Measuring and Benchmarking Power Consumption and Energy Efficiency

Jóakim von Kistowski
University of Würzburg
Germany
joakim.kistowski@uni-wuerzburg.de

Klaus-Dieter Lange
Hewlett Packard Enterprise
USA
klaus.lange@hpe.com

Jeremy A. Arnold
IBM Corporation
USA
arnoldje@us.ibm.com

Sanjay Sharma
Intel Corporation
USA
sanjay.sharma@intel.com

Johann Pais
Advanced Micro Devices
USA
johann.pais@amd.com

Hansfried Block
SPECpower Committee
Germany
hansfried.block@web.de

ABSTRACT

Energy efficiency is an important quality of computing systems. Researchers try to analyze, model, and predict the energy efficiency and power consumption of systems. Such research requires energy efficiency and power measurements, as well as measurement methodologies. Many such methodologies exist. However, they do not account for multiple load levels and workload combinations. In this paper, we introduce the SPEC power methodology and the tools implementing this methodology. We discuss the PTDaemon power measurement tool and the Chauffeur power benchmarking framework. We present the SPEC Server Efficiency Rating Tool (SERT), the workloads it contains and introduce the industry-standard compute efficiency benchmark SPECpower_ssj2008. Finally, we show some examples of how the SPEC power tools have been used in research so far.

CCS CONCEPTS

• **Hardware → Energy metering**; **Platform power issues**; **Enterprise level and data centers power issues**; • **Software and its engineering →** *Software performance*;

KEYWORDS

Power, Energy Efficiency, Performance, Benchmarking, Measurement, Load Level, SPEC

ACM Reference Format:
Jóakim von Kistowski, Klaus-Dieter Lange, Jeremy A. Arnold, Sanjay Sharma, Johann Pais, and Hansfried Block. 2018. Measuring and Benchmarking Power Consumption and Energy Efficiency. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3185768.3185775

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5629-9/18/04…$15.00
https://doi.org/10.1145/3185768.3185775

1 INTRODUCTION

Energy efficiency of servers has become a significant issue in recent times. In 2010, the U.S. Environmental Protection Agency (U.S. EPA) estimated that 3% of the entire energy consumption in the U.S. is caused by data center power draw [7]. According to a New York Times study from 2012, data centers worldwide consume power equivalent to the approximate output of 30 nuclear power plants [2].

Being able to accurately measure and benchmark the power consumption of servers is an important first step towards improving it. Measurement and testing methodologies can lead to selection criteria for more efficient servers, but can also be used in research and development. Researchers and system developers can use such measurement methodologies to test new power saving and management mechanisms. Software developers can use such a methodology to test and improve their workloads [19]. Most servers in modern day data centers are provisioned with additional capacity in order to be able to cope with unexpected bursts in load. This leads to an average load somewhere between 10% and 50% [3]. Any measurement methodology for energy efficiency must account for this by enabling measurement of power consumption and efficiency at multiple load-levels.

This paper introduces the SPEC Power methodology and the tools implementing this methodology. The methodology enables its users to measure the energy-efficiency of workloads at multiple load levels. It covers device setup, power and performance measurement, and efficiency rating. Power measurement is implemented in the SPEC PTDaemon, which communicates with power analyzers and temperature sensors. Workload dispatching, result collection, and test execution are handled by the Chauffeur framework. We describe how to use these tools to execute arbitrary workloads as part of the ChauffeurWDK framework for testing and research purposes. We also demonstrate the application of the methodology and framework by describing their use within the SPECpower_ssj2008 benchmark and SPEC Server Efficiency Rating Tool (SERT).

The remainder of this paper is structured as follows: Section 2 explains the challenges of power measurement and shows how our methodology and its PTDaemon implementation address these challenges. Next, Section 3 details the overall power and efficiency methodology and its implementation in the Chauffeur framework. Section 4 then describes the tools and benchmarks implementing the methodology. Finally, Section 5 concludes this paper.

2 MEASURING POWER

When the SPECpower Committee started working on an efficiency benchmark back in 2006 it could build on almost 20 years of benchmark development experience at SPEC. Creating numerous benchmarks for various application fields SPEC had defined basic characteristics for what constitutes a "good" benchmark. However, all SPEC benchmarks up to then had been focused on performance only. There was no precedence of characteristics for including power measurements into a benchmark to evaluate efficiency of a computer server. So, a completely new methodology including basic definitions for power measurements had to be developed for the new efficiency benchmark later known as SPECpower_ssj2008. This methodology became the foundation for including power measurements in several other benchmarks and tools.

2.1 Benchmark Power Measurement Characteristics

Efficiency benchmarks should adhere to the same principles as mere performance benchmarks, described in [16]. For efficiency benchmarks these principles must be extended to power measurements. The following benchmark characteristics are relevant for power measurements.

Reproducibility

To guarantee reproducibility of benchmark results consistency of measured power values is required, i.e. when the same benchmark is run multiple times under the same conditions the reported power values must be virtually the same with only minimal deviations allowed. Also, the major measurement parameters must be collected and reported, e.g. line voltage standard, current range settings, environmental temperature etc.

Fairness

Rules describing the power measurement setup and the scope of system components to be measured must be defined. Generally, all major power measurement conditions which are not under control of the benchmark program must be specified and required to be followed by benchmark users to ensure that measured power values are truly comparable. Measurement devices must provide certain properties which guarantee comparable power values when using different models. A qualification process must be implemented to assess these properties.

Verifiability

For verification of power value validity, the uncertainty of the measurement results must be calculated and reported. Preferably this is done automatically by software. Additional values besides power (Volts, Amps, Power Factor) should be measured and reported for potentially crosschecking the consistency of these values against reported power values.

Usability

Power analyzers are rather complex measurement devices with multiple configuration and setup options. To allow handling of power measurements even by unexperienced users supporting tools are needed for easy setup and hiding the measurement complexity. Such tools should also collect and aggregate power data automatically for use in benchmarks, especially for metric calculation.

Based on the above specification of power measurement characteristics we'll now describe the specific requirements for power analyzers and their controlling software for use in efficiency benchmarks. Finally, the implementation of these requirements in the Power and Temperature Daemon (PTDaemon) is presented.

2.2 Benchmark Power Requirements

For power measurements to be used in efficiency benchmarks highly accurate power analyzers must be utilized ensuring consistency of results. The Power and Performance Methodology [12] lists the following requirements for power analyzers:

- True RMS (Root Mean Square) power measurement ensuring that highly dynamic power changes in computer servers are recorded correctly
- Logging of measured values to an external device every second via communication interface to avoid forged results
- Uncertainty of less than 1% to ensure comparability of power values
- Regular calibration to a national standard every year is required for comparability of results
- Control of device configuration and recording of values from a program interface for ease of use and verification of correct settings
- Capability to handle amperage spikes (Crest factor) to enable correct readings under poor power conditions

In addition power analyzers must record power values continuously, not interrupted during data upload.

The controlling software must check the communication with the power analyzer regularly and detect invalid measurement states and irregular results. Because uncertainty calculations are complex and vary significantly between analyzers they must be implemented programmatically.

Also environmental temperature must be recorded because it has a major impact on server power consumption. This demands sensors with a communication interface for reporting of temparature values.

2.3 PTDaemon

SPEC's PTDaemon Design document [11] describes how benchmarks can leverage PTDaemon to offload power analyzer or temperature sensor control from a SUT (System under Test) to a Controller system. PTDaemon presents a common interface based on TCP-IP protocol that readily integrates into benchmark harnesses, while hiding different hardware interface protocols and behaviors from the benchmark software.

Figure 1 depicts the benchmark harness connecting to PTDaemon via TCP ports using a proprietary protocol to control devices and retrieve measurement data. Multiple IP/port combinations can be leveraged to control multiple devices.

PTDaemon can connect to multiple analyzer and sensor types via multiple protocols and interfaces specific to each device type, examples shown in Figure 1. The device type is specified by a parameter passed locally on the command line on initial invocation of the daemon.

The communication protocol between the SUT and PTDaemon is independent from the specific power measurement device type. This allows the benchmark to be developed independently of the measurement devices to be supported. Furthermore, SPEC provides

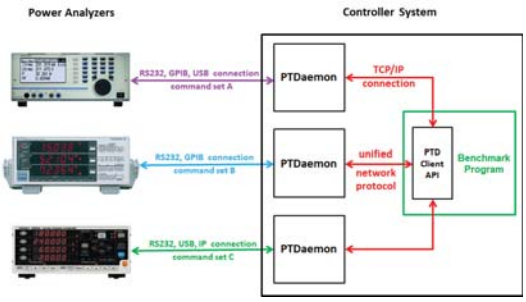

Figure 1: PTDaemon Power Measurement Infrastructure

a PTDaemon client API Java module which can be integrated into benchmarks for simplified communication between PTDaemon and the benchmark code. A generic setup for efficiency benchmarks including the components described here is shown in Figure 2.

PTDaemon is implemented using a main process that controls initialization and the network command interface and a separate thread that manages the power analyzer or temperature sensor. Some analyzers that do not operate with a standard command / response structure also require an additional thread to receive asynchronous data from the device.

SPEC's Power Measurement Setup Guide [13] provides an overview of the vastly different capabilities of accepted analyzers. E.g., the Yokogawa WT210 limited to RS232 & GPIB, requires a specific firmware version and only its direct input terminals are supported by PTDaemon, whereas the WT500 and WT1800 expand connectivity to Ethernet and USB. In addition, they can be used as 3-channel and 3-phase analyzers by PTDaemon, providing the measurement data of the individual channels and the sum channel. In order to facilitate measurement of 3-phase power sources, PTDaemon does some work behind the scenes to make a 3-phase load appear identical to a single-phase load from the user point of view.

To support different power analyzers, each supported device needs its own module in PTDaemon. PTDaemon is periodically updated to support new power analyzers, temperature sensors, and additional device features. SPEC members and licensees can use the Power Analyzer Acceptance Process [10] to add software support for new devices and submit tests to SPEC for review and possible inclusion in later PTDaemon releases.

SPEC's PTDaemon Accepted Measurement Devices web page[1] lists all accepted power analyzers & temperature sensors.

3 POWER AND ENERGY EFFICIENCY METHODOLOGY

The SPEC Power and Performance Benchmark Methodology was first introduced in [12] and has since been improved with each new SPEC power tool using it. It is designed for tools and benchmarks that measure the energy efficiency of servers at multiple load levels, enhancing their relevance [16], considering that modern servers spend most of their time in a CPU utilization range between 10% and 50% [3]. This sets benchmarks implementing the methodology apart from conventional performance benchmarks, such as SPEC CPU [4], and their respective methodologies which target maximum load and performance. To achieve workload execution at different load levels, the methodology includes a calibration step. This calibration determines the maximum transaction rate for any target workload on the SUT. The maximum transaction rate is measured by running as many transactions as possible. This calibrated rate is then set as the 100% load level for all consecutive runs. For each target load level (e.g., 100%, 75%, 50%, 25%), we calculate the target transaction rate and derive the corresponding mean time from the start of one transaction to the start of the next transaction. During measurement, these delays are randomized using an exponential distribution that statistically converges to the desired transaction rate. As a result, lower target loads consist of short bursts of activity separated by periods of inactivity.

A key recommendation in the methodology that should be explicitly noted is that intermediate load levels are defined as a percentage of the maximum application throughput, and not a target CPU utilization. While application throughput and CPU utilization are certainly correlated, the exact calculation of CPU utilization may vary on different processors and operating systems, and the definition of CPU utilization can be complicated in modern processors with multiple threads per core and cores per chip, superscalar designs, and out of order and speculative execution. Defining load levels as a percentage of maximum application throughput provides more accurate results that can be compared across a variety of different platforms.

3.1 Device Setup

The methodology requires two physical systems at minimum. The measurements are controlled by a *controller* system. This system runs the harness, the reporter, and interfaces with the external measurement devices. The SUT, on the other hand, executes the workloads under consideration. Controller and host communicate using a network interface for synchronization, load scheduling, and result collection. The systems and the software on the systems is illustrated in Fig. 2.

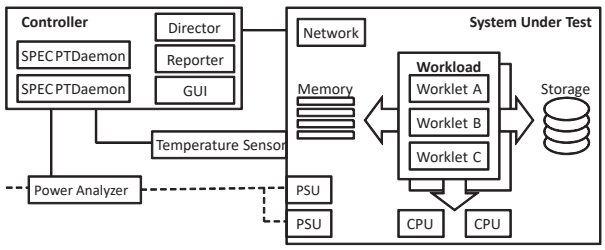

Figure 2: Device and software setup

The *director* component on the controller communicates with the *host* running on the SUT. The host spawns separate *clients* for each logical CPU (logical processor, also called hyperthreading or SMT unit). These clients are bound to their logical and physical CPU using an *affinity provider*. The transactional workload is executed sequentially on the clients. Parallelism can be achieved by running multiple clients concurrently. Clients can also choose to parallelize using regular multi-threading.

[1] https://www.spec.org/power/docs/SPECpower-Device_List.html

Our methodology requires at least one power analyzer and one temperature sensor. The power analyzer measures the power consumption of the entire SUT, while the temperature sensor verifies the validity of measurements by assuring that all experiments are conducted under similar environmental conditions. This reduces power measurement inaccuracies caused by varying leakage power, which can be a result of varying environmental temperatures.

3.2 Workloads and Worklets

We refer to the work executed on the SUT as *worklet*. Multiple worklets can be grouped in a *Workload*. Single application tests use a single worklet in a single workload with a one-to-one mapping. Measurement suites, such as the SPEC SERT, can contain multiple worklets, grouped into workloads.

A worklet is a transactional (mini-)workload. This means that it consists of relatively small-scale work-units (called transactions), for which a beginning and end can be measured. The transaction rate (throughput) achieved by the worklet serves as the primary performance metric.

3.3 Phases and Intervals

Worklet execution is split into three phases: Warmup, calibration, and measurement. The warmup phase executes the workload without recording any measurements in order to negate the influence of potential transient effects. Calibration executes the worklet in order to determine the maximum load level. Finally, the measurement phase performs the actual measurement. Each phase contains one or multiple intervals, which are the points in time at which the phase-specific work is excuted.

Each interval contains a pre-measurement and a post-measurement period. In these periods the worklets are already being executed at the target load level, yet no measurements are recorded. Both periods serve the purpose of achieving a stable state for the power and performance measurements. The duration of these periods is worklet-dependent, but we recommend a default duration of 15 seconds. The measurement period is executed between these two periods. Per default, the measurement is executed for a duration of 120 seconds. The 15 and 120 second durations are default values and can be adjusted depending on workload. E.g., the CPU worklets of the SERT 2.0 show little performance variation, allowing a measurement duration for as low as 30 seconds. In this time all transactions are logged and power measurements and temperature measurements are collected by the controller system at one second intervals. An interval's power consumption is the average of the per-second values.

To reduce intervals at different load levels or using different workloads from influencing one another, the system is left idling for 10 seconds in between each interval. Intervals within a phase are organized in sequences, which define the order in which the phase's intervals are executed. The most common sequence used in the measurement phase is the graduated measurement sequence. It executes the intervals at gradually diminishing target transaction rates. Transaction rates are determined using the result of the calibration phase and the target load level percentage. The measurement intervals and sequences for a worklet with the default interval measurement duration of 120 seconds are illustrated in Figure 3.

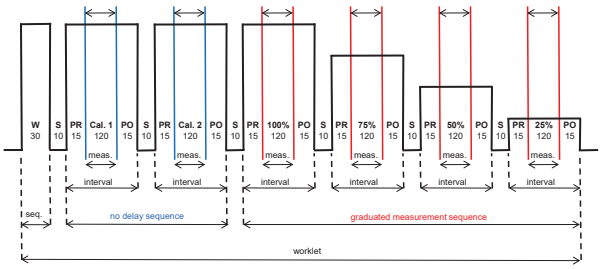

Figure 3: Example of measurement phases and intervals

We collect throughput and power consumption during each interval and calculate their respective averages at interval end. Using these averages, we can compute the per-interval energy efficiency by dividing throughput by power (Eq. 1)

$$interval\_efficiency = \frac{throughput}{power} \left[\frac{s^{-1}}{W} = \frac{1}{J} \right] \quad (1)$$

4 SPEC TOOLS AND BENCHMARKS IMPLEMENTING THE METHODOLOGY

This section is intended to round out the tutorial by presenting the Chauffeur Framework, ChauffeurWDK and already existing workloads that are easily executed with the SPEC power methodology as part of the SPEC SERT and SPEC power benchmarks. We present the workloads, their intricacies, and their applications in power research so far.

4.1 Chauffeur

One of the key features of Chauffeur compared to other benchmark frameworks is support for measuring power consumption under a variety of loads. The SPEC PTDaemon interfaces with power analyzers and temperature sensors, and Chauffeur makes the necessary calls to PTDaemon in order to collect the data for the appropriate time intervals. The resulting data is stored together with the runtime performance information so that the data does not have to be correlated after the test run is completed. The ability to run at multiple utilizations is also an inherent property of Chauffeur. This support is based on the same principles used in SPECpower_ssj2008 to vary the load by first determining the maximum transaction throughput during a calibration process, and then scheduling transactions with appropriate delays to drive the system at lower levels of utilization. This support is implemented generically in Chauffeur and can be applied to any workload that is composed of a series of short-running transactions.

Scalable

The Chauffeur framework is inherently multi-process and multi-threaded, providing scalability across a wide range of servers. Multiple-node runs are also supported, enabling Chauffeur-based workloads to run across multiple blade servers.

Chauffeur's Director (the component that instructs the Host JVM to start executing the workload) normally runs on a system other

than the SUT, and is usually collocated with the SPEC PTDaemon. It can also be collocated with an external GUI that eases Chauffeur configuration. It communicates with a host process that runs on each system that is being measured. The Chauffeur Host will automatically launch one or more Client JVMs as needed, using platform-specific affinity commands.

Chauffeur can adjust heap settings for each Client JVM based on the amount of memory in the system and the number of Clients being used. The algorithms for calculating heap sizes can be adjusted for individual Worklets; for example, the SERT, which uses Chauffeur, uses nearly all available memory when running the memory Worklets, but normally uses only 256 MB per logical processor for CPU Worklets.

Ease of Use

Benchmark configurations can be complex, particularly when power measurements are involved. Chauffeur includes a number of features intended to simplify testing as much as possible.

A particular challenge for industry standard benchmarks and tools like SERT is reporting a complete and accurate description of the system configuration. To assist users in this process, Chauffeur supports automatic collection of system configuration data. This data is included in the results file, and also made available to the SERT GUI to allow users to review and edit the information.

Chauffeur also supports automatic validation of results, both at runtime and for completed results. These validation checks can confirm that the configuration is valid, that transactions did not fail, that power analyzer data met the requirements specified by the run rules, and various other requirements. The current validation checks in Chauffeur are defined for the SERT, but the framework is generic to allow checks to be added, changed, or removed for other workloads. This validation gives users confidence that the workload is configured and running properly and that the results are accurate.

Portable

Chauffeur is implemented primarily in Java to simplify portability across different systems. The SERT currently supports 64-bit Windows and Linux on x86 processors, and AIX on the Power architecture. Limited testing has also been performed on other platforms with minimal difficulty.

Platform-specific code is included in two areas of Chauffeur: System Configuration Discovery and Affinity. In both cases Chauffeur will continue to run on platforms without explicit support. Future versions of Chauffeur can be extended easily by adding support for these features on other platforms.

Although the framework is written in Java, Worklets can make use of other programming languages, as the Storage Worklets in the SERT do. Communication between the Chauffeur Host and Clients is intentionally language-neutral, enabling a possible future native implementation of the Chauffeur Client.

Flexible

Chauffeur implements the core functionality required for the SERT. It also offers flexibility for changing the runtime behavior, either for research purposes or for future Chauffeur-based benchmarks. Many aspects of Chauffeur behavior can be changed via configuration files, without modifications to Chauffeur itself. For example, a virtualization benchmark may require the ability to run different virtual servers at different utilization levels in order to mimic dynamic heterogeneous usage patterns. Chauffeur does not currently support this usage model, but a custom implementation of a "Sequence" could be implemented and plugged in through the configuration file.

It is often desirable to collect a variety of different types of data during a benchmark run, particularly for research and development purposes. Chauffeur provides a Listener interface that allows custom data collection to be performed without modifications to Chauffeur. These listeners are notified at various stages of the run, such as at the beginning and at the end of each measurement interval. Listeners can launch platform-specific tools or collect data that is not directly supported by Chauffeur. Data obtained by listeners can be included in the main Chauffeur results file so it does not need to be correlated after the run completes.

The Chauffeur Reporter is also designed for flexibility. It reads the XML-based results file from a Chauffeur run and produces HTML or plain-text reports. The contents of these reports are defined using an XSL transform, which allows changes to be made to the reports without code changes to Chauffeur. Advanced users can create their own reports, or generate output in comma-separated values (CSV) format for easy import into spreadsheets or statistical packages.

4.2 Using the Chauffeur WDK to measure energy efficiency of arbitrary workloads

Using a framework such as the Chauffeur Worklet Development Kit (WDK) simplifies development of an energy efficiency benchmark by implementing behavior that is common to all benchmarks so that the developer can focus on the specific business logic being tested. In particular, the Chauffeur WDK can automatically run a transactional workload at a variety of different loads, by inserting appropriate delays between transactions. This results in a controlled and repeatable load which is also highly configurable.

Developing a new Chauffeur worklet involves some boilerplate code, but the bulk of development is focused on defining two components: one or more **Transactions** and a **User** [1]. The Transaction implements the business logic that the worklet is testing. The Transaction interface has two key methods: `generateInput` and `process`. The `generateInput` method creates (usually randomized) input data for the transaction. The `process` method receives this input and implements the logic of the transaction to produce some result. For some simple transactions, this code may be implemented directly in the Transaction class. When integrating existing workloads into Chauffeur, it is common for the Transaction implementation to be a thin layer that calls into the existing code. In some cases, the Chauffeur worklet implementation could just be a driver for an external application, such as an HTTP client for a web application which could be running on a remote system.

The **User** implementation provides access to state information. Some worklets may not require any state information and can use a basic User implementation. Other worklets may implement the User to provide access to in-memory data structures, a database, or other data that can be held from one transaction to the next.

Chauffeur also supports implementation of a co-mingled worklet where the individual component worklets run simultaneously rather than consecutively. This introduces more realistic task switching, which is especially useful for IO load simulation. E.g., a co-mingled

worklet can be implemented by creating a new worklet that consists of transactions taken from other worklets, e.g., a processor-intensive transaction and a disk access transaction. As in any other worklet, these transactions could be specified to execute in whatever ratios are desired, e.g., 70% processor intensive, 20% disk reads, 10% disk writes.

4.3 The SPEC SERT Suite and its Workloads

The use of multiple power analyzers and temperature sensors is supported by the SERT in order to measure a large scope of system configurations. The most basic SERT measurement configuration requires one **power analyzer**, one **temperature sensor**, a **SUT**, and a **Controller** system.

SERT uses *Chauffeur* (see Section 4.1) as its harness. Chauffeur controls the software installed on the SUT **Controller** system. Chauffeur also handles the logistical side of measuring and recording the power consumption and inlet temperature of the SUT.

The SUT gets instructed by the **Director** (Chauffeur instance) to execute the suite, which is comprised of a set of **workloads**. Each workload consists of a set of **Worklets**, which exercise the SUT while Chauffeur collects the power and temperature data. The Worklets are the actual code designed to stress a specific system resource or resources, such as the CPU, memory, or storage IO.

Each power analyzer and temperature sensor interacts with its dedicated instance of the **SPEC PTDaemon**, which gathers their readings while the Worklets are executed.

The **Reporter**, executed after all measurements phases are completed, compiles all of the environmental, power, and performance data for a complete test run into an easy-to-read HTML report as well as an extensible markup language (XML) report; the HTML report includes a graphical visualization of the results. The deployment of these components is illustrated in Fig. 2.

4.3.1 System Configuration Discovery. One of the largest challenges in benchmark and rating tool submissions is correctly identifying and capturing all of the characteristics of the SUT. Unintentional errors readily occur when collecting identification details of the various hardware components and recording the details into the benchmark report. Formatting errors overlooked while entering data may only become obvious once the final report is ready for submission. Correcting such errors and oversights in the final report can be cumbersome. The SERT addresses these issues with an automated hardware discovery process and an easy-to-use GUI workflow that assists the user in generating high quality accurate reports. The GUI therefore reduces the burden of test configuration, execution and report editing so that the user can focus on obtaining results.

The relationship between the main components (Hardware Discovery, Test Environment Editor, and Preview Report) of the SERT discovery workflow is shown in Figure 4.

4.3.2 SERT: Definition and Execution. The SERT is composed of a suite of worklets, each of which exercises the SUT in a specific way. For example, the LU worklet performs CPU-intensive matrix decomposition, while the Sequential IO worklet performs sequential IO operations on all storage devices included in the SUT. These worklets are grouped into workloads according to the

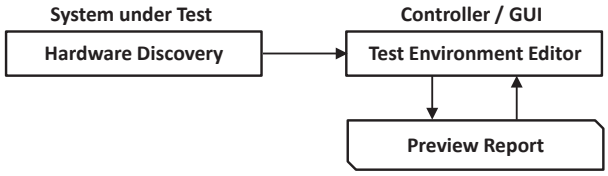

Figure 4: Discovery workflow

component of the SUT that they are intended to stress: CPU, memory, and storage IO. In addition, a Combined workload consists of application-focused worklets that stress the components of the SUT in a more balanced manner. Figure 5 shows the relationship between the overall suite, workloads, and worklets.

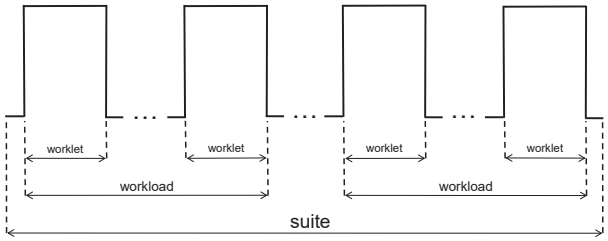

Figure 5: Suite overview

During a SERT run, each of the worklets is executed consecutively. Each worklet is run in its own set of JVMs or processes in order to minimize interactions between different worklets. Chauffeur automatically launches these client JVMs and coordinates the work among them. Most worklets use multiple client JVMs on the SUT and Chauffeur automatically uses operating system-specific affinity commands to pin each JVM to specific processors in order to avoid artificial limits to scaling. In this context, "client JVM" refers to the client side of a client-server communication pattern and is the JVM that does all of the real work. JVM command-line options are set by Chauffeur (with configurable overrides), allowing for self-tuning of heap sizes and ensuring that the command-line options are reported accurately.

The use of multiple JVMs for running a single worklet is primarily to avoid software bottlenecks (whether in the JVM implementation or in the SERT worklets) from limiting scalability since SERT is intended primarily for measuring the energy efficiency of the hardware and not the software stack. SERT is quite capable of running each worklet in a single JVM, but performance results are likely to be better when using multiple JVMs, e.g., each JVM can be affinitized to a specific processor and therefore all memory accesses will be local to that processor.

Most worklets use a "Graduated Measurement" execution sequence (Example with three load levels in Figure 3). These worklets begin by executing a short warm-up phase (30 sec.), and then run two calibration phases to automatically determine the maximum throughput each worklet can run on the SUT. The duration of the calibration phase depends on the worklet in question. CPU worklets are calibrated for 30 seconds. Durations for other worklets vary.

Next, the worklet runs at multiple load levels, such as 100%, 75%, 50% and 25% of the maximum throughput, generating independent scores for each load level. Each interval of execution includes a pre-measurement (15 sec.) and post-measurement (10 sec.) period in addition to the actual measurement period; each of these periods run for a fixed amount of time. Between each load level a sleep phase (10 sec.) is observed.

Performance and power are reported for the measurement phase (120 sec. or 30 sec.) only. This ensures that the worklet is running at steady-state in all client JVMs at the time performance and power are measured.

An alternative "Fixed Iteration" execution sequence (Figure 6) is used for worklets that do not support multiple load levels. These worklets run a fixed number of test iterations rather than for a fixed period of time. They optionally include some number of pre- and post-measurement iterations, similar to the pre- and post-measurement periods in a Graduated Measurement sequence.

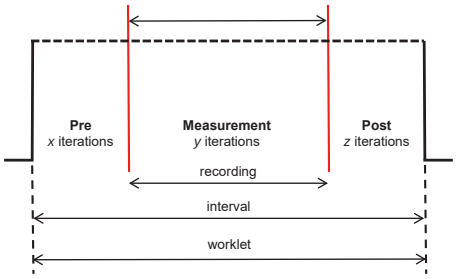

Figure 6: Phases of a Fixed Iteration Sequence

All of the time intervals are configurable (which is usually not intended for standard test runs, but useful for researchers) and the interval lengths can be adjusted separately for each worklet.

The results from individual worklets are reported individually and can also be combined into higher-level metrics at the workload level to summarize the performance for a particular subcomponent.

4.3.3 Worklets. SERT worklets were designed under a set of public guidelines [12] to ensure consistent results across a broad spectrum of technologies. For example, each workload must automatically calibrate itself to report the maximum performance available in that specific hardware configuration, and must then be adjustable to target load levels from 100-0% of the maximum performance. Each worklet also needs to scale with the available hardware resources which the execution model deemed "important", e.g., a CPU worklet needs to scale with the number of processors, cores, hardware threads and the clock frequency.

The SERT Design Document [14] offers a detailed breakdown of what each worklet does and how it works. 12 worklets are included in the SERT 2. They are categorized in Table 1.

The workloads can be summarized as:

CPU: Data compression, encryption/decryption, complex number arithmetic, matrix factorization, floating point array manipulation, sorting algorithm, string manipulation, and a CPU-heavy workload derived from ssj2008, which simulates an on-line Transaction Processing workload;

Table 1: SERT Worklets

Workload	Worklet	Sequence Execution	Metric
CPU	Compress	Graduated	Transactions/sec
	CryptoAES	Graduated	Transactions/sec
	SOR	Graduated	Transactions/sec
	SORT	Graduated	Transactions/sec
	SHA256	Graduated	Transactions/sec
	LU	Graduated	Transactions/sec
	SSJ	Graduated	Transactions/sec
Storage IO	Random	Graduated	Transactions/sec
	Sequential	Graduated	Transactions/sec
Memory	Capacity3	Graduated	Transactions/sec * sqrt(data-store-size)
	Flood3	Fixed	Memory bandwidth (GB/sec)
Idle	Active Idle	N/A	N/A

Storage IO: Two individual transaction pairs combining sequential and random read/write;

Memory: XML document manipulation and validation using pre-computed and cached data lookup, and array manipulation with read/write operations across four major classes of data transformation;

Active Idle: A steady state in which the server is ready to execute any worklet but is not actually doing so, leading to a measure of efficiency for a fully functional but otherwise idle state.

There are no worklets related to Network IO, which is handled by a "configuration modifier" that simulates the steady state efficiency of a network device. After testing a variety of network interface cards (NICs) across a range of workloads it was observed that the power consumption of the actual devices approximated very closely to a constant (including in the case of NICs that perform offloading from the host processor), with CPU and memory power consumption being the biggest factors influencing overall system efficiency. Combined with the extensive set of external hardware required to effectively test network bandwidth and performance, it was agreed with the EPA that a modifier would be applied to simulate the network IO contribution to overall server efficiency.

4.4 SPECpower_ssj2008

SPECpower_ssj2008 [6] is the first industry-standard SPEC benchmark that evaluates the power and performance characteristics of volume server-class and multi-node class computers. In a sense, it is the predecessor to Chauffeur and the SERT, which build upon the design and experiences from ssj2008. The general approach [12] is to compare measured performance with measured power consumption. An initial requirement was to include power measurement data of a system running at different target load levels to reflect the fact that data center server systems run at different target loads relative to maximum throughput.

4.4.1 Configuration Overview. The simplest SPECpower_ssj2008 hardware measurement configuration requires four main hardware

components: one **Power Analyzer**, one **Temperature sensor**, a **SUT** and the **Controller**. SPECpower_ssj2008 is composed of several elements; with the first is the test Control and Collect System (**CCS**), which handles the logistical side of measuring and recording the power consumption and inlet temperature of the SUT. It also controls the software installed on both the SUT and Controller, communicating via the TCP/IP transport protocol.

CCS communicates with the **Director**, which instructs the SUT to execute the **workload** while CCS collects the power and temperature data. The temperature sensor must be placed no more than 50mm in front of (upwind of) the main airflow inlet of the SUT. SPECpower_ssj2008 will measure the inlet temperature of the SUT and marks the results "valid" only if the temperature measured is 20°C or higher, in order to discourage the "gaming" of the test environment. A stable temperature value is not required during warm-up or measurement phases. The power analyzer must be located between the AC Line Voltage Source and the SUT. Both are connected to the Controller via their device specific interfaces, as shown in Figure 7. Each analyzer and sensor interacts with its dedicated instance of the **SPEC PTDaemon**, which gathers their readings while the worklets are executed.

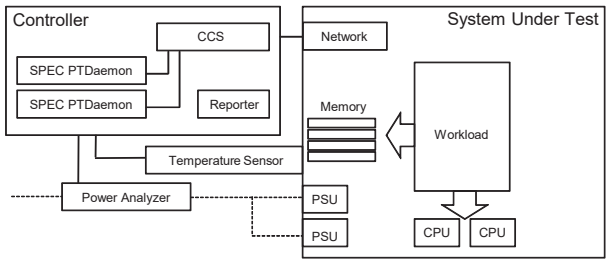

Figure 7: SPECpower_ssj2008 overview

The **Reporter**, executed after all measurements phases are completed, compiles all of the environmental, power, and performance data for a complete test run into an easy to read report. The reports are generated in HTML and plain text. The HTML report includes a graphical visualization of the results.

4.4.2 Load Levels. SPEC recognized that many servers include technologies to reduce the power consumption when the system is running at low utilizations. Since most systems spend much of their time running at less than full capacity, SPEC developed a methodology which advocated measuring performance and power consumption at a variety of system loads.

An SSJ run consists of two main phases: Calibration and running at a series of Target Loads. The calibration phase is used to determine the maximum throughput that a system is capable of sustaining. Once this calibrated throughput is established, the system runs at a series of target loads. Each load runs at some percentage of the calibrated throughput. For compliant SPECpower_ssj20008 runs, the sequence of load levels decreases from 100% to 0% in increments of 10% (Figure 8). Measuring the points in decreasing order limits the change in load to 10% at each level, resulting in a more stable power measurement. Using increasing order would have resulted in a jump from 100% to 10% moving from the final calibration interval to the first target load and another jump from 100% to Active Idle at the end of the run.

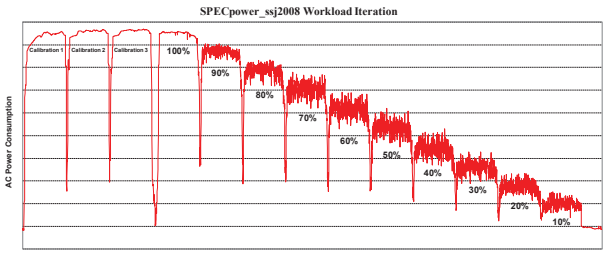

Figure 8: Load Levels

For each Target Load Level (e.g., 90%, 80%), the test harness calculates the target transaction rate and the corresponding mean time from the start of one transaction to the start of the next transaction. During the measurement interval, randomized delays are inserted into the worklet execution; these delays follow an exponential distribution that statistically converges to the desired transaction rate, as described in Section 3. As a result, lower target loads consist of short bursts of activity separated by periods of inactivity. Figure 9 shows examples of the utilization of a single processor thread running at 67% and 33% target loads using this technique.

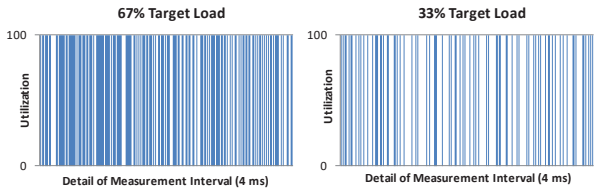

Figure 9: Load Distributions at different Target Loads

4.4.3 Driving Server Energy Efficiency. The game changing innovations, flexible design, cross-platform implementation, and automatic power measurement harness made SPECpower_ssj2008 the first industry standard benchmark to measure the power and performance characteristics of volume server-class compute-equipment.

SPECpower_ssj2008 was first released in December 2007, most recently updated in 2011, and continues to drive innovations in server efficiency. Rather than trying to approximate all the typical workloads used across organizations SPECpower_ssj2008 is focused on transactional server-side Java workloads that simulate a warehouse-based customer ordering, supply and replenishment model. This synthetic workload exercises many aspects of commercially-available Java implementations, together with the underlying server hardware including processors (with support for multiple cores per processor), memory hierarchies (including caches) and the system Symmetric Multiprocessing (SMP) scalability.

4.5 Examples SPEC Power Tool Use in Research

The SPEC power tools have been used in multiple works of research in the past. In this section, we list some of these works as examples of how the tools can be used. In general, we observe three types of SPEC power tool use: *1)* Use of SSJ and SERT results for research, *2)* Use of modified SSJ or SERT for research, *3)* Use of frameworks or measurement tools for testing with new workloads or load types.

SERT and SSJ results have been used in multiple works for evaluation, analysis or as part of new research contributions. Examples are [5], where SSJ results are used to analyze energy proportionality metrics and [18], which uses SERT results to train power prediction models.

Other works modify the SPEC SERT for custom tests. [15] modifies the SERT load level calibration to train a specific power model. [17], on the other hand, modifies the placement of worklets in order to utilize specific resources (CPU cores, etc.) on the SUT.

Finally, the ChauffeurWDK provides a framework to develop workloads and use for measurements with the SPEC power methodology. Custom research workloads for energy efficiency measurement have been deployed using ChauffeurWDK. E.g., [8] presents its own workload creation framework for energy-effiency testing, which is embedded into the ChauffeurWDK, whereas [9] deploys specialized worklets for memory testing in Chauffeur.

5 CONCLUSION

This paper sketches the SPEC Power and Performance Methodology and introduces the SPEC's compute efficiency benchmarks and tools. These tools are designed to help benchmark and rate the energy efficiency of a server system. The SPEC PTDaemon enables accurate measurement of power consumption, whereas the Chauffeur framework enables workload scheduling, placement, execution, and result collection. PTDaemon and Chauffeur can be used in the ChauffeurWDK to implement and test the energy efficiency of workloads in research and development. The SPEC SERT, on the other hand, already provides a wealth of workloads that can be used for research and server rating. Finally, the SPECpower_ssj2008 uses the methodology to execute application benchmarks that have driven energy-efficiency improvements for the last decade.

6 ACKNOWLEDGEMENTS

The authors wish to acknowledge current and past members of the SPECpower Committee and SPEC Research Power Working Group who have contributed to the design, development, testing, and overall success of SPECpower_ssj2008 benchmark, the SERT suite, and the Chauffeur WDK: Nathan Totura, Mike Tricker, Greg Darnell, Karl Huppler, Van Smith, Ashok Emani, Paul Muehr, David Ott, David Reiner, Karin Wulf, Cathy Sandifer, Jason Glick, Diana Cercy, and Dianne Rice, as well as the late Alan Adamson and Larry Gray. SPEC and the names SERT, Chauffeur, SPEC PTDaemon, and SPECpower ssj are registered trademarks of the Standard Performance Evaluation Corporation. Additional product and service names mentioned herein may be the trademarks of their respective owners.

REFERENCES

[1] Jeremy A. Arnold. 2014. Energy Efficiency Benchmark Framework (Chauffeur WDK). In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM, New York, NY, USA.

[2] C. Babcock. 2012. NY Times data center indictment misses the big picture. *InformationWeek Cloud* (2012).

[3] L.A. Barroso and U. Holzle. 2007. The Case for Energy-Proportional Computing. *Computer* 40, 12 (Dec 2007), 33–37. DOI: http://dx.doi.org/10.1109/MC.2007.443

[4] James Bucek, Klaus-Dieter Lange, and Jóakim von Kistowski. 2018. SPEC CPU2017 – Next-generation compute benchmark. In *ICPE*.

[5] C. H. Hsu and S. W. Poole. 2013. Revisiting Server Energy Proportionality. In *2013 42nd International Conference on Parallel Processing*. 834–840. DOI: http://dx.doi.org/10.1109/ICPP.2013.99

[6] K.-D. Lange. 2009. Identifying Shades of Green: The SPECpower Benchmarks. *Computer* 42, 3 (March 2009), 95–97. DOI: http://dx.doi.org/10.1109/MC.2009.84

[7] K.-D. Lange and Michael G. Tricker. 2011. The Design and Development of the Server Efficiency Rating Tool (SERT). In *Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering (ICPE '11)*. ACM, New York, NY, USA, 145–150. DOI: http://dx.doi.org/10.1145/1958746.1958769

[8] Norbert Schmitt, Jóakim von Kistowski, and Samuel Kounev. 2017. Emulating the Power Consumption Behavior of Server Workloads using CPU Performance Counters. In *Proceedings of the 25th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '17)*.

[9] Norbert Schmitt, Jóakim von Kistowski, and Samuel Kounev. 2017. Predicting Power Consumption of High-Memory-Bandwidth Workloads. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (ICPE '17)*. ACM, New York, NY, USA, 353–356. DOI: http://dx.doi.org/10.1145/3030207.3030241

[10] Standard Performance Evaluation Corporation. 2011. Power Analyzer Acceptance Process V1.3. (August 2011). https://www.spec.org/power/docs/SPEC-Power_Analyzer_Acceptance_Process.pdf.

[11] Standard Performance Evaluation Corporation. 2012. PTDaemon Design Document. (October 2012). http://www.spec.org/power/docs/SPEC-PTDaemon_Design.pdf.

[12] Standard Performance Evaluation Corporation. 2014. SPEC Power and Performance Benchmark Methodology. (December 2014). http://spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf.

[13] Standard Performance Evaluation Corporation. 2017. Power and Temperature Measurement Setup Guide. (January 2017). https://www.spec.org/power/docs/SPEC-Power_Measurement_Setup_Guide.pdf.

[14] Standard Performance Evaluation Corporation. 2017. Server Efficiency Rating Tool (SERT) Design Document. (September 2017). http://spec.org/sert2/SERT-designdocument.pdf.

[15] Christian Stier, Dominik Werle, and Anne Koziolek. 2017. Deriving Power Models for Architecture-Level Energy Efficiency Analyses. In *Computer Performance Engineering: 14th European Workshop, EPEW 2017, Berlin, Germany, September 7-8, 2017, Proceedings*, Philipp Reinecke and Antinisca Di Marco (Eds.). Springer International Publishing, Cham, 214–229. DOI: http://dx.doi.org/10.1007/978-3-319-66583-2_14

[16] Jóakim von Kistowski, Jeremy A. Arnold, Karl Huppler, Klaus-Dieter Lange, John L. Henning, and Paul Cao. 2015. How to Build a Benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015) (ICPE '15)*. ACM, New York, NY, USA. DOI: http://dx.doi.org/10.1145/2668930.2688819

[17] Jóakim von Kistowski, John Beckett, Klaus-Dieter Lange, Hansfried Block, Jeremy A. Arnold, and Samuel Kounev. 2015. Energy Efficiency of Hierarchical Server Load Distribution Strategies. In *Proceedings of the IEEE 23nd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2015)*. IEEE.

[18] Jóakim von Kistowski, Hansfried Block, John Beckett, Cloyce Spradling, Klaus-Dieter Lange, and Samuel Kounev. 2016. Variations in CPU Power Consumption. In *Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering (ICPE 2016)*. ACM, New York, NY, USA. DOI: http://dx.doi.org/10.1145/2851553.2851567

[19] Jóakim von Kistowski, Maximilian Deffner, Jeremy A. Arnold, Klaus-Dieter Lange, John Beckett, and Samuel Kounev. 2017. Autopilot: Enabling easy Benchmarking of Workload Energy Efficiency. In *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE 2017)*. ACM, New York, NY, USA. DOI: http://dx.doi.org/10.1145/3030207.3053667