  Detail Section-topology of the systems and software
##
######################################################

#
#   Workload Data Source  ################################
#
#   Configure the connection to SSJ Director.
#   If the director is run on a system other than the CCS
#   system, replace "localhost" with the IP address of the
#   "director" system.  If the SSJ director is started with
#   a non-standard port, change the port number here to match.
#

ccs.wkld.ssj_dir.type = SSJ
ccs.wkld.ssj_dir.IP = localhost
ccs.wkld.ssj_dir.Port = 8886

#
#   Power Analyzer 1 Data Source  ########################
#
#  If the PTDaemon is run on a system other than the CCS
#    system, replace "localhost" with the name or IP address
#    of that system.
#  If PTDaemon is started with a TCP/IP port other than the
#    default, change the port number below to match the
#    port number in the PTDaemon script file.
#

ccs.ptd.pwr1.type = PowerAnalyzer
ccs.ptd.pwr1.IP = localhost
ccs.ptd.pwr1.Port = 8888

#
# Power Analyzer Range Settings ##########################
#
# Select Power analyzers have multiple ranges that can be
#   set by the front panel or programmatically.

# Ranges can be set for current (amps) and voltage,
#  and can be set or changed for each benchmark load level.
#
# Range settings are sent to the power analyzer,
#   only if the device supports programmatic setting.
#
# A range may be specified for each load level.

# The range value should be the maximum of
#   the power analyzer's range setting.

# If a range setting is left blank,
#   or no setting property is
#   present for that meter (or commented out),
#   then existing meter settings will be used.

# Some analyzers must be set up on the front panel.
```



```
#   If range settings are entered at the analyzer,
#   those range(s) must be recorded in the
#   configuration section below.

# For compliant benchmark runs, and by default,
#   the SPECpower_ssj2008 benchmark generates
#   14 load levels;
#     3 calibration, and 11 graduated levels.

# Ranges for the levels are set by a comma separated
#   list that may contain one or more range values.
# The position of a range in the string is a 1:1
#   match to the load level.
#    Spaces are ignored.
# Comma separators are required except after the
#   last
```

5 Disclaimer

Product and service names mentioned herein may be the trademarks of their respective owners.

